Security against Adversaries with Privileged Access

Hayyu Imanda
Exeter College
University of Oxford

A thesis submitted for the degree of
DPhil in Cyber Security
Trinity 2023

Acknowledgements

Institutional

First and foremost, thank you to Kasper for supervising this thesis, and to the research group for all your support over the years. Thank you to Dr Wouter Lueks and Prof Andrew Martin who provided me with detailed feedback during the corrections phase of this thesis. Without each of them this work is incomplete.

To the CDT in Cyber Security community, thank you for reminding all of us that cybersecurity is interdisciplinary. To Julia Słupska, your work in cybersecurity as a network of care has inspired me since the start of our degrees. Special thanks to David Hobbs, an incredibly efficient, kind, and thoughtful CDT administrator who held us together and listened to each and every one of us when things got tough, to Janet Sadler for her finance whiz, and to Andrew Martin for his continuous belief in the programme. To Exeter College, from the academic administration team to the catering team, I have always felt taken care of within those walls.

To Sebastian Köhler, Simon Birnbach, and Richard Baker—postdocs in the next-door research group—thank you for taking the time out of your day in helping a lost PhD student who has very little coding experience, and be wonderful friends and lunch buddies in between.

Thank you to Exeter College, the CDT, the Jardine Foundation, and the Department of Computer Science for providing funds to attend conferences in which I am able to exchange ideas and present my work. Thank you to the University Covid Emergency Funds and Hardship Funds so I had less things to worry about.

Lastly, this DPhil would not have been possible without the ever-generous Jardine Foundation. I am immensely privileged to have been given this opportunity, and I will not be able to thank you enough.

Personal

It is impossible to contain all of my thanks within this space, but let's give it a go!

To complete a DPhil is an immense challenge on its own. To do one in the middle of a pandemic is somewhat bizarre. Thank you to Pak Aji and Ibu Feria, along with Mba Gadis, Mba Nini, Fajra and the rest of the staff at the Indonesian Consulate in Cape Town, for being a family away from home and giving me a space

that I can feel safe and attempt at being productive. Your kindness during a time I had no-one else to turn to and nothing to offer in return, is inspiring.

To Matt, Claudine, Sebastian, Yash—thank you for five years of friendship with neverending stories to tell each other, and to have stuck with me as I figure out my way around life. To Julia, Yung, Sruj, Nancy, Nayana, and Patrick: thank you for critical minds, thesis feedback, wholesome brunches, plant swaps, and being a support system I did not know I needed. To Marius and Cristian, thank you for sharing a joyful home with me as I settled back into Oxford, to Rangga for our 4-hour lunch catchups, and to Merry who always has an interesting tale to tell.

A significant amount of my time at Oxford was spent across different Oxford sports communities. To my teammates and committee members of OULTC, thank you for immeasurable amounts of joyful memories, lifelong friendships, and an opportunity to grow outside of academics. To the Blues Performance Team, thank you for keeping me fit even when we couldn't leave our rooms. To OUARFC for being such a wholesome group, thank you, and for welcoming me with wide open arms even though I didn't even know the sport existed.

To Hubert, who once told me to "never speak to someone with a philosophy degree", look where we are! Your brilliant mind has always helped me in articulating my thoughts, and made challenge myself and the assumptions in my work I took for granted. Thank you for your exemplar work ethic, neverending delicious food, and for being so incredibly patient with me and my chaos.

To my parents: Bunda and Yanda, siblings Dilla and Riam, thank you for always providing an environment of love and support, and never doubted me away from doing any of my questionable quests. I would never have gotten even close to being here without you, and there will never be any word or action to capture how much you mean to me. To the Madongs family, I always feel everyone cheering for me from afar. The hardest part of this DPhil was to be away from them. It really does take a village.

Abstract

In designing for security, one needs to specify an adversary model: what computations can they make and what do they have access to? What security guarantee can we achieve against that adversary?

Motivated by concrete scenarios, this thesis looks into adversaries that benefit from being in close physical proximity to the user, obtaining up to endpoint secrets. In addition, due to the locality of the adversary, they are also a Dolev-Yao adversary on the communications network. These adversaries also benefit from power imbalance between them and the user; due to this, security systems should be designed with care: users and adversaries are humans before principals in a security protocol.

From the system's point of view, these adversaries may be authenticated as the user, or have access to other information in the physical proximity of the user, hence having privileged access. This thesis designs security systems against two adversaries with such privileged access.

The first adversary, a surveillant adversary, has full access to the user's device output (e.g., visual access to the user's device screen), as well as the user's application layer data. Against an adversary with continuous presence, we design an architecture where the user can signal distress to a trusted third party through an online mechanism, without being detected by the adversary. Our second adversary, an intrusive adversary, has full access to the user's device for a limited amount of time: during that time, the system cannot distinguish between the user and the adversary. We propose an architecture where a user can guarantee data confidentiality during the period of control, and discuss authentication mechanisms to provide authentication of the user after the period, even though the adversary has compromised the entire device.

In each our security design, we provide a full security analysis and discuss our design decisions. We discuss how this thesis fits into the research landscape, and outline limitations and future work.

Contents

1 Introduction **1**

1.1 Contributions 4
1.2 Overview 5
1.3 Publications 6

2 Background **7**

2.1 What is an Adversary? 8
2.1.1 Privileged Access 9
2.2 A Crash Course in Cryptography 11
2.2.1 Mathematical Background 11
2.2.2 Cryptographic Algorithms and Protocols 12
2.2.3 Security Protocols 18
2.2.4 Distance Bounding 21
2.3 Notation and Assumptions 22

3 Signalling Distress in the Presence of a Surveillant Adversary **25**

3.1 Introduction 26
3.2 Design Goal 28
3.2.1 System Model 30
3.2.2 Adversary Model 32
3.3 Distress Deniability 33
3.4 Security Protocols 35
3.4.1 Server Enrolment 35
3.4.2 User Enrolment 36
3.4.3 Signalling Distress 38
3.5 Algorithms 41
3.5.1 Encoding and Decoding Distress 41
3.5.2 Bit Distribution 45
3.5.3 Choice of Public Key Encryption 46
3.6 Computational Overhead 48
3.7 Security Analysis 49
3.7.1 Server and User Enrolments 50

3.7.2 Distress Signal Protocol 51
3.7.3 Main Protocol 54
3.8 Discussion 57
3.8.1 TLS Client Hello or HTTP Header: A Comparison 57
3.8.2 Enrolment 57
3.8.3 User Interface 59
3.9 Conclusion 61

4 Achieving Confidentiality against an Intrusive Adversary 63
4.1 Introduction 63
4.1.1 Choice of terminology 67
4.2 Design 68
4.2.1 Design Goal 68
4.2.2 Adversary Model 70
4.2.3 System Model and Design 71
4.3 Nakula 75
4.3.1 User Setup 75
4.3.2 Session Start and Key Generation 76
4.3.3 Data Acquisition and Encryption 77
4.3.4 Session Lock 78
4.3.5 Key and Data Recovery 79
4.4 Security Analysis 80
4.5 Implementation 83
4.6 Conclusion 85

5 Authentication Environment against an Intrusive Adversary 87
5.1 Introduction 87
5.2 Design 89
5.2.1 System and Adversary Model 89
5.2.2 Protocol Considerations 89
5.2.3 Architectural Considerations 91
5.2.4 On Time-Based Authentication Mechanism 92
5.3 Authentication Mechanisms 95
5.3.1 Security Token 96
5.3.2 Verification Points 99
5.3.3 Remote Authentication With Trusted Entity 101
5.3.4 Dedicated Servers 106
5.4 Choosing f_{auth} 109
5.5 Conclusion 113

6 Related Work **115**
6.1 Strong Adversary Models 115
6.2 Undetectable Communications 119
6.3 Coercion Attacks 120
6.3.1 Distress Signalling 121
6.3.2 Plausible Deniability 122
6.3.3 Solutions to the Border Crossing Problem 124
6.4 Secure Deletion 126
6.5 Distance Bounding and Location Verification 126

7 Discussion **129**
7.1 Reflections and Limitations 129
7.2 Future Work 131

8 Conclusion **135**

A Further Cryptographic Building Blocks **137**
A.1 IND$-CPA Encryption Scheme 137

B Notation and Abbreviations **141**

References **145**

Cryptography rearranges power: it configures who can do what, from what

— Philip Rogaway in *the Moral Character of Cryptographic Work*

1 Introduction

When information is sent from one machine to another over the internet, it goes through numerous machines in between: the sender's wireless router, the local Internet Service Provider (ISP), and through the individual networks that form the internet before the process is reversed on the other side. Throughout this journey, there are multiple points at which an entity may abuse the process to obtain access to the sender's information: those sharing the same wireless router as the sender, those with the ability to request access to packets passing through an ISP, or in fact, any of the machines on the path. The Dolev-Yao model [61], though symbolic in nature, presents a realistic model that captures this situation: over the communications network, the adversary *carries the message* between the sender and receiver.

Many tools to achieve security goals have been designed in the past: encryption allows confidentiality so no unauthorised party can read the information, and signatures guarantee authentication to ensure that the participating party is who they claim to be. Reflective of Kerckhoff's principles [99], these systems use *secrets*; private encryption and signature keys ensure that access is given appropriately. The security of these systems relies on legitimate parties having appropriate access to these secrets, while the adversary does not.

The use of secrets is also ubiquitous in many settings outside of public network communications: passwords and credentials are widely used as an access control

mechanism for systems, and biometric information generates secrets that allow access to a device. At its core, security is based on asymmetric knowledge of secrets. However, practices such as password sharing occur more often than system designers would like to admit, and can be more common where there is a close personal relationships between two or more sharers [98, 151]. Multiple people can use the same device, by choice of convenience, necessity, or as a demonstration of trust [109], which, unknowingly or otherwise, leads to each person having access to the same keys rooted in the device's hardware. However, such asymmetry of knowledge is not static. Relationships and levels of trust may change over time, changing the way people share their access to digital devices due to a change in sharing of physical space. However, passwords are not always changed after a relationship dissolution [76]; in addition, home security smart lock systems don't consider a breakdown of relationships within their threat model [152], allowing for the possibility that former partners have access to the space. In other cases, secrets are available to others without the legitimate party's consent: breaches of databases containing passwords [88] or password cracking [31], are far from uncommon.

The situations above create a blurring of who system designers should consider as 'adversaries'; some may, by default or otherwise, have access to many secrets necessary for secure communications or access to a device, and are considered to have legitimate access to them from the viewpoint of a security system.

Now, what if the adversary is able to obtain knowledge of secrets through less technical means? If an adversary demands someone's personal identification numbers to their banking account with the threat of violence, many, understandably, would choose to provide it [45]. However, there are other threats that are more in the grey area of consent when it comes to providing someone else access to these secrets: first, in the United States and the United Kingdom, a customs officer is legally allowed to search through a traveller's personal electronics without a warrant, including requesting device passwords or social media information [51, 66, 89, 162]; this is far from unique to the two countries [47, 94, 96, 121, 160]. Though this in theory can affect all individuals, as oppposed to recommendations [117] some

demographics are more vulnerable to searches at borders and hence to the loss of confidentiality of the data they have on their device [36, 75, 144]; such a risk is also posed to those having certain occupations: journalists and human rights activists have been reported to being subject of the user of government spyware [110]. During a device search, the border officer has full control over the device: not only are they able to see the data stored on it, but they have the potential to keep any long-term secret key. Of course, an individual can refuse to comply with a border officer's request to unlock a device, but this comes with risk: they may confiscate the device or refuse entry if the individual is not a citizen of that country, with an impact on their immigration records. Hiding the data, as previous literature has suggested [167, 179], puts individuals at immense psychological pressure in having to take a risk in lying to a border agent with power to deny entry; not only does this impact on their wellbeing trivialised, but this action is also illegal [46].

Now consider an adversary that benefits from being in the same physical space as the user for a longer period of time. They may have access to, or control of, their secrets (through password post-it notes, being the system administrator of the common communications system, or as our previous example, by force), and may be able to—actively or otherwise—surveil a user due to their locality. For example, a survivor of intimate partner surveillance (IPS) is often in the same physical space as their abuser, who also may have full control of the network and access to all of their devices. A company may have multiple closed circuit television (CCTV) cameras around their office including near employee screens, and at the same time, owns the device that the user operates on. How can whistleblowers who have previously reported any wrongdoings, or survivors of IPS, communicate threats of harm should they be under such surveillance?

In the two cases discussed above, we see that the adversaries are able to gain access to privileged information, which may include communication secrets. We note there are a few things that they have in common: first, the adversaries benefit from locality with the user: they may have full visual access to whatever the user is doing on the device, and may even have direct access to the device themselves.

Second, that the adversaries benefit from power asymmetry: access to some secrets can be either expected, or provided because of an imbalance of power between the adversary and the user. At the same time, such imbalance may lead to consequences should the user challenge these access or capabilities.

With these in mind, we ask ourselves: what do we mean by *security* against such adversaries, and how can we achieve it?

1.1 Contributions

The first contribution of this thesis sees an introduction of two strong adversary models. We consider a *surveillant adversary*, an entity who does not have access to internal secrets but has a high degree of control over the user's device over an indeterminate length of time, including unlimited visual access to their device screen; that is, they see what the user sees. We also introduce an *intrusive adversary*, who has full access to a device for a finite amount of time, and is able to request the user to provide any authentication information. Both these adversaries are Dolev-Yao adversaries over some of the user's communications network. As the main contributions of this thesis, we design architectures where the user is able to achieve the following goals:

Ability to send a distress signal (Chapter 3). We propose a mitigation against a surveillant adversary, and we establish the notion of signalling distress to indicate that the user wishes to communicate to a trusted third party regarding a threat of harm. Due to the nature of the threat, but the mere existence of some communications may be considered suspicious by the adversary. To do this, our method uses existing websites to act as intermediaries between the user and a trusted backend, which enables the user to initiate the communication without arousing suspicion. On a technical level, we hide the distress signal by embedding it into the TLS handshake, which contains a client-chosen field of randomness so any website willing to participate can do so with minimal effort, and the adversary monitoring the traffic will only see TLS connections to legitimate websites.

Confidentiality (Chapter 4). We design Nakula, an ecosystem where data confidentiality against an intrusive adversary is possible, without demanding the psychological pressure (and illegality) of the user lying or hiding data from the adversary. The user is able to lock down data with a single click, allowing confidential physical data transport where the presence of an intrusive adversary is expected. To achieve this, the user temporarily loses the ability to access the data, and will need the assistance of a trusted third party to recover it.

Authentication (Chapter 5). As a direct follow-up work from Chapter 4, we consider the property of authenticating to a third party, where an intrusive adversary who has direct access to a device may try to do so. We discuss the properties necessary for such authentication to take place, and present four methods with varying advantages and disadvantages, and note that there is no one-size-fits-all solution.

1.2 Overview

We start with the background required for this thesis in Chapter 2, and we start with the following questions: what is an *adversary?* And what do we mean by *privileged access*?

Chapter 3 will introduce a *surveillant adversary* which models an adversary who has continuous visual access of the user's screen, and we propose an infrastructure for distress reporting to a trusted third party.

Chapter 4 will look into our approach to an *intrusive adversary*, who has full access to the device for a limited period of time, through having access to the user's authentication secrets. We construct an architecture where data confidentiality holds, without the user having to hide the data or lie to the adversary. We rely on a trusted third party that will only provide the necessary building blocks for decryption if they can be convinced that it is not requested by the adversary, and we classify potential methods of authentication in Chapter 5.

In Chapter 6 we discuss related work to illustrate where our work lies within the research landscape. We then proceed with a discussion of our thesis–including how,

though our work focuses on cryptographic solutions, it is never the case that these problems can be solved by technical work alone. We reflect and discuss limitations of our work, before discussing future work and concluding in Chapter 8.

1.3 Publications

The contributions of this thesis have resulted in the following papers:

- Hayyu Imanda and Kasper Rasmussen. 2023. Nakula: Coercion Resistant Data Storage against Time-Limited Adversaries. 2023. In *Proceedings of the 18th International Conference on Availability, Reliability, and Security (ARES '23)*. Association for Computing Machinery, New York, NY, USA, Article 4, 1-11. `https://doi.org/10.1145/3600160.3600175`. (Best Paper Award Runner Up)
- Hayyu Imanda and Kasper Rasmussen. 2023. Ask for Alice: Online Covert Distress Signal in the Presence of a Strong Adversary. *Preprint*. `arXiv:2310.03237`.

2
Background

Contents

2.1 What is an Adversary? 8
2.1.1 Privileged Access 9
2.2 A Crash Course in Cryptography 11
2.2.1 Mathematical Background 11
2.2.2 Cryptographic Algorithms and Protocols 12
2.2.3 Security Protocols 18
2.2.4 Distance Bounding 21
2.3 Notation and Assumptions 22

To design for security, we need to specify a few things. First, what are the adversary capabilities? What is the system model? And what are the goals for the user and the adversary? These will be specified in each chapter as we design our security mechanisms, but to start off this thesis, we start by discussing the terminology used in the thesis title: what do we mean by an *adversary*, let alone one with *privileged access*? We then proceed with some cryptographic background that the reader can refer back to as they proceed, and we summarise the notations and assumptions used throughout.

2.1 What is an Adversary?

According to the the National Institute of Standards and Technology (NIST), a US-based yet global-reaching agency which presents standardisation, frameworks, and qualifications, an adversary is a:

> "Person, group, organization, or government that conducts or has the intent to conduct detrimental activities" [118].

The definition above defines an adversary in terms of its *intent*, as well as its consequence. This is consistent with notion of adversary as a malicious entity commonly assumed [169], and reflective of the recurrent use of a red devil icon in computer security conference presentations[1]. The definition above also specify an adversary as an individual, or collection of individuals that form an institution, and not machines. Similarly, the Cyber Security Body of Knowledge (CyBOK) [157] presents a characterisation of adversaries who "perform malicious actions", with the characterisations based on the motivation of adversaries.

However, using intent as a requirement for defining adversary is not straightforward, when we consider, as the NIST definition above, adversary as humans interacting with the machines. Are we saying that a border officer, as a person, is *malicious* when requesting a device search, when they are simply obliged to do so as per their job description? Or as an institution: are we claiming that the systems and policies that push for device searches by a nation's border control agency malicious? Is someone who logs into a device to see their family member's location, to ensure their safety while walking home, malicious? Or has our thesis reversed the situation: is a whistleblower [126], or individuals who aim to defeat censorship tools [100], malicious, as there is consequence to 'national security'?

Defining adversary based on intent is inadequate, and doesn't sufficiently cover the diversity of situations we find in practice. In our thesis, following [104] and our work [90], we use the term adversary with no assumption about the adversary's

[1] As Philip Rogaway noted, cryptographers should stop using "cutesy adversaries" [133].

intentions, malicious or not; indeed, some interactions may be normal, or even expected from the adversaries.

On the other hand, many work in the cryptographic domain describe adversaries as algorithms [97], and characterise them based on their capabilities (e.g., Chosen Plaintext Attack adversary, Figure 2.2). Though our work is not on the symbolic front, the adversaries in this thesis are also *defined* by their capabilities.

In another NIST document, an adversary is an "entity that is not authorized to access or modify information, or who works to defeat any protections afforded the information" [53]. In this definition, an adversary described in terms of its *action*, in response to an access control mechanism that has been imposed on them.

Indeed, we second this: we define an adversary from a systems point of view, to describe an entity with specified capabilities, who we are defending our system against. The adversary is free to use up to their maximum capability to reach the adversarial goal, which is the negation of the user's security goal.

This section is not intended as a full exploration into the question (that would make another thesis on its own!), but to describe our use of the word and rid of incorrect assumptions, we highlight that it is a definition that we should not take for granted.

2.1.1 Privileged Access

We have mentioned the Dolev-Yao adversary, and we first discuss this in the model proposed [61], we assume the adversary carries the message, and can listen to any message that passes through the network, and act as a machine-in-the-middle: it can intercept any message, pretending to be the intended receiver; it can also impersonate any other entity, pretending as another sender. The adversary is also a legitimate user of the network, and can send messages under their name. However, we do not consider an attack on availability by a Dolev-Yao adversary; that is, they are not able to block the sending of messages through the network indefinitely. However, the model assumes that the entities (in terms of devices, we usually call

these the *endpoint*) are not compromised by the adversaries: this includes the secrets and the computations that occur within these devices.

Our thesis considers adversaries that are Dolev-Yao adversaries, with some additional access to endpoint secrets. This is not new: the Dolev-Yao model has long been considered insufficient especially when other channels are available [48]. Computer security courses [134] have taught undergraduates to question the model, in response to threats such as keyloggers. In addition, there have been numerous exciting and valuable work under *device* or *endpoint compromise*, with an adversary having access to the long-term key momentarily, with techniques such as key rotation implemented to allow *forward security*[2], where no messages before the compromise can be read or signed [79], or post-compromise security, where the adversary cannot see future messages or continue to be authenticated [43].

Our thesis is more than about access to these secrets. We consider the user and adversary as humans, not just machines operating in a protocol. They are characterised, as we have discussed in Chapter 1 by the following characteristics:

1. *Network control*: the adversary is a Dolev-Yao adversary on some of the user's communications network.
2. *Locality:* the adversary is physically present near the user, either in person or through the use of machines which creates a similar effect.
3. *Power asymmetry:* the adversary benefits from a power imbalance between them and the user, with adversary dominant in power.

Indeed, it is *because* of this locality and power imbalance that the adversary may have access to additional endpoint secrets. From the system's point of view, the adversary may see what the user sees, or even be authenticated as the user, obtaining a *privileged access* to the system. Note that as we introduce a surveillant adversary and an intrusive adversary later in this thesis, we do not claim that we have exhaustively considered all realistic adversary models that is characterised by the characteristics above.

[2] in trying to understand it, it's useful to think about what the reader thinks about what 'forward security' means, then reverse it.

When designing security against these adversaries, due to the power imbalance the user may be in a vulnerable position and under psychological pressure, and though our contributions are primarily on the protocol level, our security systems need to be designed with care.

2.2 A Crash Course in Cryptography

In this section, we cover the basic mathematical background and cryptographic primitives that will be used throughout this thesis. This sections only provides a high-level overview of these concepts, and we recommend the reader to consult [97] for a more thorough walkthrough of cryptography, and [147] for an introduction to elliptic curves. Readers who are comfortable with this topic may choose to skip to Section 2.3.

2.2.1 Mathematical Background

We start with some mathematical notation: an algorithm A is said to run in polynomial-time if there exists a polynomial p such that, for any input x of length $l(x)$, $A(x)$ terminates within at most $p(l(x))$. A probabilistic algorithm is one that is not deterministic; that is, we assume that the algorithm has access to a random bit at every step of the execution. A Probabilistic Polynomial Time (PPT) algorithm is one that is both.

A function f is negligible if for any polynomial p, there exists $N \in \mathbb{N}$ such that for all $n > N$, it holds that $f(n) > \frac{1}{p(n)}$. This is a fancy way of saying the polynomial goes asymptotically towards 0 as the input size increases. We denote by $\{0,1\}^n$ a bitstring of size n, and $\{0,1\}^*$ a bit string of arbitrary length.

A *group* $(G, \cdot)$ is a set G together with an operation $\cdot$ that satisfies certain properties: closure, associativity, existence of identity element, and existence of inverses. A group is *abelian* if $g_1 \cdot g_2 = g_2 \cdot g_1$ for all $g_1, g_2 \in G$. A group element g is said to *generate* G, if for every $h \in G$, repeating the group operation with g or its inverse we obtain h; in such a case, we write $\langle g \rangle = G$ and that G is *cyclic*. For example, $\mathbb{Z}_n$ is the (abelian) group of integers mod n under addition, generated

by the element 1. A *field* is a set F which with an additive operation $+$ and a multiplicative operation $\times$, where $(G, +)$ and $(G \setminus \{0\}, \times)$ are abelian groups.

Many groups and finite fields are used throughout cryptography. For example, points of an elliptic curve form a group under an interesting somewhat geometric operation [147], while the integers modulo p form a field under the standard addition and multiplication operations (this field is denoted $\mathbb{F}_p$, the finite field of p elements). In fact, all finite fields are of size p^r, where p is prime. Due to their finiteness and nice mathematical properties, certain computations to be performed efficiently by algorithms.

Elliptic Curves

Let $\mathbb{F}_q$ be a finite field of size q, with q a large prime power. An elliptic curve $\mathcal{E}$ over $\mathbb{F}_q$ can be written in the form $\mathcal{E}_{A,B} = y^2 + x^3 + Ax + B$, with $A, B \in \mathbb{F}_q$, with $B^2 - 4AC \neq 0$ together with a special point at infinity $\underline{\mathbf{o}}$. We denote $\mathcal{E}(\mathbb{F}_q)$ the set of $\mathbb{F}_q$-rational points of $\mathcal{E}$, that is, for $\mathcal{E} = \mathcal{E}_{A,B}$,

$$\mathcal{E}(\mathbb{F}_q) = \{(x, y) | y^2 = x^3 + Ax + B\} \cup \{\underline{\mathbf{o}}\}.$$

Hasse's Theorem [147] on the number of points of an elliptic curve states that for an elliptic curve $\mathcal{E}$ defined over $\mathbb{F}_q$, then $|\#\mathcal{E}(\mathbb{F}_q) - q - 1| \leq 2\sqrt{q}$. Note that if $x^2 = a$ in $\mathbb{F}_q$, then $(-x)^2 = a$. Combined with Hasse's theorem, the probability that $x \in \mathbb{F}_q$ lies on $\mathcal{E}$ is in fact approximately $\frac{1}{2}$. We will see why this is important in Chapter 3.

2.2.2 Cryptographic Algorithms and Protocols

In this thesis, we use multiple cryptographic building blocks to achieve our security. We treat most of these building blocks as black boxes, unless specified otherwise. We also introduce the security guarantees achieved from these black boxes.

Diffie Hellman Key Exchange

Diffie-Hellman Key Exchange was first introduced[3] in [59] without its explicit name, as a public key distribution system. This techniques makes use of the

[3]It was later claimed that the British intelligence agency GCHQ discovered the technique without publishing it to the public domain [139]

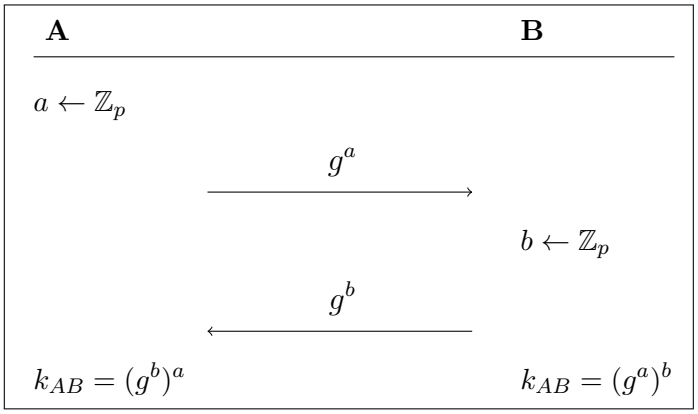

Figure 2.1: Diffie-Hellman key exchange protocol.

exponentiation function of a cyclic group G of p elements.The public parameters are a cyclic G with $|G| = p$ and $g \in G$ such that $\langle g \rangle = G$. In order for **A** and **B** to obtain a shared key, **A** first randomly chooses uniform $a \in \mathbb{Z}_p$ and computes g^a, and sends g^a over the public channel to **B**, who proceeds similarly by choosing $b \in \mathbb{Z}_p$, computing g^b, and sending g^b to **A**. The shared key is, as a group element, $(g^a)^b = (g^b)^a$. This is shown in Figure 2.1. We will see many instances of Diffie-Hellman key exchange throughout our thesis.

The security of Diffie-Hellman relies on the following three computational problems; in each case, let G be a cyclic group and $g \in G$ such that $G = \langle g \rangle$.

1. **Discrete Logarithm (DLP).** Given g^x, find, if it exists, the exponent x.
2. **Computational Diffie-Hellman (CDH).** Given g^x and g^y, compute g^{xy}.
3. **Decisional Diffie-Hellman (DDH).** Given g^x, g^y, and $h \in G$, decide if $h = g^{xy}$.

Throughout our thesis, we assume that DDH and CDH problems are hard, hence so is DLP.[4]

[4]Indeed, if one can solve the DLP, given the CDH problem, they can find the exponents x, y and simply compute g^{xy}. Similarly, when one can solve the CDH problem, then they can compute g^{xy} and simply compare with h.

Encryption

An encryption algorithm is usually used for *confidentiality*: no-one but legitimate parties are able to read the message. Note that encryption only makes the original message look 'scrambled', and not understandable by anyone else that does not have access to the relevant decryption key; encryption does not hide the fact that a message has been sent. Encryption can be symmetric or asymmetric, as we describe below.

A symmetric (sometimes called, confusingly, private-key) encryption scheme consists of three PPT algorithms $(\textsc{KeyGen}, \textsc{Enc}, \textsc{Dec})$ such that:

- KeyGen takes a security parameter n and outputs a key k.
- Enc takes input a key k and a plaintext m and outputs a ciphertext $c = \textsc{Enc}_k(m)$.
- Dec takes input a key k and ciphertext c and outputs $m = \textsc{Dec}_k(c)$. For correctness, it is required that $\textsc{Dec}_k(\textsc{Enc}_k(m)) = m$.

Similarly, an asymmetric key encryption scheme consists of three PPT algorithms $(\textsc{KeyGen}, \textsc{Enc}, \textsc{Dec})$ such that:

- KeyGen takes a security parameter n and outputs a keypair (pk, sk). We denote pk as the *public key* used for encryption and sk as the *private key*[5] used for decryption.
- Enc takes input a *public key* pk and a plaintext m and outputs a ciphertext $c = \textsc{Enc}_{pk}(m)$.
- Dec takes input a *private key* sk and ciphertext c and outputs $m = \textsc{Dec}_{sk}(c)$. It is required that, if (pk, sk) is a valid keypair output by KeyGen, then $\textsc{Dec}_{sk}(\textsc{Enc}_{pk}(m)) = m$ except with negligible probability.

[5] see the confusion?!

Intuitively, an encryption scheme (symmetric or asymmetric) is semantically secure if a PPT adversary cannot distinguish the encryptions of two plaintexts using the same (encryption) key[6]. We consider now a chosen plaintext attack (CPA) adversary:

Definition 1. Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec})$ be a public key encryption scheme. Consider the indistinguishability against CPA adversary IND-CPA$_{\mathcal{A},\Pi}(n)$ is described in Figure 2.2.

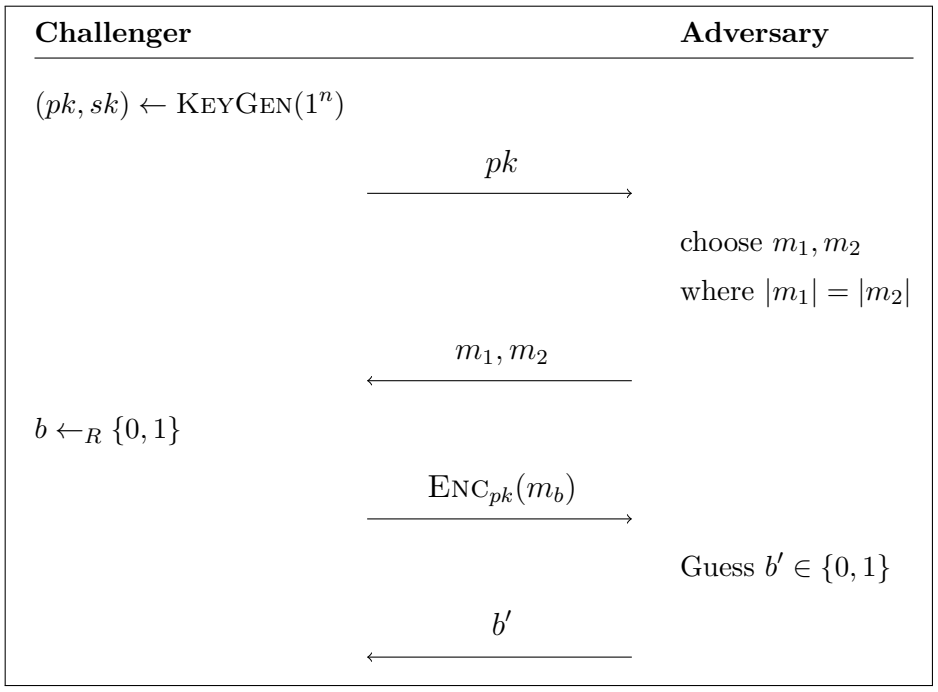

Figure 2.2: The indistinguishability against chosen-ciphertext attack experiment IND-CPA$_{\mathcal{A},\Pi}(n)$ for a public key encryption scheme $(\text{KeyGen}, \text{Enc}, \text{Dec})$.

The definition above describes CPA security, which is used throughout this thesis. Appendix A.1 also includes other security games such as ExpIND$-CPA, to determine whether a ciphertext is indistinguishable from random, should the reader be interested.

[6]This is usually considered to be the definition of indistinguishable encryptions in the presence of an eavesdropper, but with the formal definition of semantic security in [97, Definition 3.12], it is also shown in [97, Theorem 3.13] that they are equivalent.

Elliptic Curve El Gamal

We describe the Elliptic Curve El Gamal (ECEG) public key encryption. The public parameters are:

1. a finite field $\mathbb{F}_q$ and elliptic curve $\mathcal{E} : y^2 = x^3 + Ax + B$ defined over $\mathbb{F}_q$.
2. A point $P \in \mathcal{E}(\mathbb{F}_q)$.

The receiver chooses $a \in \mathbb{F}_q$ as their secret key and publishes aP. For a sender to send a plaintext M (for simplicity, M is an elliptic curve point. In practice, there is an encoding function from bitstring to an elliptic curve point before the algorithm starts) to the receiver, they compute the following:

1. KeyGen: choose $k \in \mathbb{F}_q$ as private key and compute $k(aP)$ as public key.
2. Enc: compute $Q = kP$ and $R = M + k(aP)$. The ciphertext is (Q, R).

Upon receipt, the receiver simply computes $\text{Dec}((Q, R)) = R - aQ$. Correctness is immediate. In addition, ECEG is CPA-secure under the Elliptic Curve Decisional Diffie-Hellman assumption.

Note that the ciphertext has two group elements, while the plaintext has one. It then follows that in its original form, ECEG has a 1:2 expansion of plaintext to ciphertext [113]. However, to save space, the sender instead of sending $Q = (x_Q, y_Q)$ and $R = (x_R, y_R)$ can simply send (x_Q, b_Q) and (x_R, b_R), where b_i are binary values denoting which square root the y_i is.

Hash Functions, MACs, and Digital Signatures

Encryption guarantees confidentiality against a Dolev-Yao adversary, as the adversary is not able to read the message m. However, a Dolev-Yao adversary is also an *active* attacker, and can modify messages in transit. When a ciphertext c is sent from the sender to the receiver, an adversary can simply replace c with any other c', and the Receiver will not only not receive m, but also not realise that the message was modified in transit.

We now need *integrity*: a security notion that something is not tampered by the adversary; what the receiver receives is indeed what was sent by the sender. We also need *authentication*, where an entity is actually who they say they are.

First, we define a (cryptographic) *hash function*, an efficiently computable deterministic $h : \{0,1\}^* \to \{0,1\}^n$; in other words, it takes a string of arbitrary length and maps it to a string of finite length. Of course, the pigeonhole principle states that there must be *collisions*, as the codomain is smaller than the domain. When we assume that a hash function is 'secure', we mean the following:

- *Preimage resistance:* Given any y in the image of h, it is computationally hard to find x such that $h(x) = y$.
- *Collision resistance:* It is computationally hard to find any pair x, x' where $x \neq x'$, such that $h(x) = h(x')$.

We also define a message authentication code (MAC), which intuitively, can be described as a hash function with symmetric keys. For now we assume that a sender and receiver share a symmetric MAC key k. A MAC with a key k is a function MAC_k that takes as input a message m, and outputs a relatively short tag. The sender sends both m and tag to the receiver, with a function $\mathsf{VrfyMAC}$ that takes a message m' and tag'. The receiver computes a tag (sometimes we simply call a 'MAC') with input m' and key k, and outputs accept if and only if $m' = m$ and $\mathsf{tag} = \mathsf{tag}'$. We assume a MAC is secure if it is not computationally possible to create a valid tag for an entity who does not have access to k. A MAC provides integrity: any changes to the message m or tag in transit will be detected by the receiver. It also provides authentication: only the entities who have access to that key is able to compute the correct tag.

Lastly, we introduce a *digital signature*, which is similar to a MAC, but is one of asymmetric form, with a secret signing key and public verification key. This means that any person can verify a signed message using the public verification key, but only the sender can create the signature from the message. We say a digital signature algorithm is secure when no-one can forge a message without

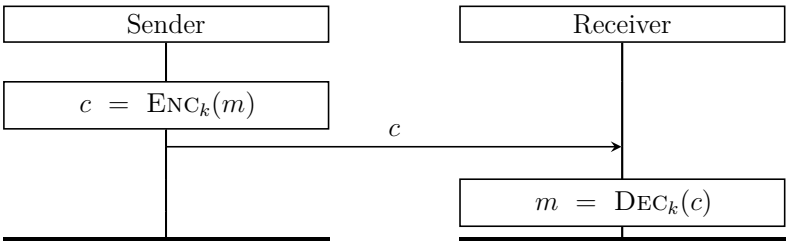

Figure 2.3: An example of a basic protocol guaranteeing confidentiality between two principal, the Sender and Receiver. We assume the Sender and Receiver have access to a shared symmetric key k. The Sender, wishing to send a message m, first computes the ciphertext $c = \text{Enc}_k(m)$ and sends the ciphertext to the receiver. The receiver, upon receiving c, obtains back m by computing $m = \text{Dec}_k(c)$. Note that under the Dolev-Yao model, the adversary has access to c, but not k or m.

the correct signing key. As expected from its name, a digital signature provides authentication, but also integrity.

A certificate authority (CA) is typically used to distribute keys for digital signatures, including for TLS connections (Section 2.2.3) for use over the internet. The participating entities are provided with the private signing key and a public certificate that is linked to the private key. In this thesis we shall assume that the CA is honest, and only provides legitimate keypairs and certificates to legitimate entities.

2.2.3 Security Protocols

Once we have the building blocks ready, we can use them to construct *protocols*: these are the rules governing a communication between two or more entities, which allow them to understand each other and describes the sequence of actions that they take.

In this thesis, we primarily use a protocol diagram (for example, see Figure 2.3) to describe our protocols, top-to-bottom, left-to-right. In the Dolev-Yao model [61], the adversary carries the message between the sender and receiver, however has no access on the secrets or computations that occur on each end.

Throughout this thesis, we use Enc and MAC in combination within one protocol message. Though this is possible in theory along with their security guarantees, we want to stress that in practice, authenticated encryption [**?**] is normally used.

We include below the Transport Layer Security protocol, the Hypertext Transport Protocol, and Distance Bounding; this will later be discuss in Chapter 3 and Chapter 5..

The Hypertext Transport Protocol

The Hypertext Transport Protocol (HTTP) [64] is a request-response protocol over the application layer between two parties. There are two types of HTTP messages: requests and responses.

Requests loosely consist of the following five elements: a HTTP method (e.g., self-explanatory GET or POST), the location of the resource to fetch (through url, for example), the HTTP protocol version used, the HTTP headers, and the HTTP body. A HTTP header consists of a string followed by a colon, and a value whose content and structure depends on the header; headers may contain information about the format of the body, or any additional information. For examples, see Figure 2.4. The HTTP body, or the payload (if necessary for the connection), may include files to be sent.

A response, as its name suggests, responds to a HTTP request and has a similar structure. It consists of the protocol version, an indication of whether or not the request was successful, a status message, HTTP headers (similar to those in requests), and if necessary, a body containing the resource requested.

Custom headers (in both requests and responses) can be included within a HTTP connection, and this can be included in communication involving a specific webserver with specifications that are unique to that webserver. For example, current communication from Google Chrome with any Google-owned domain includes the header X-Client-Data. Indeed, the prefix X- describes a non-standard, custom header.

This client-server model allows HTTP to be the basis of communications over the internet, between web browsers (the *client*) and webservers (the *server*).

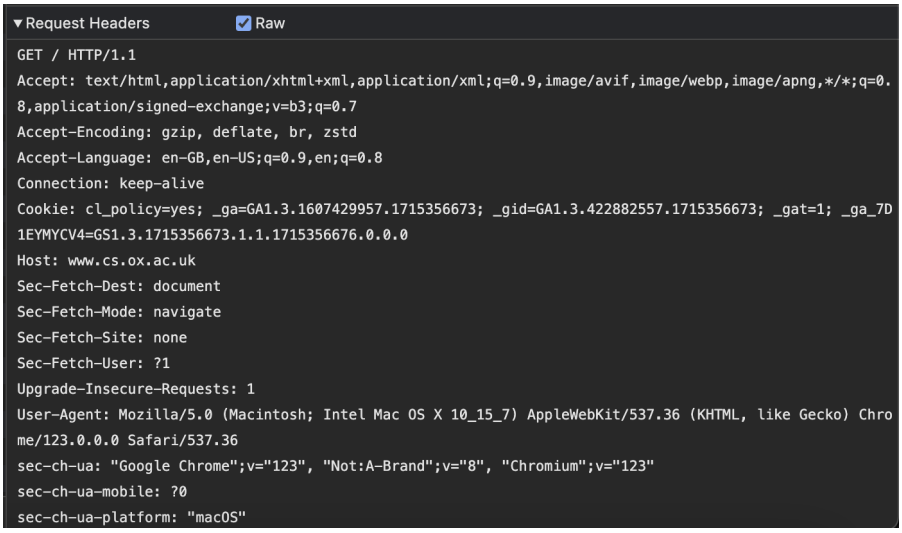

Figure 2.4: An example of HTTP Request Headers between the author's browser to `www.cs.ox.ac.uk`. The screenshot shows that the browser sends a HTTP GET request, with protocol version 1.1. Fifteen headers are shown, including "Upgrade-Insecure-Requests", which means that the client has a preference for an encrypted and authenticated response from the server.

The Transport Layer Security

The standardised Transport Layer Security (TLS) protocol [69] is a transport layer protocol which allows server-side authentication and construction of a secure session. It has evolved over the years, having built on the deprecated Secure Socket Layer (SSL); at the time of writing, most browsers and sites support TLS 1.2, and TLS 1.3, the former published in 2008 and the latter in 2018.

Over the internet, TLS is used together with HTTP to form Hypertext Transport Protocol Secure (HTTPS), with servers proving their authenticity using their digital certificates issues by a trusted certificate authority.

Before a secure session can be established, the client (in most cases, a browser) has to agree on secret keys with the server. This is done through a TLS *handshake*, and ensures freshness (i.e. it is not a replayed previous communication), the TLS protocol uses a *nonce*, which stands for a "number only used once"; as so, these numbers are indistinguishable from random. The client and server exchange messages, including the server proving that they have the correct signing key of their digital certificate. At the end of the handshake, the client and server has

a secure session where they can communicate with guaranteed freshness, (server) authentication, and confidentiality against Dolev-Yao adversaries.

After the handshake is completed, the TLS session begins, and within this session, a HTTP connection takes place. Not only is the server is now authenticated, but the contents of the HTTP connection (including header, url, and body) is encrypted and hence confidentiality holds against a network adversary.

2.2.4 Distance Bounding

A distance bounding protocol was originally proposed by Brands and Chaum [35] to determine an upper bound on the physical distance between two parties. The protocol involves a prover P and a verifier V, with V aiming to verify both P's identity, as well as that P is *near* V. The original proposal involves round-trip bit exchanges between V and P, with V measuring the time it takes for P's response to arrive back, which provides an upper bound on the distance between P and V, though there are many other proposals since. We refer the reader to the following surveys for more detail [18, 40].

The protocol usually composed of three phases: in the *slow phase*, the parties agree on the building blocks, including nonces and identification for the purposes of key lookup. The *distance-bounding phase* follows, which consists of multiple challenge-response rounds, and the verifier measures the durations that it takes for the prover to respond. Lastly, the *verification phase* takes place, where the verifier checks of the user's authenticity as well as making a decision on whether the user is within the bound. In this thesis, we consider a distance bounding protocol as a black-box, with all the three phases within.

The types of attacks that are usually considered in the distance bounding literature are the following.

- *impersonation fraud*, where an adversary pretends to be the legitimate prover
- *distance fraud*, where a dishonest prover convinces the verifier that they are at a different distance that actual.

- *mafia fraud*, where the adversary acts as a machine-in-the-middle between the prover and the verifier.
- *terrorist fraud*, where the dishonest prover colludes with another adversary that is closer to the verifier, to convince the verifier of a wrong distance to the prover.
- *distance hijacking*, where a dishonest prover convinces the verifier that it is a different distance than it actually is, by exploiting the presence of an honest prover.

2.3 Notation and Assumptions

Throughout this thesis, we denote k_{AB} as a symmetric key between two parties (usually A and B). In our protocols, we seem to use the same key k_{AB} for both encryption and MAC: indeed, to simplify our protocol, we denote k_{AB} as a key derived from the key derivation function (KDF) derived from the same key k_{AB}.

Unless stated otherwise, in our protocols, we denote by MAC_k to be mean $\text{MAC}_k(x)$, where we write x are the protocol contents directly preceding the MAC, e.g. $\text{Enc}_k(m), \mathsf{nonce}, \text{MAC}_k$ is equivalent to $\text{Enc}_k(m), \mathsf{nonce}, \text{MAC}_k(\text{Enc}_k(m), \mathsf{nonce})$ where k are encryption and MAC keys derived from a key derivation function with seed k. A list of notations used as well as abbreviations can be found in the Appendix.

We denote by the 'user' being the entity we wish to design our systems for and whose goals we want to satisfy, and a 'adversary' being an entity we wish to protect against. As we have described, the term adversary does not immediately correlate with malicious intent. We interchangeably use the term 'user' and 'adversary' to mean both the user and adversary as humans, but also the corresponding machine or device. That is, when an action is taken by the user (respectively adversary), we can mean both the user performing an action as a human, but also the device performing a particular task automatically. We hope that this difference is clear in the context it is described in.

Throughout this thesis, we assume the Dolev-Yao model [61] – that is, we assume that communication happens over a totally insecure channel with no

confidentiality and integrity guarantees and the adversary is able to actively alter the messages. Although stealth is sometimes our goal, we remain guided by Kerckhoff's principles [99] and assume the system is public. We do not consider an attack on availability.

We also note that throughout this thesis we use the word 'we' to refer as the reader and myself as the writer, but for readability also use the word 'we' to mean 'I' (the writer).

3

Signalling Distress in the Presence of a Surveillant Adversary

Contents

3.1 Introduction 26
3.2 Design Goal 28
3.2.1 System Model 30
3.2.2 Adversary Model 32
3.3 Distress Deniability 33
3.4 Security Protocols 35
3.4.1 Server Enrolment 35
3.4.2 User Enrolment 36
3.4.3 Signalling Distress 38
3.5 Algorithms 41
3.5.1 Encoding and Decoding Distress 41
3.5.2 Bit Distribution 45
3.5.3 Choice of Public Key Encryption 46
3.6 Computational Overhead 48
3.7 Security Analysis 49
3.7.1 Server and User Enrolments 50
3.7.2 Distress Signal Protocol 51
3.7.3 Main Protocol 54
3.8 Discussion 57
3.8.1 TLS Client Hello or HTTP Header: A Comparison . . . 57
3.8.2 Enrolment 57
3.8.3 User Interface 59
3.9 Conclusion 61

3.1 Introduction

In the United Kingdom, the *Ask for Angela* campaign [159] allows any individual who feels unsafe an exit from the establishment they are in by asking for a fictional member named Angela, prompting a trained member of staff to ensure a safe passage and potentially call the authorities. Similarly, major pharmacies across the UK have taken part in *Ask for Ani (Action Needed Immediately)* [114], where the pharmacy would provide a consultation room for people experiencing domestic abuse. These campaigns are aimed to allow survivors of domestic abuse to systematically be able to inform someone of their situation in a discreet manner, in a physical space that is common for them to visit.

During the Covid-19 pandemic, there has been an increase in demand for domestic abuse services, which indicate an increase in the severity of abuse being perpetrated while survivors were not able to leave home [119]; EU Member States experienced a similar trend [108]. While some regions in the US have reported that the number of calls received by domestic violence hotlines have dropped, experts believe that this is not a reflection of a decrease in the number of cases but rather that survivors were unable to safely connect with services while they are in the same space as their abusers [63]. A solution where survivors can safely reach out from their own homes can be valuable, however this is not a trivial task.

In this paper, we present an adversary model where the adversary has a high degree of control over a user's device and complete control over the local network. Our model allows an abstraction of real-life scenarios, where an adversary has visual access to the user's device while the user wishing to request help—this can be through sharing a physical space or the adversary's ability to monitor the screen through a camera. In addition to the above, this may include travellers who have been required to enter their authentication information at a hostile nation's border [121], or an intelligence agent who has to covertly inform of their capture while being in full surveillance by their captors [83].

We present a scheme where the fundamental goal is for a user to signal distress to an online entity. In certain situations, if communication dedicated to signalling distress is detected, it can lead to further consequences from the adversary. Hence, it is important that the system hides the fact that the user is signalling distress. We accomplish this using several different techniques: firstly, we rely on (existing) webservers to enrol in the scheme so they can act as intermediaries in communication between the user and the backend – this goes a long way towards making the connection seem innocent even if the device is under observation. Secondly, to make sure the distress signal is undetectable on the network, we embed the distress signal by utilising protocol objects within HTTP and TLS protocols. This ensures that a signal can be sent without the adversary's knowledge while not compromising the underlying TLS session. We formalise our security notion using a security game, and with the formal definition of security and the adversary model we are able to provide a thorough proof of the security of our protocols.

The contributions on this chapter are as follows:

- We introduce a surveillant adversary $\mathcal{A}_{\text{SUR}}$, who in addition to being a Dolev-Yao adversary on the network, also has visual access to the user's device screen as well as all the user's application layer data.
- We create an infrastructure for distress reporting, where one central entity receives distress signals along with accompanying information, as well as webservers who voluntarily participate acting as entry points.
- Within the distress infrastructure, we introduce enrolment protocols for the user and the webserver, as well as the main protocol with which the distress signal is relayed by the webserver to the central entity.
- We introduce encoding and decoding functions to hide the distress and necessary information while being sent to the webserver with the presence of $\mathcal{A}_{\text{SUR}}$.
- We present two options for the output of the encoding function to be embedded in: (1) the client nonce within the TLS handshake, and (2) a custom header

object within HTTP request. We discuss the advantages and disadvantages of each method.

The fundamental goal of our scheme is for the user to signal distress to an online entity, where enrolment is possible beforehand. Our design is on the protocol level; we stress that the follow-up actions by the online entity is very important and require further design, however this is outside the scope of this specific work. There are many non-technical facets of our scheme that are out of scope for this chapter, including the resolution of the situation after the distress is received. Further, our scheme discusses the protocols that occur behind the scenes of the user initiating the distress; the human interaction with the device needs to be designed in a proper, user-friendly manner. These issues are incredibly important but are considered future work, however we highlight some possibilities in Section 3.8.

3.2 Design Goal

One trivial attempt in designing our scheme is for the user to submit an online, encrypted communication to a particular trusted third party (e.g. the Met Police website). However, the adversary knows the recipient of the message: the adversary has access to the network whilst HTTP/TLS reveals the recipient's IP address, or they simply reads the URL from the user's device. Given so, the very presence of such communication being sent might raise suspicion. The adversary has full control of the network and therefore is able to block certain websites, including those that publicly aim at supporting groups that the user may be a part of.

Our approach shares some similarities with *Ask for Ani* and *Ask for Angela*, namely that we depend on participation of existing establishments to be the primary receiver of the distress signal, before relaying them. In particular, we believe well-known websites to be well placed to be this intermediary and therefore advocate for them to adopt such a system. A wide enough adoption would eliminate suspicion based on the recipient of communication, because the user can choose which

participating websites would minimise suspicion. The websites will have to undergo an enrolment process, and ideally this enrolment may be open on a rolling basis.

This participation from websites in the scheme is necessary for the system to work and to minimise suspicion from the adversary. We believe that there are few drawbacks of such a scheme for websites. In particular, we argue that our design does not generate a large amount of overhead in computation and communication for the website.

Note that in *Ask for Ani*, a domestic abuse survivor may walk to any participating pharmacies, even with the presence of the abuser, and ask for "Ani" (Action Needed Immediately) to a staff member who will give them access to a safe space and contact the relevant authorities. Similarly, the fictional staff "Angela" may be requested in a bar, or a "pizza" ordered when in fact this call was made to the police and not a pizza takeaway [23]. These offline attempts require the use of a codeword that the receiving party can understand or infer about the speaker's intentions. In network communications, this translates to the user signalling distress by sending messages that may look unsuspicious, but has been agreed upon with a trusted third party that this is a cry for help – e.g., uploading a particular text, or *codeword*, on social media. Even without the adversary knowing the codeword, this scheme does not give enough guarantees that a cry for help will not be detected; codeword or codephrase that appears to be out-of-place or irrelevant to ongoing events may also trigger suspicion – such inconsistency of expected content is discussed as *social indistinguishability* as proposed in [24].

Note that the adversary may perform statistical tests to detect when a particular communication is sent. Other methods in including hidden information on social media (for example, through hiding data on a text's white space [42, 146], or hiding messages in images [39]) can still be distinguished by the adversary. If this scheme is to be deployed with the target of reaching as many potential users that may benefit from this, we need assume that the system we design is public, and the adversary only lacks access to several secret components (also known as Kerckhoff's principle).

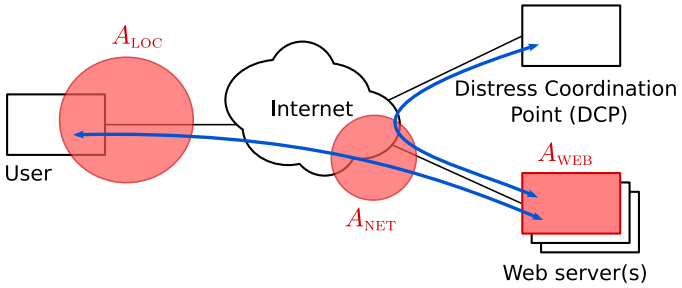

Figure 3.1: System Model. The user communicates with a webserver to signal distress. The webserver in turn forwards the message to DCP who initiates the required actions. $\mathcal{A}_{\text{SUR}}$ is assumed to be absent during user enrolment.

To ensure stealth, we utilise the HTTP and TLS protocols with enrolled websites (and as such, users connecting via HTTP or HTTPS can be accommodated). HTTP(S) are widely deployed and we minimise the complexity of the scheme deployment on a large scale. In addition, we use public key encryption for the user to communicate with the webserver for scalability. Our strategy is to encode this information into communication objects that are unpredictable and random-looking in the underlying protocol. Our first proposal specifically embeds the distress within the client nonce within the TLS handshake, and our second relies on adding a custom header object within HTTP. Both are values chosen on the user side, as well as having a large enough size to include necessary information that can be understood and relayed by the participating websites. Its unpredictability means that we can propose a way such that the user can plausibly deny sending a distress signal.

3.2.1 System Model

The three principal players of the system are as follows , and are shown in Figure 3.1.

1. The *user*: the player trying to signal distress to a trusted party. The user's goal is to communicate a distress signal to the trusted party without the adversary's knowledge.
2. The *distress coordination point* (DCP): a trusted party who receives distress signals. We assume that DCP is always honest.
3. The *webserver*: a webserver who relays the user's distress signal to DCP. To take part in the scheme, the webserver has to go through an enrolment

process, and user chooses which webservers to send the distress signal to. We do not, however, assume that the webserver is always honest.

Any webserver may sign up to participate, though it is up to the user to decide the webserver to send a distress signal to if the choice of websites is plenty (and this amount is up to the system designer). Should there be many options, for the user, this decision can be made primarily based on two things: the trustworthiness of the website and minimising suspicion from $\mathcal{A}_{\text{SUR}}$ when visiting the site.

The trustworthiness of the website can be judged in many ways, though the user might choose a more popular sites over those with less reputation to lose. To minimise suspicion from $\mathcal{A}_{\text{SUR}}$, the user can choose websites that they regularly visit or a popular site they would have a legitimate reason to visit in any circumstance; again, this depends on the specific scenario: during a border check that requires the user to open their email or social media site, these sites may be chosen during enrolment.

In addition, we assume that the user, webserver, and DCP are all able to make a TLS connection during enrolment and we assume that the user can communicate freely with the webserver and the DCP at all times.

Our system operates on three steps:

1. the *server enrolment*, where the webserver communicates with DCP so the server is added to the *server database*, as well as to facilitate the exchange of necessary encryption keys.
2. the *user enrolment*, where the user communicates with DCP so the user is added to the *user database*, for the user to choose the webservers they trust, as well as the exchange of identifier and encryption keys.
3. the *distress signal protocol*, which, as the name suggests, the user uses to signal distress.

Though these steps are separate and somewhat sequential with the server enrolment having to have occurred before user enrolment and the user would need to be enrolled before being able to signal distress using the main protocol, it is

important to note that new webservers and new users can enrol at any time. In addition, webservers and users who have enrolled before are able to update any enrolment details at any time.

3.2.2 Adversary Model

We consider three different adversaries located in different parts of the system, each with different capabilities and goals:

1. A network adversary $\mathcal{A}_{\text{NET}}$, an external Dolev-Yao type adversary. The goals of $\mathcal{A}_{\text{NET}}$ is to violate message confidentiality and message integrity. For example to understand which users are signalling distress, or falsely trigger distress on behalf of an honest user.
2. A malicious webserver $\mathcal{A}_{\text{WEB}}$, whose goals are similar to $\mathcal{A}_{\text{NET}}$ but $\mathcal{A}_{\text{WEB}}$ who is acting as a proxy for the user's messages and is expected to relay communication to the DCP.
3. $\mathcal{A}_{\text{SUR}}$, the *surveillant adversary.* $\mathcal{A}_{\text{SUR}}$ has control over all of the user's communication channels (hence, can access application layer content of TLS and HTTP through a packet sniffer or a TLS termination proxy) and passive access to the user's output device (e.g., visual access to a computer monitor; the adversary sees what the user sees). We only assume that the internal state of running applications, including the user's secret keys, can be kept secret, and that $\mathcal{A}_{\text{SUR}}$ is not present during the user enrolment process and will not have any access to past enrolment communication. The main goal of $\mathcal{A}_{\text{SUR}}$ is to detect a distress signal when it is sent. We make this more precise in Section 3.3.

We consider $\mathcal{A}_{\text{SUR}}$, $\mathcal{A}_{\text{NET}}$, and $\mathcal{A}_{\text{WEB}}$ as three independent entities. Correlation attacks with colluding adversaries, for example by traffic or timing analysis, are outside the scope of this chapter, as are attacks on availability.

3.3 Distress Deniability

In this section, we introduce a security definition to capture the situation in which we illustrate how $\mathcal{A}_{\text{SUR}}$ can use their capabilities in order to detect a distress signal, with the user able to plausibly deny existence of a distress call. We first define a *distress hiding* function to describe a family of functions in which a distress as well as some extra information can be used as an input to a function, but is also somewhat reversible.

Definition 2. For $d \in \{\mathsf{true}, \mathsf{false}\}$, and x from a finite domain, a function $f_N(d, x, k)$ is a fixed-length reversible function if:

1. $|f_N(d, x, k)| = N$
2. Except with negligible probability, there exists $g(y, k')$ such that

$$g(f(d, x, k), k') = \begin{cases} d, x & \text{if } d = \mathsf{true} \\ d & \text{if } d = \mathsf{false} \end{cases} .$$

The definition above simply states that the scheme preserves a binary value d as well as some additional information x if $d = \mathsf{true}$ that is reversible by using some key (k, k'). In addition, such function has a fixed output size N. Of course, the trivial mapping would satisfy the definition, but we also need it to satisfy some security: for our scheme to be secure, we require the function to not only preserve the distress, but *hide* it amongst other connections. Given that the adversary has access to HTTP/TLS connections, they may have recorded a polynomial amount of client nonces or HTTP headers in normal communication when the user is not intending to signal distress. We formalise this in the following security game.

Definition 3. Let f_N be a fixed-length reversible function, x_i a finite string, and pk a public key for a public key encryption scheme. Consider the following experiment $\text{ExpDenDistress}_{\mathcal{A}}^{f_N}(n)$:

Challenger		**Adversary**
	$\xleftarrow{x_1, \ldots, x_n}$	
$(pk, sk) \leftarrow \mathcal{K}$		
$b \leftarrow_R \{0, 1\}$		
Construct $r_1, \ldots, r_n$		
	$\xrightarrow{pk, r_1, \ldots, r_n,}$	
		Guess $b' \in \{0, 1\}$
	$\xleftarrow{b'}$	

The challenger constructs $r_1, \ldots, r_n$ as follows:

1. if $b = 0$: $u_i \leftarrow f_N(\mathsf{false}, x_i, pk)$ for each $i = 1, \ldots, n$,
2. if $b = 1$: D consists of $n - 1$ integers $r_i \leftarrow f_N(\mathsf{false}, x_i, k)$ and for exactly one $j \in \{1, \ldots, n\}$, $r_j \leftarrow f_N(\mathsf{true}, x_j, pk)$

Define the advantage of $\mathcal{A}$ on f_N to be:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{DenDistress}, f_N}(n) = |\Pr[\mathcal{A} \text{ outputs } 1 | b = 0] - \Pr[\mathcal{A} \text{ outputs } 1 | b = 1]|.$$

We say that a fixed-length reversible function $f_N(x, y)$ is DenDistress secure if for all probabilistic polynomial time adversary $\mathcal{A}(n)$, there exists a negligible function $\epsilon(n)$ such that $\mathrm{Adv}_{\mathcal{A}}^{\mathrm{DenDistress}, f_N}(n) \leq \epsilon(n)$.

The experiment above describes the scenario where the adversary is able to receive a distribution of n nonces, and to guess whether or not it contains a distress call, which accurately captures our scenario as the vast majority of online connections will be normal communications not signalling distress. Hence, we say that a distress hiding function is *deniable* because the user can plausibly deny sending a distress signal within the distribution. When given any amount of nonces, we want to create a scheme where the adversary will not be able to correctly guess whether or not a distress signal exists within that distribution. Intuitively, we need an encryption function to preserve and keep the distress confidential; indeed, note that

any fixed-length encryption function would satisfy this, however we do not require the function to return the whole plaintext when $d = \mathsf{false}$. Hence, we use encryption for part of but not all of f_N; we will proceed in constructing a fixed-length reversible function that is DenDistress secure in Section 3.5.

3.4 Security Protocols

In this section, we design the architecture where the user can covertly signal distress to DCP via the webserver. Before the distress can be sent, both the user and webserver have to separately go through an enrolment process.

3.4.1 Server Enrolment

The server enrolment protocol aims to include the webserver in the server database, along with the sharing of secrets between the webserver and DCP which will be necessary for the communication between the webserver and DCP after the webserver receives a distress signal from the user. This process is shown in Figure 3.2.

Initially, the webserver connects to DCP using TLS. The webserver then generates a public and private key pair $(pk_{B'}, sk_{B'})$ for the encryption scheme discussed in Section 3.5.3, and stores it locally. In addition, the webserver chooses a Diffie-Hellman exponent b. The webserver then calculates and sends g^b, along with their identity B, their certificate Cert_B, their encryption public key $pk_{B'}$, and a signature $\textsc{Sign}_B(g^b, B, pk_{B'})$ signed using the private key of the certificate Cert_B. Note that Cert_B is a certificate for the public key of the webserver, not the newly generated encryption keys.

After receiving the message, DCP verifies the certificate Cert_B, and uses the certificate's public key to verify $\textsc{Sign}_B$; by doing this, DCP verifies that the webserver actually holds the private key of Cert_B, proving authenticity. When all the verification steps have been passed, DCP then chooses their Diffie-Hellman exponent c and computes the Diffie-Hellman key $k_{BC} = (g^b)^c$. This key will be used as a shared, secret key for communication between the webserver and DCP later on in the when they are required to share information when the webserver

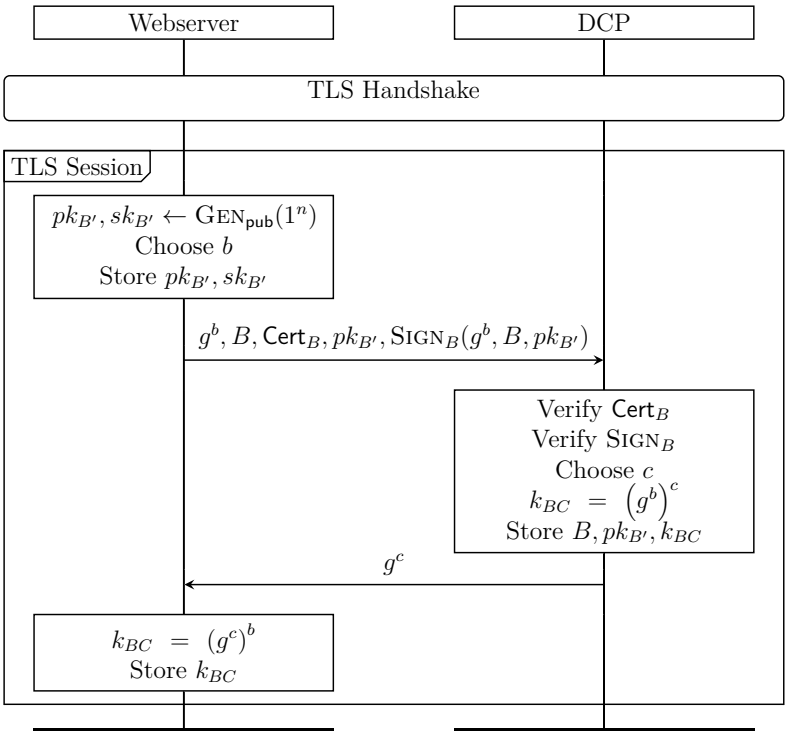

Figure 3.2: Server Enrolment. This describes the enrolment process for a webserver to participate in the scheme. At the end of the server enrolment, the DCP has added the webserver to the server database along with the webserver's public key, and the webserver and DCP agreed on a secret, shared key.

receives a distress signal. DCP then stores the webserver's identity B, the freshly generated public key $pk_{B'}$, and the shared key k_{BC} in the *server database*, before sending g^c to the server so the server can also compute k_{BC}.

3.4.2 User Enrolment

The user enrolment process, occurring independently from the server enrolment, allows the user to be added to the user database and for both the user and DCP to obtain necessary secrets to which the main protocol rely on when the user wants to signal distress. The protocol is shown in Figure 3.3.

The user starts a TLS connection with DCP, then chooses a username usr_A and password pwd_A to register with (or to authenticate themselves if they have previously enrolled), as well as a sequence number sqn_A and a Diffie Hellman exponent a. The user sends usr_A, pwd_A, sqn_A, g^a to DCP. In addition, the user sends their information info_A, which is needed for DCP to react appropriately in case of distress.

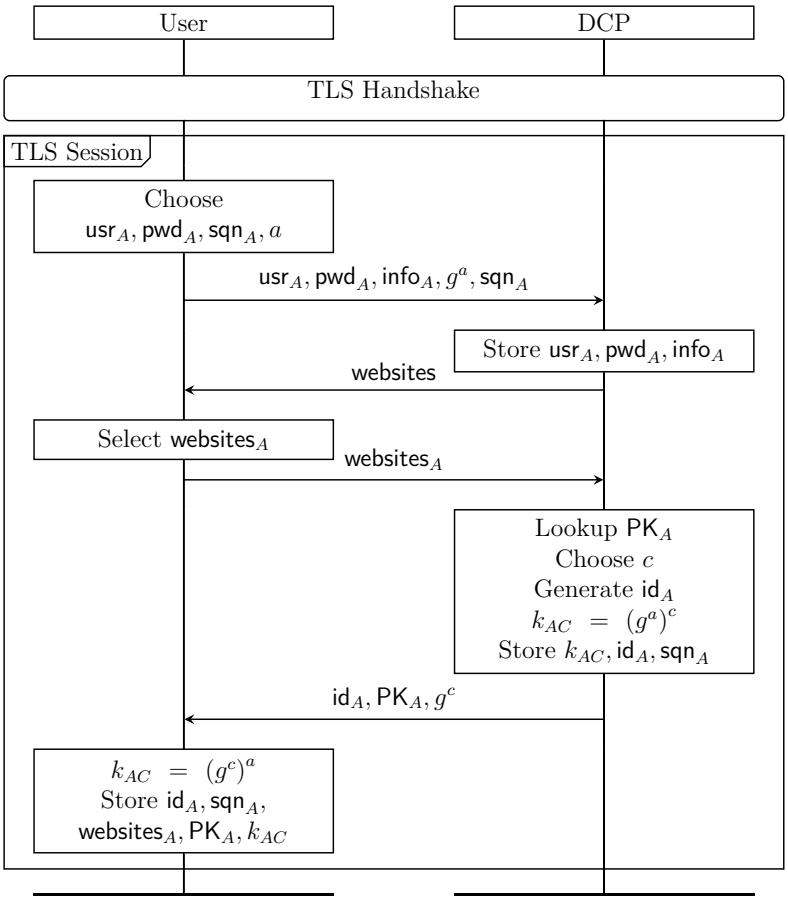

Figure 3.3: User Enrolment. This describes the enrolment process for a user to participate in the scheme. At the end of the user enrolment, the DCP has added the user to the user database, the user has obtained the list of webservers they trust along with the public key of each webserver, along with their identifier and sequence number. The user and DCP agreed on a shared secret key.

For example, this could be contact information for a family member, a domestic abuse shelter, or instructions to call the police. In addition info_A can include a preference on how the user can obtain confirmation, which we describe in Section 3.8.

DCP proceeds to store $\mathsf{usr}_A, \mathsf{pwd}_A, \mathsf{info}_A$ in the *user database* and sends back the list of participating webservers from the server database. From the list, the user chooses a subset $\mathsf{websites}_A$, and returns their choice to the DCP. As previously mentioned the user should choose the webserver in such a way that visiting it will not seem suspicious to $\mathcal{A}_{\text{SUR}}$.

DCP chooses a Diffie-Hellman exponent c and computes the shared key $k_{AC} = (g^a)^c$. In addition, DCP generates id_A, a unique random string of fixed size m_i

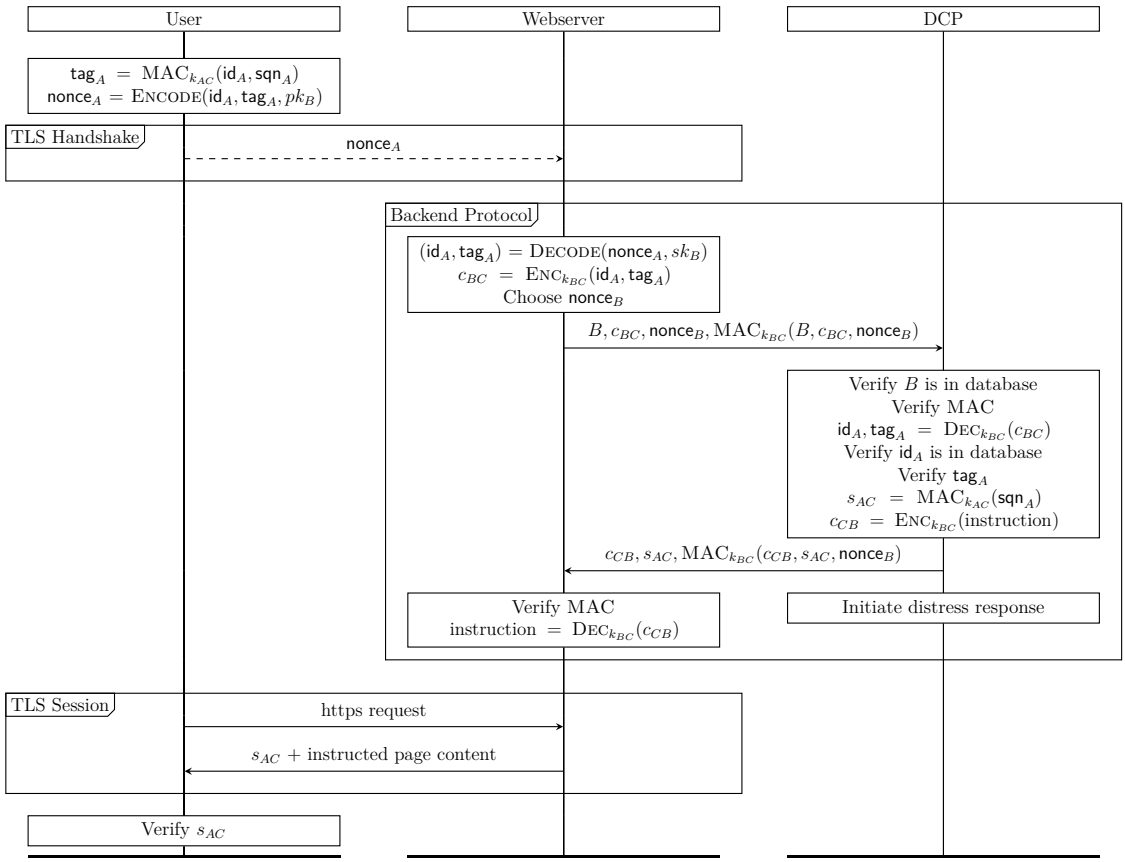

Figure 3.4: Distress Signal Protocol (via TLS). The user, when deciding to send a distress signal, will initiate this protocol and send the encoded distress through the client nonce of a TLS handshake with the webserver. The webserver then relays the user identifier and integrity tag to DCP which will later initiate the agreed distress response. To ensure the user that the distress has been received by DCP, a verification message is relayed by the webserver to the user.

which acts as the user's identifier in the system, and stores $k_{AC}, \mathsf{id}_A, \mathsf{sqn}_A$ in the user database. Finally DCP sends the user identifier id_A, the list of public keys of websites in $\mathsf{websites}_A$, as well as g^c to the user. This allows the user to compute $k_{AC} = (g^c)^a$, and stores the identifier id_A, the shared sequence number sqn_A, the list of webservers $\mathsf{websites}_A$ and their public keys PK_A, as well as k_{AC}, a secret, shared key between the user and DCP.

3.4.3 Signalling Distress

After the server and user enrolments are complete, the user can now indicate distress at will, including in the presence of $\mathcal{A}_{\text{SUR}}$. To do so the user must connect to one

of the webservers they chose during enrolment. In the following the user connects to webserver B with corresponding encryption key $pk_{B'}$.

The user first computes $\mathsf{tag}_A = \mathrm{MAC}_{k_{AC}}(\mathsf{id}_A, \mathsf{sqn}_A)$ using the key shared with DCP. The tag includes the user identifier and sequence number obtained during user enrolment. Note that when a sequence number is used in the main protocol it is incremented by one, regardless of whether the protocol is later aborted.

The user proceeds to create a *distress nonce* using ENCODE, a function we will describe in detail in Section 3.5. In short, ENCODE *hides* the distress, along with id_A and tag_A within an N-bit distress nonce, i.e., it is a distress hiding function. There are two ways that we propose the nonce is sent through: over TLS or HTTP.

If using TLS, the user initiates a TLS handshake with the webserver and sends the distress nonce as the client nonce in Client Hello, and proceeds to completing the handshake. Upon receiving a new TLS connection, the webserver attempts to unravel the client nonce to check if it is a random value (as it normally is in TLS) or if it has a structure that indicates a distress signal. This process is shown in Figure 3.4

If through HTTP, the user sends a HTTP request to the webserver (this is recommended to happen within a TLS session, instead of a plain HTTP connection) and within it, a custom header containing the nonce. The webserver, upon receiving the request, extracts the nonce value from the header and as above, checks if this indicates a distress signal. This process is shown in Figure 3.5

To determine if a distress signal has been sent, this process is through running the DECODE algorithm, described in detail in Section 3.5. If DECODE outputs "not distress" then there is nothing further that the webserver needs to do and the protocol terminates; otherwise, the webserver obtains id_A, along with tag_A as the output of DECODE. Termination of the distress protocol does not affect the processing of the main HTTP/TLS connection, so normal connections function as expected.

If a distress signal is identified, the webserver now encrypts id_A and tag_A with k_{BC}, a shared key that was generated during server enrolment, to obtain c_{BC}. The webserver then sends their identity B, the ciphertext c_B, a nonce nonce_B, as well as $\mathrm{MAC}_{k_{BC}}(B, c_{BC}, \mathsf{nonce}_B)$, signed using the shared key k_{BC}.

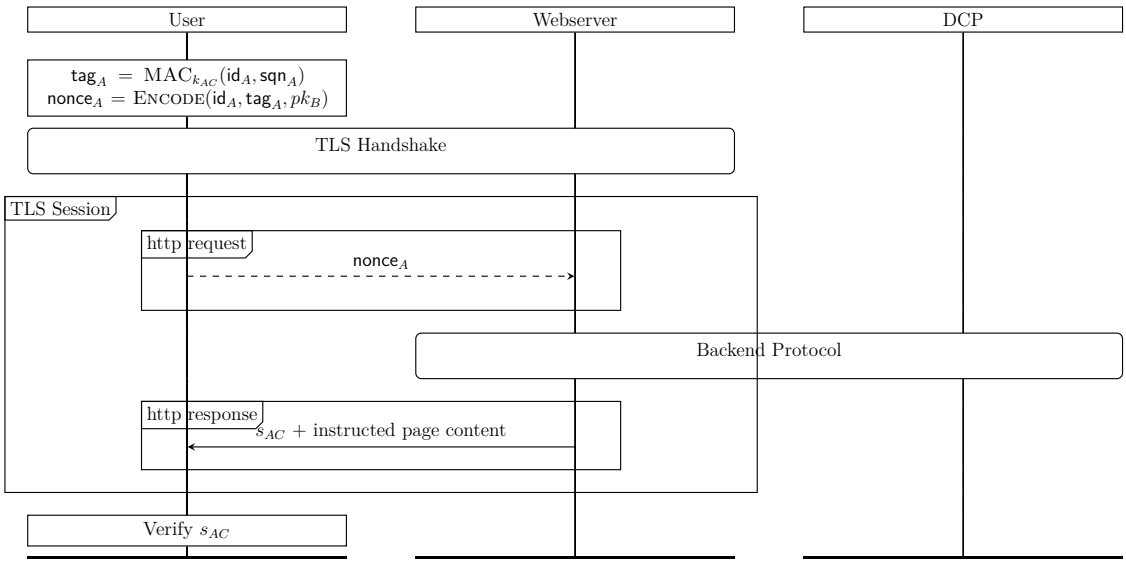

Figure 3.5: Distress Signal Protocol (via HTTP). Similar to Figure 3.4, the user, when deciding to send a distress signal, will initiate this protocol and send the encoded distress. Instead of the nonce embedded in the TLS client nonce, this is now embedded in a custom header as part of a HTTP request; the remaining protocol remains identical.

Upon receipt, DCP checks that B is indeed in the server database and finds the corresponding key k_{BC} so that DCP can verify the MAC. DCP then proceeds to decrypt c_{BC} with k_{BC} to obtain $\mathsf{id}_A, \mathsf{tag}_A$. Now, to verify tag_A, DCP would use the sequence number sqn_A agreed in user enrolment to verify tag_A. To do this, for a specified n_{max} and for $i = 0, \ldots, n_{max}$, DCP computes $\mathsf{VrfyMAC}(\mathsf{tag}, k, (\mathsf{id}, \mathsf{sqn} + i)$ until $\mathsf{VrfyMAC}$ outputs accepts. Otherwise, the protocol aborts. Note that n_{max} is a parameter that the system designer can specify to allow for malicious or accidental errors in previous distress signals not being received by DCP. If the MAC verification is successful, DCP updates sqn to $\mathsf{sqn} + 1$.

Now, DCP needs to inform the user that their distress has been received, which they do by sending a signed sequence number to the user $s_{AC} = \mathrm{MAC}_{k_{AC}}(\mathsf{sqn}_A)$. DCP provides instructions to the webserver on how to send s_{AC} to the user, as specified in the user enrolment as part of info_A; this is done by encrypting instruction with k_{BC} to obtain $c_{CB} = \textsc{Enc}_{k_{BC}}(\text{instruction})$. DCP then sends the signed $c_{CB}, s_{AC}, \mathsf{nonce}_B$ using k_{BC} to the webserver so that the webserver can verify message integrity, authentication, and freshness. After DCP sends the confirmation

message to the webserver, the agreed upon distress response will be initiated, as specified in the user information submitted during enrolment.

Upon receipt, the webserver decrypts c_{CB} to obtain the necessary instructions and verifies MAC using the shared key k_{BC} using nonce_B to check that the message is fresh. After verification is successful, the webserver waits for a https request from the user, and sends s_{AC} within the page content as specified in instruction. The user can then verify DCP's MAC. The specific techniques used to embed the response in the page content can be important if the user is under observation. We discuss ways to do this in practice in Section 3.8.

3.5 Algorithms

In order for the user to hide the distress, we *encode* the distress into what looks like a TLS client nonce while still satisfying the security requirements of TLS. To do so, we construct a fixed-length reversible function with a specified $N = 256$, the size of the TLS client nonce in TLS 1.3. In addition to hiding the distress state, we are also embedding the user's identifier as well as MAC tag. The webserver, upon receiving a nonce in the TLS handshake, will have to *decode* what they have received.

Note that should N vary (e.g. if using HTTP, TLS 1.2, or QUIC), then the bit distributions of the plaintext may be altered accordingly (see Section 3.5.2). In addition, should N be larger, there may be extra information included (other than distress state, identifier, and tag) such as severity of distress, and further action requested from the user at that instance.

3.5.1 Encoding and Decoding Distress

The main idea of the encoding algorithm within TLS or HTTP is to exploit randomness created by an encryption algorithm. Explicitly, the user wishes to send a plaintext containing information about a distress state (as well as an identifier and a MAC for integrity); the plaintext is then encrypted and the ciphertext is presented instead of a nonce.

Algorithm 1 Encoding a distress in an N-bit nonce

1: **function** ENCODE(id, tag, pk)
2: **return** $\text{ENC}_{pk}(1^{m_d}||\mathsf{id}||\mathsf{tag})$
3: **end function**

Fix $m_d, m_i, and m_t$ such that $m_d + m_i + m_t = M$. When using the TLS method, $M = 127$ and $N = 256$, with M being the plaintext size and N being the ciphertext size, and ENC be the mECEG algorithm (see Section 3.5.3).

Our second option involves the use of the HTTP header, with a custom header containing the distress nonce, e.g:

```
X-Nonce:  fb 07 de a5 ef ca f4 cc f0 88 7f a9 c8 5a 29 62 b9 d7 d2
                8a 55 3b 9e 90 ef b7 19 6b 4b bd 15 1b
```

The advantage of using HTTP over TLS is that this can be used (though not recommended unless necessary) within http protocols without TLS; this still allows deniability as the nonce contained may still be non-distress nonces. As there is no size limitation to this, N and M can be chosen freely, while considering overhead.

We describe in Algorithm 1 how the user encodes their distress and identifier into a nonce.

The function ENCODE is used only when the user wishes to signal distress. It requires three inputs: the user's ID, a MAC tag, and the public key of the webserver. The function simply concatenates a string of 1s, the user ID, and the MAC tag. This M-bit integer is then encrypted using pk, the public key of the webserver, to produce a N-bit *distress nonce*.

Algorithm 2 describes the function DECODE, and how the webserver, upon receipt of the nonce, will be able to extract whether or not the user signals a distress; and when they do, also the user identifier and the MAC tag, the latter used for integrity and freshness to be verified by DCP.

Let us show correctness of the algorithm.

Algorithm 2 Decoding a nonce to obtain distress value

```
1: function DECODE(nonce, sk)
2:     r := DEC_sk(nonce)
3:     if MSB_{m_d}(r) == 1^{m_d} then
4:         id := MSB_{m_i}(LSB_{M−m_d}(r))
5:         tag := LSB_{m_t}(r)
6:         return true, id, tag
7:     else
8:         return not_distress
9:     end if
10: end function
```

Proposition 1 (Correctness)**.** *If (pk, sk) is a public/private key pair for public key encryption, then except with negligible probability,*

$$\textsc{Decode}(\textsc{Encode}(\mathsf{id}, \mathsf{tag}, pk), sk) = \mathsf{id}, \mathsf{tag}.$$

Proof. We have $\textsc{Encode}(\mathsf{id}, \mathsf{tag}, pk) = \textsc{Enc}_{pk}(r)$ where $r = 1^{m_d}||\mathsf{id}||\mathsf{tag}$. Now, given that (pk, sk) is a public key encryption pair, then $\textsc{Dec}_{sk}(\textsc{Enc}_{pk}(r)) = r$, except for negligible probability of decryption error. Now, as $\text{MSB}_{m_d}(r) = 1^{m_d}$, Decode returns $\text{MSB}_{m_i}(\text{LSB}_{128-m_d}(r)) = \mathsf{id}$ and $\text{LSB}_{m_t}(r) = \mathsf{tag}$ by construction. That is, indeed, $\textsc{Decode}(\textsc{Encode}(\mathsf{id}, \mathsf{tag}, pk), sk) = \mathsf{id}, \mathsf{tag}$. □

We discuss our design choice. As mentioned in Section 3.4.3, if the user signals distress, Encode will return a distress nonce, which the webserver will decrypt to obtain id and tag. Indeed, the webserver will have access to id. One may be tempted to encrypt id so that the webserver has no access to this, however using Enc which is probabilistic takes more space, whilst using Enc that is deterministic does nothing as a malicious webserver can simply forward the ciphertext. In addition, a malicious webserver also has access to the user's IP address, so having a the user identifier doesn't add much extra information if the goal of $\mathcal{A}_{\text{WEB}}$ is to violate privacy.

The user constructs a MAC tag before inputting it into Encode, which is used to obtain integrity for DCP. This was added to solve two problems that a scheme without tag would have: (1) the webserver can send distress signals from arbitrary users by guessing id; (2) if the user has previously sent a distress

signal, then the webserver is able to replay it and send valid distresses to DCP with the correct id, without the user's knowledge. Using tag which incorporates a sequence number solves these issues.

Another option to obtain integrity and freshness, other than using a MAC (and sequence number), is to simply construct $\mathrm{MAC}_{k_{AC}}(\mathsf{time})$, where time is a timestamp, with DCP checking MAC values of $[\mathsf{time}+t, \mathsf{time}-t]$ for some buffer period t. This introduces time synchronisation as an additional requirement. However, a surveilant adversary, who knows that this scheme exist, can simply change the user's clock on the device settings. Using a secret sequence number solves this, as $\mathcal{A}_{\text{SUR}}$ does not have access to this internal sequence number.

We now proceed in constructing a distress hiding function which utilises Encode and Decode.

Proposition 2. *Let (pk, sk) be a key pair for a public key encryption* Enc*. The following function*

$$f_N = (d, (\mathsf{id}, \mathsf{tag}), pk) = \begin{cases} \textsc{Encode}(\mathsf{id}, \mathsf{tag}, pk) & \text{if } d = \mathsf{true} \\ \mathrm{PRG}(N) & \text{if } d = \mathsf{false} \end{cases}$$

where $\mathrm{PRG}(N)$ outputs a pseudorandom string of size N, is a distress hiding function.

Proof. Let $g_N(y, sk) = \textsc{Decode}(y, sk)$.

Firstly, consider the case when $d = \mathsf{true}$. Given how we choose Enc in Encode to have a fixed output size (see Section 3.5.3), then the first criteria is satisfied. Correctness is shown in Proposition 1. Now consider $d = \mathsf{false}$. Then f_N outputs a random string of size N, so the first criteria is satisfied. Now $\textsc{Decode}(\mathrm{PRG}(N))$ will output $d = \mathsf{false}$ for all $r = \textsc{Dec}(\mathrm{PRG}(N))$ except for r such that $\mathrm{MSB}_{m_d}(r) = 1$, and the probability of that happening is $1/2^{m_d}$. Hence, if m_d is large enough, then correctness is satisfied. □

We will use f_N above as our distress hiding function. Note that if the user does *not* signal distress, the user sends a nonce as usual (e.g., through a pseudorandom number generator). The webserver, upon receipt of a TLS handshake or HTTP/S

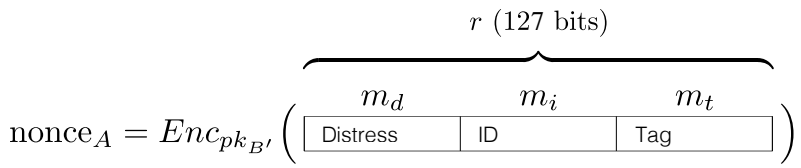

Figure 3.6: Encoding of nonce data (TLS). The input to the public key encryption is a concatenation of a distress signal, an identifier, and an authentication tag with lengths of m_d, m_i, and m_t bits respectively. The three lengths have to add up to 128 bits which will yield a 256 bit output of the encryption.

request from *any* user, would decrypt the client nonce as part of DECODE, regardless of whether or not the user is enrolled in the scheme. Due to our choice of (GEN, ENC, DEC), given the original security properties of a nonce, a false positive will happen with probability $\frac{1}{2^{m_d}}$. If such a false positive occur, the webserver will also obtain some id' and tag' and forwards them to DCP. That is, other than receiving a distress signal directly from a user, the webserver also has the function of filtering out false positives.

Upon receipt of $\mathsf{id}', \mathsf{tag}'$ from a user that did not signal distress, DCP will check this with the user database. The probability that the id is valid is $\frac{N_{user}}{2^{m_i}}$, where N_{user} is the total number of enrolled users. If, already with probability $\frac{1}{2^{m_d}} \cdot \frac{N_{user}}{2^{m_i}}$ such id' is valid, then the probability that a valid $\mathsf{tag}' = \text{MAC}_k(\mathsf{id}', \mathsf{sqn})$ occurs is negligible.

3.5.2 Bit Distribution

The output of ENCODE needs to contain three different pieces of information: a marker that lets the webserver determine whether this is a valid distress signal or a normal nonce, an identifier, and a tag that authenticates the user to the DCP. In this section we discuss the best way to allocate the limited number of available bits to these three fields. We denote the length of the three fields as m_d, m_i, and m_t for number of bits for distress, identifier, and tag, respectively. This is visualised in Figure 3.6. Note that we have a limited number of space for m_d, m_i, and m_t given that $m_d + m_i + m_t = 127$ in order to fit in to the size requirement of client nonce.

The webserver always runs DECODE on every TLS or HTTP/S connections initiated. When DECODE is called on a received nonce which shows distress (i.e. $\text{MSB}_{m_d}(r) = 1^{m_d}$), there are three possibilities: (1) the user with corresponding id

has signalled distress, (2) a false positive from a user enrolled in the system, (3) a false positive from a user not enrolled in the system.

The probability of a false positive happening, from *any* user, is $1/2^{m_d}$. Hence, we want m_d to be as large as possible as this would reduce the probability of false positives. Ideally, $m_d \geq 17$ so that less than 1 in 100,000 TLS connections would come up as false positives. For $m_d = 20$, this goes up exponentially to 1 in 1,000,000.

Now, we want m_i to be large enough so that all users who want to enrol are able to, and that the scheme is more resistant to resource depletion attacks. In addition, with a larger m_i, a malicious webserver $\mathcal{A}_{\text{WEB}}$ is less likely to be able to 'guess' arbitrary ids, though even though they succeed in guessing a valid id, the probability that they will pass the integrity verification through $\text{MAC}_{k_{AC}}(\mathsf{id}, \mathsf{sqn})$ should be negligible—hence, lastly, m_t indicates the size of the tag, and the larger m_t is, the better MAC security obtained.

Though we do not specify the exact size of m_d, m_i, and m_t, we illustrate the case for $m_d = 32$. That is, the probability of the webserver in receiving a false positive is $1/2^{32}$, which, on average, means that the webserver receives one false positive in every 4,294,967,296 TLS connections. This leaves $m_i + m_t = 95$, so if $m_i = 32$ and $m_t = 63$, the user identifier gives space to over 4 billion users, and a MAC tag with output size of 63-bits.

Of course, when there are no constraints on N or M (i.e. $N > 256$ and $M > 127$), then there each m_d, m_t, m_i can be made as large as necessary, while bearing in mind extra overhead computation. In this case, when using HTTP header over TLS client nonce, each of m_d, m_t and m_i can be large enough to reach the desired level of security, as the only size restrictions are based on computation.

3.5.3 Choice of Public Key Encryption

In this section, we discuss the choice of public key encryption required to embed the distress signal within the client nonce within TLS Hello. For the HTTP method, the system designer can choose any IND-CPA secure encryption algorithm.

For the case with TLS handshake, we use the modified Elliptic Curve El Gamal (ECEG) algorithm. For more background on elliptic curves and textbook Elliptic Curve El Gamal (ECEG), the reader is referred to Section 2.2. We recall that traditionally, a ciphertext is of the form (P, Q) where $P = (X_P, Y_P)$, $Q = (X_Q, Y_Q)$. Note that for an elliptic curve $Y^2 = X^3 + AX + B$ over $\mathbb{F}_q$, then we use the modern approach of simply sending the x-coordinate of P and Q, as well as two additional bits b_P and b_Q showing which square root of $X^3 + AX + B$ are the y-coordinates of P and Q respectively. This is an inexpensive computation by the receiver, and allows us to save space [113]. We use this approach, which we refer to as *modified* El-Gamal encryption (mECEG) in throughout this chapter.

Unfortunately, the ciphertexts produced from mECEG are not indistinguishable from random. Indeed, because ciphertexts are elliptic curve points, they follow some structure that allows an adversary to come up with a strategy when trying to see if a ciphertext is an output of an mECEG encryption, instead of chosen uniformly from random. There are several attacks that an adversary can do, as discussed in [29], but the most severe one is simply checking whether $X^3 + AX + B$ is a quadratic residue in the field, which the adversary may succeed with probability $\frac{1}{2}$. We show later, that even with this extra information, the adversary will not be able to distinguish a distribution which contains a distress signal against the uniform distribution.

We chose mECEG[1] instead of other encryption schemes as it satisfies the two main conditions: a ciphertext size as a function of the plaintext size, as well as its semantic security, so that TLS security is maintained. Other encryption schemes, for example IND$-CPA encryption schemes [29, 168] would have a better property: their ciphertexts are indistinguishable from random. However, IND$-CPA

[1]In a previous version of our work, we made a mistake. We defined a distress indistinguishability experiment which describes an adversary advantage being it able to distinguish between $f_N(\mathsf{false}, x, pk)$ and a $f_N(b, x, pk)$ where b is chosen randomly from $\{\mathsf{true}, \mathsf{false}\}$. Our choice of public key encryption, the RSA-based public encryption function in [168], was secure against a IND$-CPA adversary. The proof by reduction was satisfying to write. However, it became clear later that there is no way that the ciphertext with any reasonable plaintext size containing information, will fit into 128 bits! In addition, the computation by the webserver would have been very heavy. Hence, we needed to reframe what we considered 'security': instead of distinguishing between two nonces, the adversary now is required to distinguish a distress nonce within a distribution, which suffices for deniability.

schemes are not practical in this sense: the ciphertexts are too long to fit into the 256-bit space (without compromising on security), and the decryption takes 128 exponentiations, which hinders scalability as webservers will unlikely wish to have such overhead. For more details and definitions on IND$-CPA security, we refer the reader to Appendix A.1.

3.6 Computational Overhead

As the participation of webservers is necessary for the scheme to function, it is important that our design does not generate a large amount of overhead in computation and communication for the website. We discuss the overhead required when using the TLS method, as the HTTP method provides more flexibility on the encryption algorithm used.

Should a webserver participate in our scheme, whenever a TLS connection is requested, the webserver is required to, in parallel to completing the TLS handshake, decode the client nonce received during Client Hello. Note that for the webserver, a TLS handshake already contains two expensive operations: the calculation for the early shared secret key by multiplication of elliptic curve points, and signing a hash using the certificate's public key for authentication. Our protocol adds an additional computational overhead of one elliptic curve multiplication during decryption. The additional calculation of two y-coordinates of the ciphertext elliptic curve points are inexpensive [113].

After decryption is completed, the webserver checks whether the plaintext indicates a distress, through a bitwise AND operation, which is computationally cheap. Should the plaintext reveal a distress, the webserver proceeds to encrypt the user ID and tag and compute a MAC, with symmetric keys obtained during enrolment. After DCP verifies that the distress signal is from a valid user, then the webserver verifies a MAC and decrypts the message from DCP again symmetrically. These operations are not computationally expensive. That is, design choices we made have been to minimise this overhead to one extra elliptic curve multiplication operation, and an increase in distress received only affects overhead linearly.

We have previously argued that the number of false positives received by the webserver is low for $m_d = 32$. To illustrate this, consider Google search, which is the most visited website in the world by traffic [143]. Google on average receives over 40,000 search queries per second [92], or over 3.5 billion per day. Although there is no official data on how many servers are in Google data centres (a 2016 estimate is 2.5 million servers [55]), if 100,000 servers are dedicated to Google Search and assuming that the TLS connections are distributed evenly across all servers, this means that for each server, a distress false positive happens once every 34 years. Even in the worst-case-scenario where all the connections go to a single server, on average, a distress false positive will happen every 29.83 hours.

A malicious user may attempt a DDoS attack (to either the webserver or to DCP) by sending 'false' distress signals; that is, construct $\textsc{Enc}(1^{m_d}||x)$ with arbitrary x of the right size, and sending that through to the webserver as a TLS client nonce. However, as we have discussed above, the operations by the webserver after decryption is of little computational cost. In addition, the MAC verification by DCP aborts the protocol for invalid MACs. Should a malicious user enrol in the scheme and send an extremely high number of distress signals which cause more computations for both the webserver and DCP, in this case, usual DDoS prevention mechanisms can solve this.

3.7 Security Analysis

We assume the following:

- CA The certificate authority issuing certificates of each protocol party is honest.
- DS The signature scheme used by each protocol party is secure; that is, the probability of successfully forging a signature without access to the secret key is negligible.
- M The MAC used by each protocol party is secure.
- N The probability that an honest party picks the same nonce twice is negligible.

3.7.1 Server and User Enrolments

Guarantee 1 (Server Authenticity)**.** *The server enrolment protocol will only complete successfully, if the webserver is able to produce a valid signature corresponding to its certificate (real world identity).*

Proof. If an adversary wants to claim that they are the webserver B, they have to successfully send $x, B, \mathsf{Cert}_B, pk_{B'}, \textsc{Sign}_B(x, B, pk_{B'})$. To replay the message, they need to have previously captured $\textsc{Sign}_B(x, B, pk_{B'})$ from another enrollment, but each enrollment is done in its own TLS session so this message is not available to the attacker. To craft the message, the attacker has to produce $\textsc{Sign}_B(x, B, pk_{B'})$ using a private key associated with Cert_B. Given our Certificate Authority is a trusted entity (CA) and the underlying signature scheme is secure (DS) this is not possible. □

Functionality-wise, the above guarantee means that DCP can be convinced that the identity calimed by the webserver indeed belongs to the webserver.

Guarantee 2 (Server Shared Key)**.** *If the server enrolment protocol is completed successfully, then the webserver obtains a symmetric key that is known only to the webserver and DCP.*

Proof. Firstly, DCP is authenticated during the TLS handshake, and the webserver is authenticated by Guarantee 1. Both g^b and g^c are sent within the same TLS session, so confidentiality and integrity follows. □

Guarantee 3 (DCP Authentication)**.** *If the user enrolment protocol is completed successfully, then the user can be certain that they have been speaking to DCP.*

Proof. DCP acts as the TLS server in the TLS connection, and is thus authenticated during the TLS handshake. □

Guarantee 4. *(User Shared Key and Registration) If the user enrolment protocol is completed successfully, then the user is registered and the user obtains a symmetric key and sequence number that is known only to the user and DCP.*

Proof. The user has a TLS session with DCP by Guarantee 3. Both g^b and g^c are sent within the TLS session, so confidentiality and integrity of the key follows. □

Additionally, because the DCP is honest, the user knows that DCP has added the user's unique identifier, sequence number, personal information, authentication information in the user database. In addition, the user obtains their unique identifier, sequence number, and public keys of websites they chose. Lastly, it is guaranteed that any changes to the database entry is indeed requested by the user.

3.7.2 Distress Signal Protocol

We show the security guarantees from our main protocol, when the user signals distress. A crucial security property we want is captured in Guarantee 5, which we will discuss further in Section 3.7.3.

Guarantee 5. *It is guaranteed that if the user sends a distress signal to the webserver, then it is undetectable by* $\mathcal{A}_{\text{SUR}}$.

Guarantee 6. *If the main protocol is completed successfully, then for DCP, it is guaranteed that the message from the webserver is fresh and indeed comes from an enrolled webserver.*

Proof. To break the guarantee, an adversary needs to send $B, c_B, \mathsf{nonce}_B, \text{MAC}_{k_{BC}}(B, c_B, \mathsf{nonce}_B)$ for an enrolled webserver B. The adversary has to send a valid MAC signed using k_{BC}, a shared secret key between the webserver and DCP. This key is known only to the webserver and DCP, by Guarantee 2, and in particular the adversary doesn't have access to it. The adversary has two options in sending $\text{MAC}_{k_{BC}}(B, c_B)$: by forgery or replay. Forgery is impossible by Assumption (A3). If the adversary replays a message, c_B would be used twice, but c_B contains the sequence number so it cannot be reused. □

Guarantee 7. *If the main protocol is completed successfully, it is guaranteed that the reply from DCP is recent, and indeed comes from DCP.*

Proof. For an adversary to claim to the webserver that they are DCP, they need to send $m_1, m_2, \mathrm{MAC}_{k_{BC}}(m_1, m_2, \mathsf{nonce}_B)$ successfully for an arbitrary m_1, m_2. The adversary can do this by creating their own message or replaying a previously captured one. By Guarantee 2, only the webserver and DCP know the key k_{BC}, so the adversary, who has no access to the secret key, can only successfully forge the MAC with negligible probability by Assumption (M). To replay the message from DCP the adversary would have to capture a previous message using the same nonce, however by Assumption (N) this happens only with negligible probability. □

Guarantee 8. *If DCP receives a valid* id_A *and a valid* tag_A*, then it is guaranteed that the user with corresponding* id_A *has recently signalled distress.*

Proof. Firstly, by Guarantees 6 and 7 we know that the webserver and DCP are indeed speaking to one another, so an adversary cannot be of the form $\mathcal{A}_{\text{NET}}$ or $\mathcal{A}_{\text{SUR}}$. Hence, an adversary has to be $\mathcal{A}_{\text{WEB}}$.

For $\mathcal{A}_{\text{WEB}}$ to break this guarantee, they have to send B, c_B, nonce_B, $\mathrm{MAC}_{k_{BC}}(B, c_B, \mathsf{nonce}_B)$ successfully without the user ever signalling distress, or by replaying a previous distress signal sent by the user.

We consider the first case. For any nonce_A that the adversary receives from the user, $\textsc{Decode}(\mathsf{nonce}_A, sk_B)$ will output "not distress" (except with negligible probability) and in particular, the adversary does not have access to id_A. Hence, the adversary will need to guess $\mathsf{id} \leftarrow_R \{0,1\}^{m_i}$ and sends through $B, c_B, \mathsf{nonce}_B, \mathrm{MAC}_{k_{BC}}(B, c_B, \mathsf{nonce}_B)$ with $c_B = \textsc{Enc}_{k_{BC}}(\mathsf{id}, \mathsf{tag}_A)$. Upon verification, $\mathcal{A}_{\text{WEB}}$ passes the identity verification since B is in the server database, and B has correctly signed a MAC with the shared key. However, DCP upon obtaining id from decrypting c_B, will only pass the verification with probability $\frac{N_{user}}{2^{m_i}}$ where N_{user} is the total number of users registered in the scheme. In addition, $\mathcal{A}_{\text{WEB}}$ needs to forge $\mathsf{tag}_A = \mathrm{MAC}_{k_{AC}}(\mathsf{id}, \mathsf{sqn}_A)$, which succeeds with negligible probability according to Assumption (A4).

We now consider the second case, where the user has signalled distress in the past. In this case, $\mathcal{A}_{\text{WEB}}$ will receive $\mathsf{id}_A, \mathsf{tag}_A$ and as above, sends this in encrypted

form to DCP. Now, DCP will receive a valid id_A, but $\mathcal{A}_{\text{WEB}}$ has to pass the MAC verification: that is, $\mathcal{A}_{\text{WEB}}$ needs to send $\mathsf{tag}_A = \text{MAC}_{k_{AC}}(\mathsf{id}_A, \mathsf{sqn}_A)$ successfully. According to Guarantee 4, both k_{AC} and sqn_A are shared only between the user and DCP. $\mathcal{A}_{\text{WEB}}$ can try to forge a MAC without the k_{AC}, which is impossible except with negligible probability (Assumption (A4)), or replay a previously sent tag_A. However, this requires the user to send the same sqn_A more than once, which is not possible by design. □

We proceed to the main guarantee for the user:

Guarantee 9. *If the user signals distress and the protocol completes successfully, then the user is certain that DCP has recently received the distress.*

Proof. For the adversary to convince the user that DCP has received the distress, the adversary needs to send $\text{MAC}_{k_{AC}}(\mathsf{sqn}_A)$ to the user and successfully pass the verification. Firstly, given that the this communication happens through a TLS session, it is secure against $\mathcal{A}_{\text{NET}}$ and $\mathcal{A}_{\text{SUR}}$. Hence, an adversary has to be $\mathcal{A}_{\text{WEB}}$.

To send a correct $s_{AC} = \text{MAC}_{k_{AC}}(\mathsf{sqn}_A)$, $\mathcal{A}_{\text{WEB}}$ can do this in two ways: forge the MAC or replay a previous message. By Guarantee 4 the key k_{AC} is known only to the user and DCP, so any attempt at forging $\text{MAC}_{k_{AC}}(\mathsf{sqn}_A)$ will only succeed with negligible probability according to Assumption (M). To replay a previous message $\text{MAC}_{k_{AC}}(\mathsf{sqn}_A)$ will fail on the user's verification, as sqn will be outdated.

We conclude that the message $\text{MAC}_{k_{AC}}(\mathsf{sqn}_A)$ has been sent by DCP is fresh and without alteration from $\mathcal{A}_{\text{WEB}}$. DCP, being an honest party, sends this message only after receiving a valid connection from the webserver; by Guarantee 8, the user has indeed recently signalled distress. □

Note that there are a few situations in which $\mathcal{A}_{\text{WEB}}$ are able to act maliciously within our system. Firstly $\mathcal{A}_{\text{WEB}}$ in receipt of a distress signal from the user, may not forward that signal to DCP. However $\mathcal{A}_{\text{WEB}}$ does this, the user will not receive the correct feedback, and using Guarantee 9, the user knows that the distress signal is never received by DCP. Secondly, if the user sends a distress signal which is forwarded from $\mathcal{A}_{\text{WEB}}$ to DCP, after DCP's receipt of the distress $\mathcal{A}_{\text{WEB}}$ may not

send the confirmation message to the user, which may lead to the user believing that the distress has never been received by DCP. These are attacks on availability and are not solvable by any protocol, and therefore out of scope. In practice, if the user does not receive a confirmation from a particular website, they can simply try signalling distress to another webserver.

3.7.3 Main Protocol

We show the security guarantees from our main protocol, when the user signals distress. A crucial security property we want is captured in the following theorem.

Proposition 3. *Let* $\Pi = (\textsc{Gen}, \textsc{Enc}, \textsc{Dec})$ *be modified Elliptic Curve El Gamal encryption scheme used by the function* Encode. *If PRG is a secure pseudorandom number generator then the following fixed-length reversible function*

$$f_N\left(d, (\mathsf{id}, \mathsf{tag}), pk\right) = \begin{cases} \textsc{Encode}(\mathsf{id}, \mathsf{tag}, pk) & \text{if } d = \mathsf{true} \\ \mathrm{PRG}(N) & \text{if } d = \mathsf{false} \end{cases}$$

with $g_N(r, sk) = \textsc{Decode}(r, \mathsf{sk})$ *is* DenDistress *secure.*

Proof. The strategy of this proof is simple: statistically close distributions are polynomially indistinguishable [78]. Let D_b be the distribution of nonces formed when bit b is chosen, i.e. if $b = 0$ then D_b is simply the uniform distribution, and when $b = 1$ then there exists precisely one distress signal in D_1. Let U be the uniform distribution (so the range of D_0 and D_1 are U) and V be the distribution of elliptic curve points. The statistical distance between $D_0 = \{X_1, X_2, \ldots\}$ and $D_1 = \{Y_1, Y_2, \ldots\}$ is:

$$\Delta(D_0, D_1) = \sum_u |\Pr(X_i = u) - \Pr(Y_i = u)|$$

Now, for all $u \in U$ except for one distress point d, $\Pr(X_i = u) = \Pr(Y_i = u)$. Also note that the number of possible outputs of El Gamal is $\frac{2^N}{4}$ as there exists two X-values in the ciphertexts, each lying on the curve with probability $\frac{1}{2}$. Hence,

$$\Delta(D_0, D_1) = |\Pr(X_i = d) - \Pr(Y_i = d)| = \frac{1}{2^N} - \frac{1}{n} \cdot \frac{4}{2^N} = \frac{n-4}{n \cdot 2^N},$$

which is negligible. □

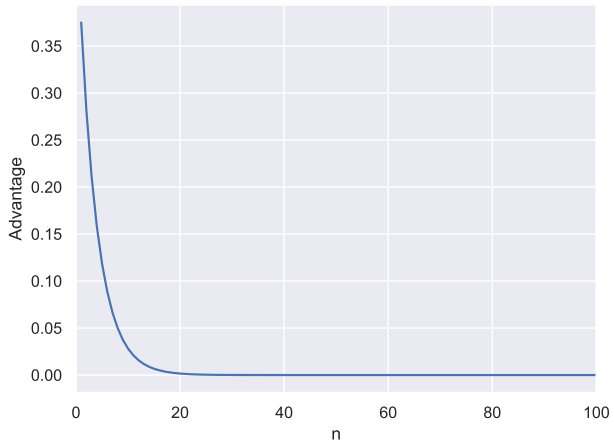

Figure 3.7: $\text{Adv}_{\mathcal{A}}^{\text{DenDistress},f_N}(n)$ quickly goes to 0 as n increases.

We also demonstrate below how an adversary, knowing the structure of the ciphertext, is not able to distinguish a distress ciphertext from a distribution of nonces.

Demonstrating Adversary Advantage

Consider the adversary $\mathcal{A}$ with the following strategy. Given a distribution of nonces $r_1, \dots\ r_n$:

1. for each r_i, check if r_i has the ciphertext structure.
2. If there exists a nonce that follows the ciphertext structure, then flip a coin.
3. If there are no nonces that follow the ciphertext structure, then output $b' = 0$.

Proposition 4. *Let $\mathcal{A}$ be an adversary described above. Then* $\text{Adv}_{\mathcal{A}}^{\text{DenDistress},f_N}(n)$ *is negligible.*

Proof. We consider an adversary $\mathcal{A}$ that uses knowledge of the elliptic curve structure to distinguish between whether or not a distribution of nonces contain a distress call. The adversary's distinguishing advantage is:

$$\text{Adv}_{\mathcal{A}}^{\text{DenDistress},f_N} = |\Pr(b' = 1|b = 1) - \Pr(b' = 1|b = 0)|.$$

Let D_b be the distribution of nonces that the adversary receives; that is, when $b = 1$ there exists a distress signal in the distribution. Let $r_1, \dots, r_n$ be the nonces

that the adversary receives. Write $r_i = X_{i,1}||b_{i,1}||X_{i,2}||b_{i,2}$, where $X_{i,j}$ is 127 bits long and $b_{i,j}$ are single bits. Now, if r_i is an output of mECEG, then $X_{i,j}$ would be on the curve $\mathcal{E}_{A,B} : Y_{i,j}^2 = X_{i,j}^3 + AX_{i,j} + B$ for some $Y_{i,j}$. That is, $X_{i,j}^3 + AX_{i,j} + B$ is a quadratic residue. From Hasse's theorem, this has probability approximately 1/2.

For a nonce r_i, let $ec(r_i) \in \{0, 1\}$ denote whether r_i has the structure of an mECEG ciphertext, that is, $ec(r_i) = 1$ if and only if both $X_{i,1}$ and $X_{i,2}$ lie on the curve $\mathcal{E}_{A,B}$.

Now, when the adversary receives D_1, the distribution may come from either D_1 or D_0, so $\Pr(b' = 1) = \frac{1}{2}$. When the adversary receives D_0, then the adversary looks into each nonce. If $ec(r_i) = 1$, then the adversary flips a coin and if $ec(r_i) = 0$ then the adversary outputs $b = 0$. Hence,

$$\Pr(b' = 1|b = 0) = \frac{1}{2}\Pr(\exists r_i \text{s.t. } ec(r_i) = 1) + 0 \cdot \Pr(\forall r_i \text{s.t. } ec(r_i) = 0).$$

Therefore, $\text{Adv}_{\mathcal{A}}^{\text{DenDistress},f_N} = |\frac{1}{2} - \frac{1}{2}\Pr(\exists r_i \text{s.t. } ec(r_i) = 1)|$. Now, let A_i be the event that $ec(r_i) = 1$. Let $\Pr(ec)(n)$ be the probability that there exists an r_i with $i \in \{1, \ldots, n\}$ such that it has the mECEG ciphertext structure, so by the inclusion-exclusion principle we have

$$\Pr(ec)(n) = \Pr\bigcup_{i=1}^{n} A_i = \sum_{k=1}^{n}(-1)^{k+1}\left(\sum_{1\leq i_1<\ldots<i_k\leq n}\Pr\left(A_{i_1} \cap \ldots \cap A_{i_k}\right)\right)$$

Now $\Pr(A_i)$ is the event where both $X_{i,1}$ and $X_{i,2}$ lie on the curve, that is, $X_{i,j}^3 + AX_{i,j} + B$ is a square, with probability 1/2.

For any i, we have $\Pr(A_i) = \frac{2^{2n-2}}{2^{2n}}$. The denominator comes from the fact that given n nonces we are looking at whether *both* $X_{i,1}$ and $X_{i,2}$ are on $\mathcal{E}_{A,B}$, and the numerator looks at all the possible combinations (as the distribution of outputs PRG is indistinguishable from the uniform distribution) conditioning on both $X_{i,1}$ and $X_{i,2}$ being squares.

Similarly, for $i \neq j \neq k$, we have $\Pr(A_i \cap A_j) = \frac{2^{2n-4}}{2^{2n}}$ and $\Pr(A_i \cap A_j \cap A_k) = \frac{2^{2n-6}}{2^{2n}}$, and so forth. Hence the adversary's advantage is

$$\begin{aligned}\mathrm{Adv}_{\mathcal{A}}^{\mathrm{DenDistress},f_N} &= \left|\frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{n}\binom{n}{i}(-1)^{i+1}\frac{2^{2n-2i)}}{2^{2n}}\right)\right| \\ &= \left|\frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{n}\binom{n}{i}(-1)^{i+1}\frac{1}{2^{2i}}\right)\right|,\end{aligned}$$

which is negligible (and is shown in Figure 3.7). □

3.8 Discussion

In this section we discuss a few topics with additional points we wish to clarify. We further discuss limitations of our work in this chapter in Chapter 7.

3.8.1 TLS Client Hello or HTTP Header: A Comparison

Over the course of this chapter, we presented two options in how the distress nonce can be embedded: through the TLS client nonce or a custom HTTP header. Though we have described them in detail throughout, we summarise the key differences for the two methods on Table 3.1.

3.8.2 Enrolment

To make use of our proposed scheme, a user is required to enrol when the adversary is not present. This is important to allow the main distress signal protocol to be initiated within a single action and for the security guarantees be met. We stress that our scheme is not suitable for situations where enrolment is not possible.

Acts of abuse are often difficult to recognise, and our scheme is only appropriate to be set up in cases where the user has recognised that they are in a vulnerable position and are able to prepare should they decide to seek help. Unfortunately, in many cases domestic abuse survivors unable to leave their abuser, or return to the abuser after an initial offence [136, 137].

We emphasise that we do not intend to minimise the difficulty of helping domestic abuse survivors. Our proposal can fit into a set of solutions that are currently available—from practical advice in managing personal devices and communications

TLS	**HTTP Header**
Works only with https connections	Works with both http and https connections
At most 64-bit security	128-bit security (or more)
Natural randomness in client nonce	Additional header field created, which may trigger suspicion and/or rejected by defensive measures (e.g. firewalls)
Overhead of one elliptic curve multiplication	Overhead dependent on choice of encryption algorithm

Table 3.1: Differences for system designers to consider on choosing between HTTP Header and TLS client nonce to hide the distress content.

[10, 91], to the support given by organisations supporting survivors of domestic abuse, and can be a valuable building block for future work.

This enrolment procedure is similar to those used in practice for other related schemes, for example Path Community [44] were created for individuals to walk home safely, where one might add emergency contact numbers or guardians when they set up the application. When enrolling in this scheme, some personal information may be included so the backend can act appropriately, for example by contacting an emergency contact or forwarding the case to the relevant authority.

In other situations, for example journalists or activists travelling through a hostile nation, preparations can easily be made to face a situation where a distress signal may be required. In this case, the employer or organisation of the individual can set up an architecture where they are the DCP. The response to a distress by the individual may include contacting the relevant embassy including the necessary personal details, including location. Our scheme does not specify the particular user information required for enrolment, nor the response from the backend, as

it is incredibly context-specific and we do not wish to narrow down these many possibilities of implementation.

3.8.3 User Interface

Our scheme specifies what happens in the background when distress is initiated by the user's device, but we have not specified the user's interactions with the device when initiating the distress signal, or in the verification of response from the DCP through the webserver.

Firstly, the enrolment phase includes an installation of the distress system on the user's device. When the adversary is present and the user wishes to initiate the signal, some action is required from the user to do so.

There are plenty of ways in which such action can be done stealthily, and there are plenty of existing techniques which can be implemented. For example, the user can initiate it by pressing a specific keyboard shortcut as they enter the website address (see, for example, [50]), or entering a duress password if the interface is designed in a way that doesn't reveal any information about what's being entered, or pressing a hidden icon on the browser. Note that an action has to be designed to be done *before* the user enters the website as the distress is sent during the TLS handshake, and not the session. There are many considerations a system designer may consider within this UI challenge, including stealth and usability, while reducing false positives through accidental inputs. At the same time, this action needs to be simple and easy to remember for the user who might be in a state of duress when sending the distress signal. This is an interesting challenge and a considerably important one, which we consider as future work on the deployment of this architecture, while noting that this design is entirely context-dependent.

Many organisations have considered a well-designed UI when deploying websites for survivors. Safe Horizon, a New-York based survivor assistance organisation, has introduced SafeChat [85], allowing support to those impacted with domestic violence through one-to-one textual 'chat' between the website user and a trained advocate. SafeChat contains an additional feature which allow the user to quickly leave the

site, and in addition, reminds the users to clear the cache so that anyone who might gain access to your device is unable to view the interactions with SafeHorizon. This allows the user to request information or further action from their home. Indeed 'quick exit' buttons are a common feature in such sites [10].

In our protocol, the user verifies the response s_{AC} to confirm that DCP has indeed received the user's distress signal. The s_{AC} is sent through the TLS connection from the webserver as part of the TLS session—that is, within the page content. The instruction for how best to embed the confirmation in a page can be selected by the user at enrolment and communicated to the webserver by DCP when a distress signal is received. To not limit the possibilities of implementation, we have intentionally not specified how, but we give several options below.

An ideal embedding would be one that gives the user positive confirmation that the distress signal has been received, and at the same time allows for s_{AC} to be extracted by a plugin (or the browser) to be verified cryptographically. Any image generated from a seed (e.g., randomart [22]), can be easily verified, but whether that would appear "normal" on a website, is very context dependent. For websites that commonly include advertisements, it may be possible to display selected advertisements as confirmation, where the visual details (or indeed the text) encode the exact value of s_{AC}. If a website commonly includes dynamic content like video, a response that includes moving images or sound, are options. Particularly with video, there is the opportunity to include other measurable, non-visible parameters such as time delay and use timing-based covert channel techniques. We refer to existing research on undetectable communication [24], for additional examples.

In short, the response sent between the webserver and the user needs to be: (1) invisible to the adversary on the TLS connection, e.g., it is important that s_{AC} is not being sent on its own, but is embedded within a page content so the existence of s_{AC} is obfuscated, (2) verifiable by the user's device, (3) visually verifiable by the user, and (4) unnoticeable to the adversary who has visual access of the device. These are important design choices that need to be made when the architecture is deployed, and decisions are to be made dependent on the context and situation of

the user as well as the webserver's purpose and design. We note that there is no one-size fits all solution that covers all of the possible use-cases of this architecture.

3.9 Conclusion

We introduced a new adversary model $\mathcal{A}_{\text{SUR}}$, who not only has visual access to the user's screen, but full control over all of the user's communication channels as well as the application layer content of the TLS. We introduced distress signalling as a goal for a user to perform against such adversaries. We formalised the adversary's abilities in the form of a security game, and the adversary's goal is to detect a distress signal within a set of normal communications.

To fulfil the user's goal, we constructed an architecture where the user can signal distress using webservers as an intermediary. To do so, the distress is embedded along with additional information as the client nonce in the TLS handshake between the user and the webserver; we specified this in our *encode* function, which uses the Elliptic Curve El Gamal public key encryption scheme. Upon receipt, the webserver *decodes* the distress and forwards the extra information to the backend so that appropriate action can be taken.

Using TLS as well as public key encryption allows for better scalability as well as stealth, and the protocol can coexist with normal uses of TLS. For the webserver, the decoding function only contributes to computational overhead equivalent to one exponentiation per TLS request, and a false positive happens very rarely, so there is very little overhead in terms of communication. When false positives on the webserver's side happen and forwarded to the backend, the backend will perform further integrity checks, so that actual false positives happen only with negligible probability.

We performed a full security analysis of our architecture, including proving distress indistinguishability of our scheme. Lastly, we discussed further practical considerations that a full scale deployment needs to address.

4

Achieving Confidentiality against an Intrusive Adversary

Contents

4.1 Introduction 63
4.1.1 Choice of terminology 67
4.2 Design 68
4.2.1 Design Goal 68
4.2.2 Adversary Model 70
4.2.3 System Model and Design 71
4.3 Nakula 75
4.3.1 User Setup 75
4.3.2 Session Start and Key Generation 76
4.3.3 Data Acquisition and Encryption 77
4.3.4 Session Lock 78
4.3.5 Key and Data Recovery 79
4.4 Security Analysis 80
4.5 Implementation 83
4.6 Conclusion 85

4.1 Introduction

In this chapter, we present our second adversary: one with even more power than $\mathcal{A}_{\text{SUR}}$, however, in contrary, the adversary in this chapter only has such power for a limited amount of time.

The inspiration for this chapter came twofold: first, within the literature a *coercive attacker* [129, 179] has been introduced; this is similar to the use of the term *rubberhose cryptanalysis* to describe the use of force (threats, blackmail, or torture) [141]. This adversary usually demands an authentication secret from the user (password, biometric input), with negative consequences for the user who refuses to do so; hence, to protect themselves the user needs to provide the adversary with the authentication secret, or convince them that they have provided them with a correct one. Indeed, these adversaries and their threats are far from imaginative, with a wide range of motivations, from monetary gain [45] to obtaining confidential data [95].

Secondly, secure border crossing is an issue that has constantly not only reappeared, but escalated [94, 160]. Indeed, some countries have legislations in place that would allow state organisations the power to access electronic devices and demand the owner of the password to provide the password or unlock these devices [66, 96, 162]. It has been reported that when crossing certain Chinese borders, government malware is installed on the traveller's mobile device, which scans the entire contents of the phone [47, 121]. In addition, it has also been reported that US government officials have taken the devices away from the user for hours [89] or copied the contents of digital devices that have been searched at the border, which is then stored in a database for up to 15 years [82].

Depending on the user's risk assessment, they may choose to provide the adversary with the authentication information and lose confidentiality. However, we note that the device that the user carries—both personal or corporate-managed—may contain enormous amounts of sensitive information. The user may be:

1. a lawyer with client information,
2. an individual with private medical information,
3. a charity worker with addresses of domestic abuse shelters,
4. a government employee with classified documents,
5. a whistleblower with evidence of political wrongdoings,
6. a nuclear company employee with a list of nuclear scientists,

7. a journalist with contact details or documents that may reveal information about their sources.

Regardless of the motivation of the adversary, we assume an adversary who wishes to break confidentiality while having the capability, legally or otherwise, in requesting or demanding authentication information from the user. Hence, our goal is to allow the user to maintain confidentiality against such adversary.

At the time of writing, the University of Oxford Information Security team has advised those travelling to high-risk countries to use encrypted devices, however be "prepared to decrypt at border crossings" [142]. Indeed, if a decryption password is demanded the user can either provide it or decline: the latter may be met with an undesired situation, which may depend on the user's citizenship status of that country.

Other approaches to this problem may include *hidden volumes*, a method to hide data on a device so it is not apparent to others who look at it. For example, the device may be split up into different volumes, which unlocks depending on which password the user enters. The idea of the system is as follows: when the user is requested to enter their password, they will input a 'decoy' password instead, to which the file system will direct the user to a file directory with predetermined inconspicuous-looking data [167, 179]. The actual data that the user is trying to hide, remains confidential.

With these approaches in mind, unless the user wishes to decline to provide the password, the user has two of options: provide the password and lose confidentiality, or alternatively hide the data and/or provide false information. The latter may indeed protect the data, however, expecting the user of the system to lie puts them at immense psychological pressure; in addition, should the user be at a border-crossing situation, such practices are in direct conflict with the statement by the Electronic Frontier Foundation [46]:

> "We appreciate and respect technologists' efforts to find ways to help travelers protect their data. However, we recommend against using methods that may be, or even appear to be, calculated to deceive or

> mislead border agents about what data is present on a device. There is a significant risk that border agents could view deliberately hiding data from them as illegal. Lying to border agents can be a serious crime, and the agents may take a very broad view of what constitutes lying."

Among many other sources, an article published in the American Bar Association seconds this [66]:

> "under no circumstances should you lie to a border agent or seek physically to interfere with the agent's performance of official duties."

With the above in mind, in this chapter, we introduce Nakula[1], an approach to protecting device data in the presence of a strong adversary. Specifically, we consider a situation which some data is required to be transported via a physical medium, when there exists a finite period of time in which the adversary has physical access to the device as well as the ability to compel the user carrying the device to reveal and provide any authentication information (e.g., passwords or biometrics). In addition, the adversary would have access to the user's communications channels, which is reflective of the capabilities of a state-level adversary. We will denote this adversary by $\mathcal{A}_{\text{INT}}$, an *intrusive* adversary.

The most important feature of Nakula is as follows: the system does not require the individual to lie or provide any false information even when coerced. We do not rely on hidden volumes or any data hiding mechanisms that are designed to mislead the adversary that some data does not exist. Our approach comes from a simple concept: the user is unable to provide the information to the adversary because the user, legitimately, does not have access to it. The sensitive data within the device, during the period of adversary control, is not available even to the user, because the user does not hold the key required to decrypt the encrypted data. This key is held remotely by a trusted third party, who will not return the necessary building blocks to recover the data unless they are convinced that the user is legitimately requesting so, which is done after the adversary is no longer present on the device.

We summarise the contributions of this chapter as follows:

[1]The name *Nakula* comes from the Mahabharata tales, originally written in Sanskrit. In the mythology, Nakula is a character who is trusted in keeping secrets.

- We create an infrastructure for secure physical data transport, where confidentiality against an adversary with full access of the device and authentication secrets, is guaranteed without the user having to lie or deceive the adversary. Within this infrastructure, we introduce a trusted third party who maintains recovery keys as well as verifying the user authentication after the adversary is no longer present.
- We design protocols between the user and the trusted third party to achieve these goals and provide a proof-of-concept implementation of the scheme.
- We discuss different options in which the user can authenticate themselves to the trusted third party in order to recover the data that was temporarily inaccessible.

A similar work that presents similar goals and solution is discussed in [166], and we discuss this further in Chapter 6.

4.1.1 Choice of terminology

As we will later see, we have named our adversary discussed in the next two chapters as *intrusive adversary*, denoted by $\mathcal{A}_{\text{INT}}$. We have gone against the more standard (and in so, expected) term of "coercive adversary" in the literature, including on our own previous work [90], for the following reasons.

It limits the scope of applicability. The Oxford English Dictionary defines coercion as "constraint, restraint, compulsion; the application of force to control the action of a voluntary agent" [124]. In fact, the only requirement in that we need for our definition of adversary is the following: the adversary is able to obtain the user's authentication information. True, it may be by force ("drug him and hit him with this $5 wrench until he tells us the password" [2]) but it may also be a requirement that the user has little choice in without physical threat (e.g. device seizure by law enforcement), as well as other possibilities: the adversary being able to brute-force or guess the password (e.g., through password reuse [54, 116]

[2] A figure every security researcher has seen: `https://xkcd.com/538/`

or password sharing [98], having access to the password manager, or finding a post-it note of the user's password(s)).

It trivialises physical and emotional damage from being subject to force. Usually defined an attack by a coercive adversary, the term 'rubberhose cryptanalysis' (i.e. "beating that person with a rubber hose" [170]), exposes people to unnecessary thought of cruelty. In addition, if we ask different demographics of what "coercion" means, the answer will likely be different. Though the examples in the literature surrounding coercion may be consistent with physical violence, coercive behaviour is a common pattern within abuse in intimate relationships, and is a criminal offence in England and Wales [3]. We do not want to make the reader uncomfortable by immediately associating the adversary with the use of force in a capacity that the reader first correlates it with.

We understand that terminologies may change over time; in particular, the authors in [153] made the shift from domestic abuse to coercive control, to reflect that people who experience coercive control are not necessarily cohabitating, and are not necessarily physically violent. In this paper, we chose the more appropriate (and neutral) term *intrusive* adversary instead, denoted by $\mathcal{A}_{\text{INT}}$. This reflects well on the fact that the adversary *intrudes*, or forces, itself to the user's device without invitation without having the two problems outlined above.

4.2 Design

In this section, we describe the goal of Nakula and the design choices we made to reach such goals.

4.2.1 Design Goal

The main goal of this chapter is to keep information confidential in the presence of a strong adversary. Clearly, to ensure confidentiality some sort of encryption is needed. Given that $\mathcal{A}_{\text{INT}}$ is able to demand authentication secrets from the user, existing methods such as a password-based symmetric key encryption (e.g. full-disk encryption), for example, is not suitable as the user will simply reveal the correct

password and the plaintext will be available to the adversary. A second reason why full-disk encryption is unsuitable is the duration in which encryption process takes place: encrypting a large amount of data within one click requires a large amount of time. This is unideal, given that our user requires the data to be encrypted quickly before the adversary obtains access to the device.

Alternatives that would be consistent with the recommendations have that have been suggested include backing up these data over the cloud, and then deleting them on the device [171]. However, this takes significant planning: if the user is in a place where there exists no or limited connectivity, this might not be an option. Equally, if the data is of large size and the adversary is expected to have access to the device within a short period of time, the upload process might not be completed in time. In addition, sending a large amount of information may seem suspicious for users that are under surveillance by the adversary. Though this would be suitable for some data transport, we would like to explore alternatives where data size or connectivity are not limitations.

To solve this, we utilise what is normally called on-the-fly encryption, where the data is automatically stored encrypted with a symmetric key k without user action [112]. When the file is loaded, it is automatically decrypted. Our method is simple: when the user wishes to make the data unavailable to the adversary, the symmetric key k is made unavailable by encrypting it with a public key pk, before k is deleted, so that the user will not have access to k and hence the plaintext.

Obviously, if the user has the corresponding secret key sk either on the device or derived from the user's knowledge, then this is the same case as earlier: the adversary will just coerce the user to reveal sk. Our approach is to involve a trusted third party (the *backend*), who holds sk and will only perform computations to recover the data when they are convinced that any request to do so is legitimately done by the user after the adversary is no longer present. Given our reliance to the backend, we want our system to be secure even if the backend is later compromised, which we will discuss in Section 4.2.3.

To summarise, the goals of Nakula are as follows:

- Secure physical data transport where confidentiality during the presence of a strong adversary is guaranteed.
- Quick, simple data lock: the user should be able to encrypt the data within a push of a button (or any equivalent command, including voice and gesture).
- The user does not have to state any incorrect or misleading information to the adversary.
- Long-term secrecy: should the backend be compromised at a later stage, the backend will not have access to the data stored on the user's device.

We note that though our approach is aimed at being consistent with the recommendation of not hiding or providing false information, we understand that we have not discussed further legal issues, the risk of device confiscation, or risk of harm. These are extremely complex matters that are dependent, at the very least, on jurisdictions and we note that these issues are out of scope of this thesis. Travelling with sensitive data, and in particular doing so across borders, is an exercise in risk management: there are factors about the individual (citizenship, immigration status, travel history, occupation) as well as about the data on the device (data sensitivity, internet connectivity, necessity of data access), that the user will need to factor into their decision making. Some guidance for those travelling with a device have been discussed in [46, 66, 171].

4.2.2 Adversary Model

During a finite time period (during *adversary control*), $\mathcal{A}_{\text{INT}}$ has access to the user's authentication secrets and any encryption keys, having full, physical access to the device and and its underlying logic[3]. We also assume that $\mathcal{A}_{\text{INT}}$ is able to clone the device, and store the cloned data for an indefinite amount of time (but not long enough to break the encryption algorithm used). Lastly, we assume $\mathcal{A}_{\text{INT}}$ has full control over the public network that the user operates on, which is reflective of the capabilities of state-level adversaries,

[3]indeed, to abstract away the adversary's methods of obtaining these, we can simply assume that the adversary has oracle access to the user's authentication secrets

We acknowledge that $\mathcal{A}_{\text{INT}}$ can confiscate and destroy the device, but that is not something any protocol can prevent. Our functional guarantees only consider the case when the device remains in the user's possession.

We assume $\mathcal{A}_{\text{INT}}$ is rational and is aware of the existence of Nakula and that it may be used in the system. In addition, we assume that $\mathcal{A}_{\text{INT}}$ does not have access to the backend's secrets. The goal of $\mathcal{A}_{\text{INT}}$ is to break confidentiality.

Lastly, we again note that our use of the word *adversary* is simply from a systems point of view and not necessarily to imply that they have malicious intent.

4.2.3 System Model and Design

In this chapter, we consider the following honest players:

1. the *user*: the party who wishes to maintain confidentiality of some data.
2. the *backend*: the trusted third party backend. We assume that the backend is an honest-but-curious party.

We assume that secure deletion on the user's device is possible. That is, when we say some data is *deleted*, we mean that it is irrecoverable from the device. We refer the reader to some methods in Section 6.4.

We assume that the user can predict when $\mathcal{A}_{\text{INT}}$ is present, or is given warning to when such occurrence will happen, which may be within a short amount of time. The user's goal is to maintain confidentiality when $\mathcal{A}_{\text{INT}}$ is present, and we assume that the user remains truthful when coerced by $\mathcal{A}_{\text{INT}}$.

In this section onwards, we consider the *data* to be a collection of information in plaintext form, that the user wishes to remain confidential against $\mathcal{A}_{\text{INT}}$. We consider five separate actions that the user performs in order for the system to correctly function:

1. **Setup.** As the name suggests, this includes the user setting up the system on the device. This step is necessary for the user and backend to obtain the necessary keys. In practice, this may include the installation of an application on the user's device, and after the setup, any data that the user wishes

to remain confidential is to be collected and stored encrypted through the application.

2. **Data Acquisition**. Here, the user has access to the data and can continue acquiring information that will be later locked.
3. **Data Lock.** The purpose of data lock is simple: the encryption key is deleted, so the data within the device is now inaccessible to anyone—this includes both $\mathcal{A}_{\text{INT}}$ and the user.
4. **Key Recovery.** The user connects to the backend to obtain a (masked) key back, while performing authentication function f_{auth}.
5. **Data Recovery.** The user obtains back data.

The data flow on the user device happens as follows, and is shown in Figure 4.1. Firstly, the user performs user setup to obtain the backend's public key pk, and we assume that the user is able to communicate to the backend during user setup (e.g., before leaving for a business trip). We then describe the period of time between user setup and data lock to be the *data acquisition* period. During data acquisition, the user has access to the data in plaintext form, and is able to make changes to them. However, the data is always stored in encrypted form on the device and decrypted when it is loaded for the user to perform operations—the encryption and decryption of the data happens without user intervention. We denote the symmetric keys that store the data as *session keys*.

Before $\mathcal{A}_{\text{INT}}$ is present, the user can instigate data lock: at this point, the user loses access to the data. The user can perform data lock at any point after data acquisition, and multiple times before adversary access. That is, the user can perform data lock using a session key k_i and start a new data acquisition session with a new session key k_{i+1}. This allows the user to manage the risks of adversary presence better: though our system assumes that the user is given warning when $\mathcal{A}_{\text{INT}}$ will be present, in reality the there might be cases where this is not possible, for example when the phone is lost or taken with force. Equally, the user might simply wish to lock the data at the end of a working day. In that case, the user

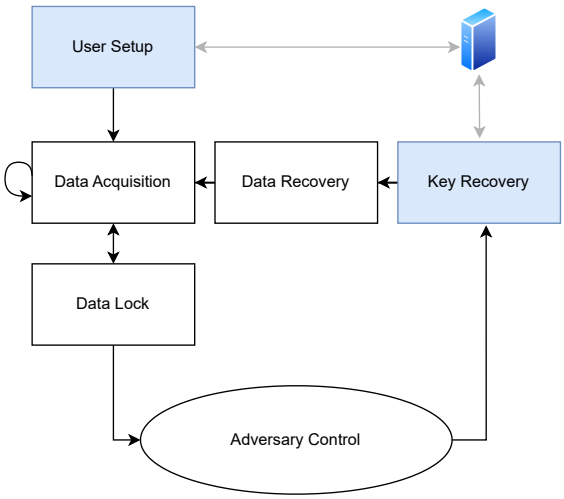

Figure 4.1: Nakula Design: After user setup, the user can collect data during data acquisition period; the user may lock the acquired data at any point, while still being able to collect additional data. Before adversary control, the user completed Data Lock and at this point the user do not have access to the data anymore. When $\mathcal{A}_{\text{INT}}$ is no longer present, the user can complete key recovery followed by data recovery to obtain the data back. The cycle starts again with data acquisition. During user setup and key recovery, the user communicates with the backend.

can minimise the amount of data that is accessible at any point by performing multiple data locks; we describe this in detail in Section 4.3.

As an extra layer of security, during data lock, before the user encrypts with pk to make the data unavailable, the symmetric key k is first 'masked' by encrypting k with a symmetric *masking key* k_R. The masking key is kept within the user's device, and this makes sure that the backend after decrypting will only have access to $\text{ENC}_{k_R}(k)$, and never has direct access to k as an extra layer of protection. Specifically, during data lock, each session key k is masked with k_R, before being encrypted with pk. Then, the user deletes k and $\text{ENC}_{k_R}(k)$. Note that we use public key encryption, instead of symmetric key encryption to ensure that the user at no point in the data life cycle has access to the secret key sk. Note that we can consider pk as a *deletion* key, and sk as a *recovery* key.

After data lock is instigated, the user enters the *adversary control* period, where the user no longer has access to the data as the encryption key is no longer accessible. The adversary control period is indeed the period of time in which we can guarantee confidentiality of the data when $\mathcal{A}_{\text{INT}}$ is present.

Note that because the user does not have access to, and has never had any previous knowledge of sk, the user is legitimately unable to decrypt the the encrypted data as there is no decryption key on the device to decrypt the data. In this way, if the user is demanded by $\mathcal{A}_{\text{INT}}$ the authentication information for the encrypted data, the user simply states that they are not able to decrypt (or *unlock*) the data as they do not have access to the decryption key. This can be supported by a well-designed UI on the device that states this information and how it works, but a large adoption of the system improves the probability that $\mathcal{A}_{\text{INT}}$ is aware of this mechanism, hence convinced that the user is telling the truth and reduces the risk of harm.

It is also important to note that when $\mathcal{A}_{\text{INT}}$ is present, the user does not have access to the session keys nor the data. That is, the user will not be able to see or modify what is inside it, so it is important that the user is aware of this risk before using the mechanism.

When $\mathcal{A}_{\text{INT}}$ is no longer present on the device, the user can instigate *key recovery*, and we assume that communication with the backend is possible during this (e.g., when the user has left a region with limited connectivity). During this process, the user sends the ciphertext of the masked, encrypted session keys to the backend, to which the backend can use the recovery key to decrypt the ciphertext and obtain the masked key. This is then sent back to the user so that they are able to obtain the session keys and obtain access to the data through *data recovery*.

Of course, key recovery needs to be done in a way that preserves authentication of the user, even with $\mathcal{A}_{\text{INT}}$ having access to the user authentication secrets and full, physical access to the device. The backend will not release the encrypted key (by decryption using sk) until they are convinced that any request to do so is not by $\mathcal{A}_{\text{INT}}$, and is legitimately done by the user after $\mathcal{A}_{\text{INT}}$ is no longer present. Hence, the user should not have the ability to authenticate themselves relying solely on their own knowledge. For now, we assume a function f_{auth} that outputs a binary value such that $f_{\mathsf{auth}} = 1$ if and only if the user is legitimately requesting key recovery without the influence of $\mathcal{A}_{\text{INT}}$ We discuss this function in Chapter 5.

During key recovery, a new public key pk' is sent from the backend, with pk deleted on the user's device. This is because $\mathcal{A}_{\text{INT}}$ may store the device data, including the encrypted data and masking key for a long period of time. Should the backend be compromised at a later stage after key recovery, $\mathcal{A}_{\text{INT}}$ henceforth will not have access to the data.

Lastly, we stress that the backend is a trusted party. In practice, this can be the user's employer or trusted organisation and the implementation of Nakula, especially when it comes to designing f_{auth} (as we will discuss in Chapter 5) and a wide-enough adoption allows for $\mathcal{A}_{\text{INT}}$ to be aware of Nakula.

4.3 Nakula

In this section, we describe the inner workings of Nakula. First, we discuss the user setup protocol shown in Figure 4.2, where the user communicates with the backend to agree on the necessary keys. We then describe the life cycle of a session (Figure 4.3) and the various algorithms involved in Nakula (Algorithms 3, 4), ending with key recovery (Figure 4.4), where the user communicates with the backend again to obtain a masked session key and finally recover the original data (Algorithm 5).

4.3.1 User Setup

The goal of the user setup protocol is for the user to obtain the backend's public key pk and agree on a symmetric key k_{AB} used for subsequent communication with the backend. This protocol is an enrollment process that only has to happen once.

To initiate, the user connects to the backend through TLS, and verifies the TLS certificate. The user then chooses a Diffie-Hellman exponent a and sends over g^a to the backend within the authenticated TLS session, along with a preferred method of f_{auth} along with necessary registration information $\mathsf{info}_{f_{\mathsf{auth}}}$. Upon receipt, the backend generates a public-private key pair and chooses a Diffie-Hellman exponent b, and stores the shared key $k_{AB} = (g^a)^b$ and $\mathsf{info}_{f_{\mathsf{auth}}}$. The backend sends the public key pk and g^b to the user and the user computes $k_{AB} = (g^b)^a$. In addition, the user generates a *masking key* k_R and stores it. At this points, three keys are stored: pk

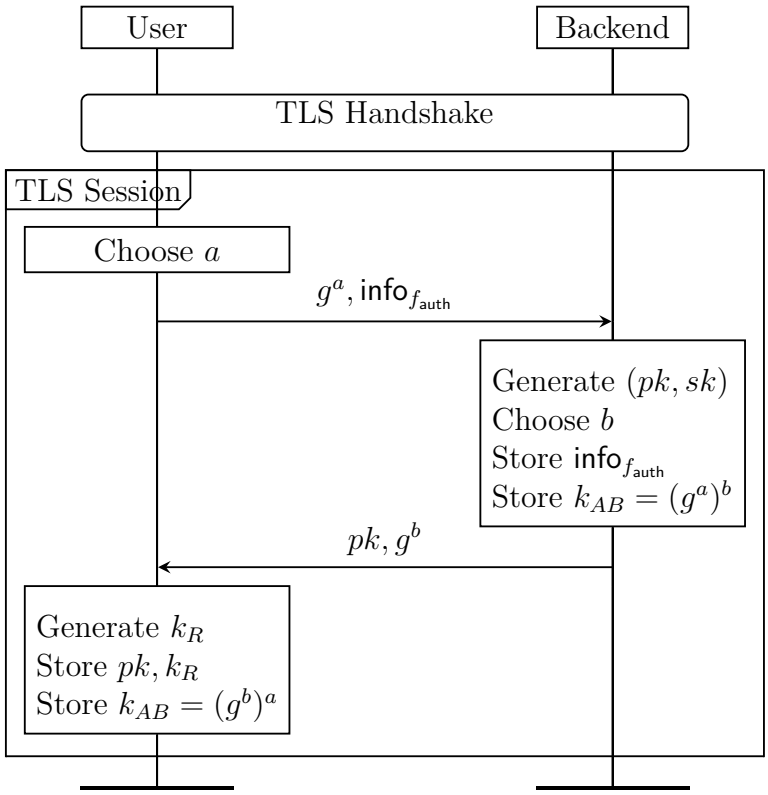

Figure 4.2: User setup protocol. In this protocol the user obtains the backend's public key pk and agree on a symmetric key k_{AB} with the backend. In addition, the user generates the masking key k_R associated to pk.

will act as the user's deletion key for the current epoch, k_R as the masking key, and k_{AB} will be used for subsequent communication with the backend.

Note that (pk, sk) is not global, and is generated uniquely for each enrolled user, and hence also acts as an identifier. The asymmetricity of (pk, sk) is key to the security of our scheme.

4.3.2 Session Start and Key Generation

After completing user setup, the user has obtained all the building blocks required to ensure that confidentiality during adversary control can be guaranteed assuming the user observes the proper life cycle of the data acquisition session. This life cycle is described in Figure 4.3.

The first step is to generate a session key k. This involves a couple of steps and is done with the GENERATE function shown in Algorithm 3. First the user

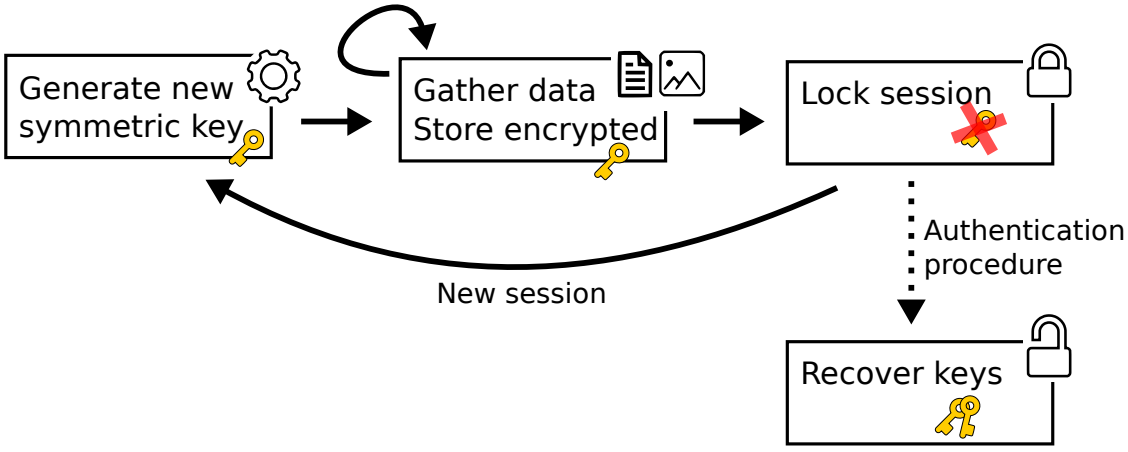

Figure 4.3: The user starts a new session by generating a new symmetric key according to Algorithm 3. The key is used to encrypt data as it is generated and the session is locked when the key is deleted. A new session can be started by creating a new key, or the user can recover locked sessions to regain access to locked data.

Algorithm 3 Session key generation

1: **function** GENERATE(pk, k_R)
2: generate storage key k
3: store $\text{ENC}_{pk}(\text{ENC}_{k_R}(k))$, and $h(k)$
4: **return** k
5: **end function**

generates the actual session key k. The public key pk from the backend along with the masking key k_R are then used to create an encrypted backup of k that the user cannot access without the assistance of the backend, since it is encrypted with pk. This allows the user to delete k at any time in order to protect the access to the data. The user also stores a hash of the key to be used for verification in case the session key ever needs to be recovered from the backend.

4.3.3 Data Acquisition and Encryption

Now the user can start acquiring data (text, photos, etc.). It needs to be ensured that all data is encrypted with k. This is conceptually straightforward although there are some pitfalls if this is done on a mobile device.

Some mobile operating systems will cache images when they are taken, in order to improve the user experience when switching between applications. This is true

Algorithm 4 Session lock for active session (with key k)

```
1: function LOCK()
2:     Secure delete session key k
3: end function
```

to some extent for both Android and iOS and will need to be considered if every photo is sensitive. We discuss this in more detail in Section 4.5.

There is no practical limit on the length of a single session. The user can keep using the same session key k for as long as needed, but if there is ever a situation where the phone might be seized, e.g., going though a roadside checkpoint, or crossing an international border, the user can lock the session to make the data inaccessible.

4.3.4 Session Lock

Given that the data itself is already encrypted, and the user has already stored an encrypted version of k for data recovery, to lock a session all the user has to do is delete k as shown in Algorithm 4.

Once the session key k has been securely deleted, the data is not accessible to anyone, including the user. It is a key feature that the user cannot recover the data by themselves, because if they could, then they would be vulnerable to the request to unlock by $\mathcal{A}_{\text{INT}}$.

This works regardless of whether the adversary is aware of the technical details of Nakula, but a technically literate adversary plays to the user's advantage in practice, since they will understand that there is nothing the user can do, and therefore there is no point in continuing any interrogation.

We should note that if the user knows the data, e.g., they might remember a picture they took, Nakula does not protect against an adversary trying to extract such information from the user themselves. But at least the actual data in physical form, along with the possible intricate details, is beyond reach.

Note that the user may initiate session lock multiple times with the same public key pk (although with different session keys k_i), as shown in Figure 4.3. This allows the user to lock the session if there is any chance at all it might be needed, for

example before going to bed in an unsecured hotel room. We denote by $\vec{C}_{pk}$ the list of ciphertexts under the public key pk, that is, $\vec{C}_{pk} = \{c_{b_1}, c_{b_2}, ...\}$.

To summarise, after session lock is completed i times, the user has stored on the device the following fragments, which we consider public information:

1. $\text{ENC}_{k_1}(\mathsf{data}_1), \ldots, \text{ENC}_{k_i}(\mathsf{data}_i)$
2. $\vec{C}_{pk} = \{\text{ENC}_{pk}(\text{ENC}_{k_R}(k_1)), \ldots, \text{ENC}_{pk}(\text{ENC}_{k_R}(k_i))\}$
3. k_R
4. $\vec{K} = \{h(k_1), h(k_2), \ldots, h(k_i)\}$
5. pk
6. k_{AB}.

4.3.5 Key and Data Recovery

When $\mathcal{A}_{\text{INT}}$ is no longer present, the user can initiate key recovery, to recover access to the session key with the help of the backend. This is done with the Session Key Recovery Protocol shown in Figure 4.4..

The user chooses a nonce n_A and sends it along with the deletion key pk and a list of encrypted backup session keys $\vec{C}_{pk}$ to the backend. It is critical that the user cannot be forced by the adversary to complete this step. We discuss in Chapter 5 how this might be achieved in practice, but for the purpose of the protocol we model it as some authenticating information 'auth'.

The backend, upon receipt of the message from the user, verifies that pk is in the system, then looks up k_{AB} and sk which corresponds to pk. The backend then uses k_{AB} to verify the MAC, before verifying that $f_{\mathsf{auth}}(\mathsf{auth}) = 1$ (see Chapter 5), ensuring that the request from the user is genuine. The backend then decrypts $\vec{C}_{pk}$ with sk and sends this back to the user along with a new deletion key to be used for the next epoch.

The user verifies the $\text{MAC}_{k_{AB}}$ and n_A to ensure integrity and freshness and then computes the set of session keys $\vec{K} = \{k_i\}$ by decrypting each member of the list with k_R. A new masking key k'_R is computed and saved along with the new deletion key pk'.

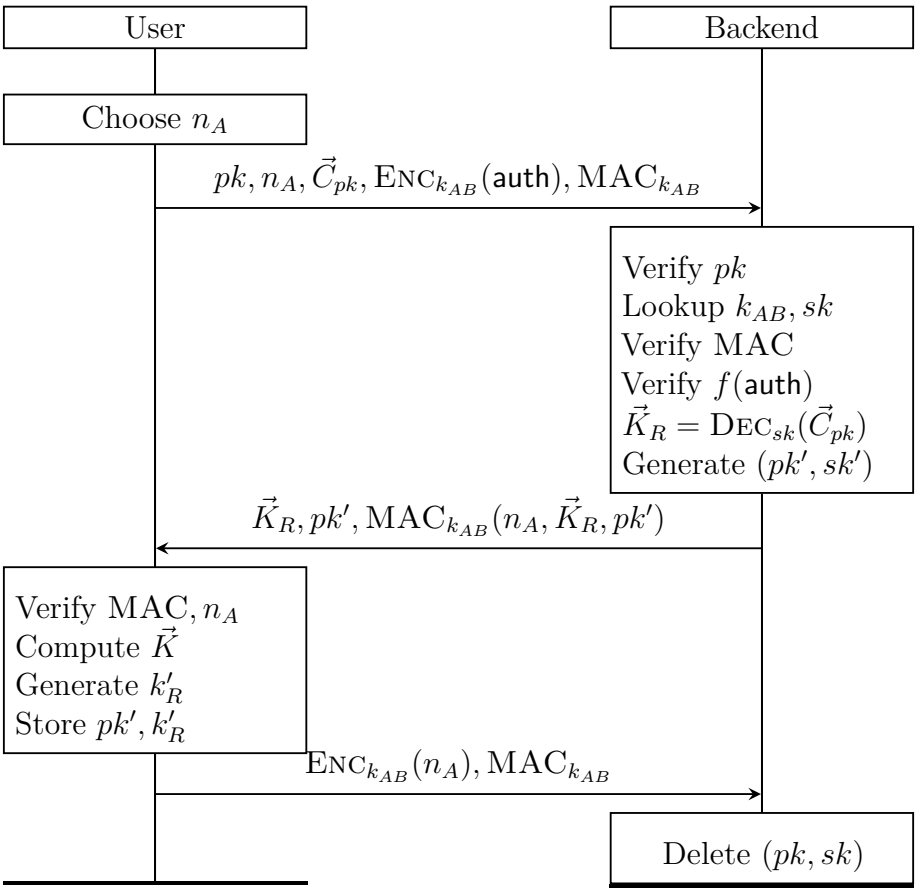

Figure 4.4: Session Key Recovery Protocol. The user supplies the encrypted (and masked) session key along with suitable authentication material (discussed in detail in Chapter 5), and gets back $\vec{K}_R$ from which the recovery key can be calculated, as well as a new public encryption key to use in the next epoch.

Algorithm 5 Data Recovery for key k

1: **function** RECOVER($k_R, pk, \text{ENC}_{k_R}(k)$)
2: $k = \text{DEC}_{k_R}(\text{ENC}_{k_R}(k))$
3: $\mathsf{data} = \text{DEC}_k(\text{ENC}_k(\mathsf{data}))$
4: Delete pk, k, k_R
5: **return** data
6: **end function**

Finally the user sends a confirmation to the backend that then deletes the now defunct (pk, sk), and as a final step to recover the data, the user runs Algorithm 5.

4.4 Security Analysis

We make the following assumptions:

(CA) The certificate authority issuing certificates of each protocol party is honest.

(SD) Once an object is *deleted*, it is irrecoverable from the medium (*secure deletion*).

(SS) Every encryption algorithm used is semantically secure.

(H) The hash function used by each protocol party is preimage-resistant; that is, given $h(m)$, it is infeasible to find m.

(M) The MAC used by each protocol party is secure.

(N) The probability that an honest party picks the same nonce twice is negligible.

(A) $f_{\mathsf{auth}}(\mathsf{auth}) = 1$ if and only if auth is provided at will by the legitimate user.

Guarantee 10. *At the end of user enrolment, and before adversary control, the user and backend shares a key only known to them.*

Proof. Firstly, the backend is authenticated using TLS (Assumption CA). Given that the key exchange happen within the same TLS session, confidentiality and integrity follows. □

Note that the above guarantee only ensures secrecy of k_{AB} strictly *before* adversary control, as $\mathcal{A}_{\text{INT}}$ has the capability of full access to the device, in which they can obtain k_{AB} and is then considered public information.

Guarantee 11. *If key recovery is completed successfully, then the backend is certain that the message coming from the user indeed comes from the user and is fresh.*

Proof. For an adversary to claim that they are the user, they need to pass the backend's MAC verification. That is, they need to either forge or replay $\text{MAC}_{k_{AB}}(\ldots, n_A)$. The latter is not possible due to Assumption (N).

Now assume the adversary does not have access to k_{AB}. Then due to Assumption (M) the adversary also isn't able to forge MAC.

Now assume the adversary has access to k_{AB}. Then the adversary will pass the backend's MAC verification. However, due to Assumption (A), we have $f_{\mathsf{auth}}(\mathsf{auth}) \neq 1$ as this is not requested by the user, and the protocol terminates. □

Guarantee 12. *If key recovery is completed successfully, and that the adversary does not compromise the backend during* adversary control*, then the user is certain that the reply coming from the backend: (1) is a response to the recent request from the user, and (2) indeed comes from the backend.*

Proof. (1) is immediate from Assumption (N). To break (2), an adversary who wishes to imitate the backend will have to pass the MAC verification; they can do this by forgery or replay. The latter is impossible due to Assumption (N). For the former, if the adversary does not have access to k_{AB} then the adversary cannot forge MAC due to assumption (M). So now assume the adversary has access to k_{AB} (as this becomes public information after adversary control). In this case, the adversary does not have access to sk, the private key corresponding to pk. Hence, except for negligible probability, $h(\textsc{Dec}_{k_R}(\textsc{Dec}_{sk}(\vec{C}_{pk})) \neq h(\vec{K})$, so the protocol aborts. □

Guarantee 13. *Assuming the backend doesn't have access to k_R, the backend does not have access to the session keys k used in the user's device.*

Proof. The backend only receives $\textsc{Enc}_{pk}(\textsc{Enc}_{k_R}(k))$, and due to having access to sk they can compute $\textsc{Enc}_{k_R}(k)$. As the backend doesn't have access to k_R, the proof directly follows from (SS). □

Guarantee 14. *During adversary control, no-one, including the user, has access to* data *unless the backend is compromised strictly during adversary control.*

Proof. During adversary control, the data is only stored of the form $\textsc{Enc}_k(\mathsf{data})$ (Assumption SD), so to obtain data they require the symmetric encryption key k (Assumption SS). However, k is only stored in the form of $\textsc{Enc}_{pk}(\textsc{Enc}_{k_R}(k))$ (Assumption SD), and $h(k)$. Given Assumption (H), they cannot obtain k from $h(k)$, so to obtain k they require both the masking key k_R and the secret key sk.

The masking key is present in the user's device, so the user and the adversary have access to them. However, sk is only held by the backend. The backend decrypts $\textsc{Enc}_{pk}(\textsc{Enc}_{k_R}(k))$ if and only if the user freely requests key recovery, but this is outside of adversary control. □

Guarantee 15. *After key recovery is completed successfully, $\mathcal{A}_{\text{INT}}$ will not have access to* data *even if they are able to compromise the backend.*

Proof. Note that during adversary control, the adversary has access to the fragments that we consider public information. If key recovery is successfully completed, the backend has deleted (pk, sk) (Assumption SD) and computes a new (pk', sk'), so even if the adversary is able to compromise the backend, they will only have access to sk' and $\text{DEC}_{sk'}(\text{ENC}_{pk}(\text{ENC}_{k_R}(k))) \neq \text{ENC}_{k_R}(k)$ except with negligible probability. □

We proceed with the main functionality guarantee.

Guarantee 16. *If data recovery is completed successfully, then the no-one, other than the user, is able to gain access to* data*.*

Proof. As described in Algorithm 5, if data recovery is completed successfully, then key recovery has been completed. The user obtains $\vec{K}_R = \text{DEC}_{sk}(\text{ENC}_{pk}(\text{ENC}(k_R)(k_i)) = \text{ENC}(k_R)(k_i)$. Now, the user has access to k_R, so the user simply decrypts each element of the list to obtain the keys k_i.

Now, Guarantee 15 states that an adversary will not have access to data, so now we only need to consider the backend. However, the (honest) backend will only be able to decrypt using sk if sk was not deleted, contradicting Assumption (SD). □

4.5 Implementation

We implemented Nakula in the form of an Android application on the client-side, as shown in Figure 4.5, which was written using Android Studio [68]. We implemented the backend as a local web application using Flask [122]. The interaction between the app and the backend was tested locally using an Google Pixel XL emulator with Android API 33 on Android Studio.

In our implementation, the user conducts data acquisition from within the app. Should the user wish to store a certain message, they can enter it in a text box, along with an optional filename (which will be generated randomly if left

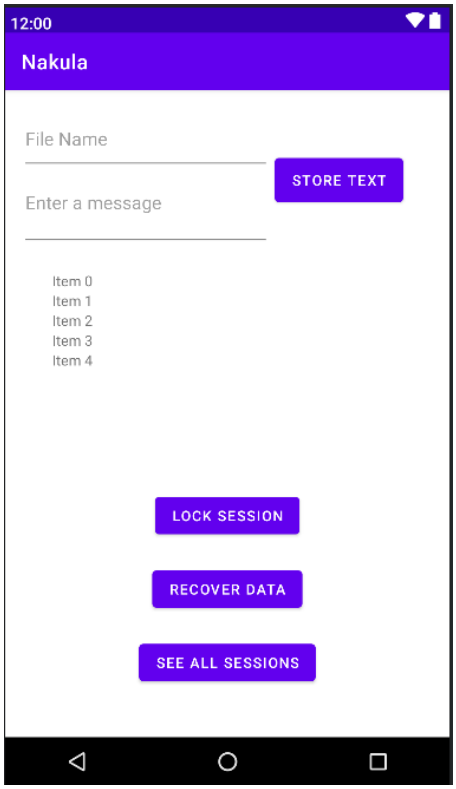

Figure 4.5: A screenshot of our Nakula app, which shows the UI for an unlocked session. In this app, the user can collect text information. When pressing "Store Text", the message is saved encrypted with a session key under the entered file name (e.g., Item 1). The user can still load the text until "Lock Session" is pressed, after which the user then is unable to have access to them. When $\mathcal{A}_{\text{INT}}$ is no longer present, the user can press "Recover Data" to obtain data back, provided that communication is possible and authentication has been completed.

blank) and stored within a specific folder on the device. Note that the file name will not be encrypted so the user should carefully choose the filename that will not reveal the content of the file.

For simplicity, we store these files within the app's internal storage; this makes sure that other applications on the device do not have access to these, however due to the fact that the data will be stored encrypted, in practice, confidentiality will still be maintained. Note that saving the file in internal storage adds the risk of data deletion, as data stored within the app's internal storage is automatically deleted when if the app is uninstalled.

We note that it is likely that cache of files are stored temporarily on the device memory, and hence during data lock it is necessary that these are cleared. We

do not implement these as these are OS-specific.

Key management in Android has developed continuously. The most recent key management, the Android Keystore (for Android API Level ≥ 18) is designed so that keys cannot be extracted from the application process. This is rather challenging for us should we use the Android Keystore to generate and store our session keys, as we need to perform computations on it, and securely delete the key. In our solution, we generate the session keys outside of the keystore, and wrap each of them using an additional encryption key that is managed by Android KeyStore. As we generate the keys externally, we are able to access the key and compute the encryption with the masking key, followed by RSA encryption using the backend's public key. We don't specify the details of this in our protocol as it is implementation-specific.

Note that in our implementation, we are storing short text data just as a proof of concept. However, note that in a full-scale implementation this may include the storage of any other kind of data: images, videos, short and long text including passwords. In addition, the instigation of data lock can be changed to any input method: voice control, gestures, or a specific sequence of actions on the device.

4.6 Conclusion

We proposed Nakula, a mechanism to allow secure data transport against a time-limited adversary $\mathcal{A}_{\text{INT}}$, with full control over a device and any authentication secrets, which, among other scenarios, reflects device seizure during a border cross. We discussed why previous attempts to solve this are insufficient or undesirable, and Nakula is designed as an alternative that overcomes these limitations. We highlight that the main goal of Nakula is to allow data confidentiality when the adversary is present on a device, with the goal being reached without the user having to lie or deceive $\mathcal{A}_{\text{INT}}$, consistent with many legal recommendations of device seizure situations by government officials. Nakula is appropriate to use in situations where the user has to transport the data physically, and is useful in cases where the user is not able to, or does not wish to, transport the data over the internet (even in encrypted form)—for example, there may be poor internet connectivity

for a large data transport, or when there is, they do not wish to increase suspicion by sending a large amount of data over the internet.

To achieve the goal, we introduce the backend: a trusted third party who holds a secret key not accessible by the user; when the user instigates the system lock, the symmetric key that is used to encrypt the data is deleted, and only the backend is able to provide the building blocks necessary for the user to obtain the symmetric key back, which they will do if some authentication mechanism is completed by the user when $\mathcal{A}_{\text{INT}}$ is not present.

We discussed the design decisions for our scheme, proposed the protocols and algorithms within the architecture, and performed a security analysis on our protocols. We discussed a proof-of-concept implementation and we will discuss in detail the authentication mechanism in the next chapter.

5

Authentication Environment against an Intrusive Adversary

Contents

5.1 Introduction 87
5.2 Design 89
5.2.1 System and Adversary Model 89
5.2.2 Protocol Considerations 89
5.2.3 Architectural Considerations 91
5.2.4 On Time-Based Authentication Mechanism 92
5.3 Authentication Mechanisms 95
5.3.1 Security Token 96
5.3.2 Verification Points 99
5.3.3 Remote Authentication With Trusted Entity 101
5.3.4 Dedicated Servers 106
5.4 Choosing f_{auth} 109
5.5 Conclusion 113

5.1 Introduction

In Chapter 4, we've suggested that the security of the protocol is dependent on the assumption that there exists an authentication function f_{auth} such that $f_{\mathsf{auth}}(\mathsf{auth}) = 1$ if and only if the user with authentication input auth is freely requesting key recovery when $\mathcal{A}_{\text{INT}}$ is not present.

Let us first recap the problem in the context of Nakula. During the Key Recovery protocol (Chapter 4, Figure 4.4), the user sends to the backend an (encrypted) input auth to an authentication function f_{auth}. Once the backend verifies $f_{\mathsf{auth}}(\mathsf{auth}) = 1$, the backend replies to the user's message with the building blocks required for the user to decrypt the message. Note that the adversary has full control over the user's device for a finite amount of time, as well as having access to the user's authentication secrets; indeed, the adversary will try to convince the backend by using the correct inputs to auth that the user is legitimately initiating the communication.

This chapter discusses the authentication function f_{auth}. For system designers, what possible options of f_{auth} are there in deploying Nakula ecosystem? No one solution fits all, and there will be restrictions on deployability depending on the architecture of the deployment: a one-person deployment will have different requirements than an intermedia media company with thousands of journalists and offices around the globe.

Though this is a direct follow up from Chapter 4, we note that this system can be designed as a standalone mechanism for other purposes, where the adversary and system model are the same, without necessarily focusing on the low-level protocol or implementations. For continuity, we remain consistent with the terminology we used in Chapter 4. However, at a higher level of abstraction, our mechanisms are about allowing a third party to make a decision on authorship of a request, against an adversary with access to authentication secrets and finite-time, full control over the device. In its very definition, we are creating an *authentication* environment.

In Section 5.2, we briefly discuss the system and adversary model and introduce some terminology and assumptions, and discuss the properties that f_{auth} require and why some previously thought options may be short-sighted. Section 5.3 will see our main contribution: we propose four high-level authentication mechanisms that can satisfy our functionality and security goals. In Section 5.4 we discuss how the different properties can affect which mechanism a system designer may choose to deploy, in the context of Nakula. We conclude in Section 5.5.

5.2 Design

In this section, we discuss the design requirements for f_{auth}.

5.2.1 System and Adversary Model

The system model in this chapter is similar Section 4.2.3. We consider an honest and trustworthy *user* and *backend*. The user, backend, and the Nakula architecture may be a part of a larger infrastructure (e.g. a company network). We also recall the adversary[1] capabilities: it has control over the public networks that the user operates on, full access to the user's device for a limited amount of time, including the ability to clone the device, and access to the user's authentication secrets during the period of adversary control.

The user's goal in this chapter is to authenticate themselves to the backend outside of adversary control, and the backend's goal is to grant authentication if and only if the user is freely requesting it; that is, the user's device is not within adversary control. The adversary's goal is to impersonate the user at any point of time. Similarly as before, the adversary does not perform an attack on availability.

Indeed, since the adversary has full control over it or its clone, we consider the user's device to be untrustworthy, and we want to remove trust away from the device towards something else that the adversary does not have access to. We discuss considerations in designing the authentication mechanism: first, on the protocol level, and second, on a higher architectural level.

5.2.2 Protocol Considerations

As we have previously discussed, a simple username and password authentication method does not suffice. Our adversary has access to the user authentication secrets, so the adversary can simply send a request towards to backend using this information. One may also be tempted to design f_{auth} based on IP address or

[1] note that our notion of $\mathcal{A}_{\text{INT}}$ is less instinctual to use here, as we are also talking about the period *after* adversary control, when the adversary is no longer local to the user. However, we note that we are still talking about the same adversary—it is *intrusive* in the sense that they still have access to the user's device clone.

location; that is, the backend will only grant access if the user is in a country that is added to an allowlist (alternatively, not within a blocklist) on the system through their IP address should it be predetermined that the adversary is only present within specific jurisdictions that the user is travelling in; we remind the reader that the risk considerations for classifying the level of safety for different jurisdictions, is out of scope for this thesis. However, authentication which relies purely on the device's claim of its location is also not desirable, as it is easy to spoof device location [176], even for a low-capability adversary.

It is useful to characterise what properties that f_{auth} is required to have. During adversary control, the adversary has full control of the device, including the ability to clone it, and can request authentication secrets from the user. In addition, the adversary can either try and impersonate the user (by forging a request) either during or after adversary control. From the backend's perspective, the user's genuine device is indistinguishable from the cloned device held by the adversary, so the security of f_{auth} cannot rely only on the existence of an object on the user's device, and/or only the user's knowledge.

Even though the user's device and the cloned device look the same to the backend for both during and after adversary control, there is one main difference between the two periods: after adversary control, the adversary physically holds the cloned device, and the user physically holds the genuine device. The physical presence of the user with the device is a *necessary* property that we can design with. Of course, we also want to protect against trivial replay[2] and machine-in-the-middle relay attacks in our authentication mechanism, so some sort of timestamp or freshness guarantee is also necessary.

Using the two necessary properties: physical space and time, we note that this is not only about where the user is, but also about what the user has access to and what the adversary does not. Importantly, the user has access to the actual

[2]we note that a replay attack, though it breaks authentication, does not break confidentiality of data in the Nakula ecosystem, as once key recovery has been completed, the backend deletes the corresponding key sk so that they are no longer able to decrypt correctly and provide the adversary with the right storage key (Chapter 4, Guarantee 15).

device, while the adversary has access to a cloned device. Lastly, we add that trust is inherently human. Authentication between two individuals, e.g., a line manager seeing their colleague in person and having a conversation with them while being in a safe location showing no signs of duress, is a method that will outperform any purely digital mechanisms.

5.2.3 Architectural Considerations

So far, we have discussed what the design of f_{auth} should consider in terms of the protocol dependencies. However, in designing the full mechanism, there are more limitations that we should consider. In an imagined world, should the user and backend have full control of multiple secure satellites, then the user's location can be verified securely—however this caters to a small number of infrastructural demographics, and we wish to provide solutions that can cater to something as wide-ranging as possible. These are the questions that we can consider:

1. *cost*: what are the cost limitations of the architecture?
2. *scale of architecture*: is this mechanism deployed by an individual, or a large enterprise? How much physical architecture is owned by the enterprise, and how globally spread are they?
3. *flexibility*: does the user require the data straight away after adversary control? Is the user's movements after adversary control flexible, by preference or requirement?
4. *adversary confinement*: is the adversary confined to a specific (static) geographical area?

We do not classify based on sensitivity of the data, as the goal of Nakula is indeed to maintain confidentiality—we assume that the data is of high sensitivity level, based on the user or enterprise's own risk assessment, to warrant the use of this ecosystem.

Given our protocol and architectural considerations, we first consider mechanisms that rely purely on time, and show why it is impossible to build mechanisms on that.

5.2.4 On Time-Based Authentication Mechanism

In our earlier work [90], we mentioned that a time delay mechanism is possible. Indeed, should the mechanism be independent of the user's location, we allow flexibility of the user's movements after adversary control and `data` can be accessed from any physical location, if they can manage the risk of not having immediacy of data access afterwards. At a high level, a time-based authentication mechanism allows the data to be available back to the user after a certain predetermined amount of time has passed.

First, we discuss why this property is at all applicable for our ecosystem. Is it possible, let alone desirable, to consider mechanisms based on these? Though indeed, under our assumption, $\mathcal{A}_{\text{INT}}$ has a time-limit within adversary control; however we have not specified this time-limit, and determining what this duration is, is not straightforward and very much depends on the situation: the United States Customs and Border Protection policy states that device confiscation should ordinarily not exceed 5 days, though it can be extended with a supervisor's approval [51]. The United Kingdom on the other hand, is the only country in Europe that does not impose time limits on immigration detention at its borders, except for vulnerable individuals [120]. The duration also varies for detainment by other law enforcement agencies [1].

Note that an adversary, should they know the duration of the lock period, may be able to confiscate the device until that period has passed and the decryption completed, legally or otherwise. For now, for the sake of simplicity, let us assume that a reasonable duration t can be quantified and that the adversary will not be able to extend the adversary control period longer than t. We can consider the use of a cryptographic primitive called *timelock puzzles* [30, 73, 131]. In short, this allows a user to send a message to the future by relying on a mathematical puzzle that requires time for a computer to work on the puzzle to obtain the solution. We might be tempted to think: why do we need a trusted backend at all, if we can simply use a timelock puzzle to encrypt our storage key during data lock, and it'll automatically decrypt after a predetermined length of time?

Of course, the simple answer is that the adversary has access to the puzzle stored on the user's device during adversary control. Hence, the adversary is able to make a copy and solve the puzzle on its own; in addition, the security of timelock puzzles relies on hardware assumptions: an adversary may have access to better resources with more computational power than a mobile phone, and be able to solve the puzzle much quicker than the expected time it takes on the user's device.

Does this mean that to use a mechanism purely on time, the use of a third party is necessary? The simple answer is no: the adversary can wait the required amount of time, and then contact the backend using the cloned credentials. In the following subsection, we present this.

An attack on an offline time-based protocol

In this section, we present a simple protocol that uses timestamps and hash functions, presented briefly on our work in [90], and illustrate why this protocol, or any other protocols relying only on time delay, is not desirable[3].

First, we add additional information in User Setup (Figure 4.2) and Data Lock (Algorithm 4) to allow some building blocks to be added.

During User Setup, the backend and the user also agree on a t_{min}, where it denotes the minimum amount of time after data lock for the backend to accept a key recovery request from the user. In addition, the backend passes on another secret MAC key k_t to the user that is unique for the session tied to pk. At the end of the user setup, the user and the backend would have shared pk, k_{AB}, k_t, t_{min}, and the user would have generated the masking key k_R.

During data lock, the user records the current time to the nearest hour t_L, and stores $M = \text{MAC}_{k_t}(t_L, \text{ENC}_{pk}(\text{ENC}_{k_R}(k))$ and replaces k_t with its hash. We ensure that the current time is recorded by the device, and is recorded within a MAC that cannot be recreated by an entity other than the backend, as k_t is destroyed in the user's device immediately after.

[3]This subsection can also be called an example of growth of a researcher, as something I worked on for weeks ended up being, quite clearly, impossible.

Algorithm 6 Session lock for active session (with key k) with current time t_L

1: **function** LOCK(t_L)
2: Store $t_L, \mathrm{MAC}_{k_t}(t_L, \mathrm{ENC}_{pk}(\mathrm{ENC}_{k_R}(k)))$
3: $k_t = h(k_t)$
4: Secure delete session key k
5: **end function**

After adversary control, the user instigates the key recovery by sending (among other components) $\mathsf{auth} = t_L, \mathrm{MAC}_{k_t}(t_L, \mathrm{ENC}_{pk}(\mathrm{ENC}_{k_R}(k))$. Upon receipt, the backend needs to verify $f_{\mathsf{auth}}(\mathsf{auth}) = 1$, and does so using Algorithm 7. In short, there are two checks that the backend is doing: (1) whether or not enough time has passed for the key to be released, and (2) whether or not the message was created genuinely by the user.

The first check of the time duration is trivial. The second check is through the use or a MAC, with the following justifications:

1. the use of key update: once a key k_t has been used, it is immediately updated to $h(k_t)$ using a preimage-resistant hash function. This way, the key used to create the MAC no longer exists and this stops the adversary from forging the MAC.
2. Including the ciphertext $\mathrm{ENC}_{pk}(\mathrm{ENC}_{k_R}(k))$ within the MAC, to link the timelock and the storage key k.
3. Store the (hashed) keys in which the user has requested key recovery beforehand, to prevent replay.

At this point, it seems that the use of MAC and timestamp would provide authentication. However, it is (now) clear that this method does not work because of this mechanism relies only on information that lies only within the device. Our adversary has access to the device, and indeed has the ability to clone the device, and has all of the building blocks required to succesfully request key recovery:

$$pk, \vec{C}_{pk}, \mathrm{ENC}_{k_{AB}}(t_L, \mathrm{MAC}_{k_t}(t_L, \mathrm{ENC}_{pk}(\mathrm{ENC}_{k_R}(k))), \mathrm{MAC}_{k_{AB}}$$

Algorithm 7 f_{auth} verification during Key Recovery with current time t_{cur}

```
1: function f_auth(t_cur, t_min, k_t, M, t_L)
2:     if t_cur ≥ t_min + t_L then
3:         for i = 0 to N do
4:             if MAC_{h^i(k_t)}(t_L) == M and i ∉ S then
5:                 store i ∈ S
6:                 return 1
7:             else
8:                 return 0
9:             end if
10:        end for
11:    else
12:        return 0
13:    end if
14: end function
```

In addition, the adversary knows t_{min}, and can request key recovery after t_{min} has passed from t_L. Though indeed, if the user submits key recovery first, then the adversary will not succeed; however if the adversary is able to submit the request before the user, then not only does the adversary have access to data, but this is also an attack on availability as the user no longer will be able to pass the f_{auth} authentication.

As illustrated above, any mechanism that records a timestamp during (offline) data and uses it as a sole property of authentication does not work, for the simple reason that the adversary will have a copy of any building block, including the timestamp. This result holds even if we add a password-based authentication layer, as the adversary also has access to this information. Hence, we conclude that the physical property of location is necessary to include in designing an authentication mechanism.

5.3 Authentication Mechanisms

We proceed with our main contribution in this chapter, where we present four location-based authentication mechanisms, with time or freshness as another necessary component to protect against trivial replay attacks. Our options vary in flexibility and scalability; in Section 5.4 we detail how system designers can

better make a decision on the best authentication method based on their risk factor. Note that we are presenting four authentication methods, we are not claiming that list is exhaustive.

5.3.1 Security Token

In this subsection, we consider the use of a separate authentication device. The idea of this mechanism is simple: place an authentication device (sometimes called security token or security key) in a trusted location where the adversary does not have access to. The user's access to this device provides a guarantee that the user is not within adversary control. Though this will be specific to the architecture, examples of trusted locations may be:

1. The user's home or office, if it can be guaranteed that the adversary will not have access to this space (through a warrant, for example).
2. Kept with a trusted person (e.g. a lawyer, as proposed in [140], a colleague, or family member), who will only hand back the authentication device when they meet in person, showing that the user is physically not with the adversary.
3. In a deposit box where there are pre-existing mechanisms in place to ensure that the user is not compelled by another person to make a withdrawal.
4. Held within the employer's space, moving the trust and legal responsibility away from the user to the employer.

Though there are many security protocols and authentication devices in the market, we assume that the device runs under a FIDO U2F Protocol [154], which enables relying parties to offer a cryptographic second factor authentication for the user. U2F provides protections and detections against man-in-the-middle adversaries, ingenuine/cloned devices, or malware installed on the user's device. This is a standardised authentication protocol used across different authentication devices and across many industries, and is well-reviewed in terms of its security [93, 123]. Instead of reinventing the wheel, we repurpose mechanisms that are already widely deployed which will reduce the cost of deployment.

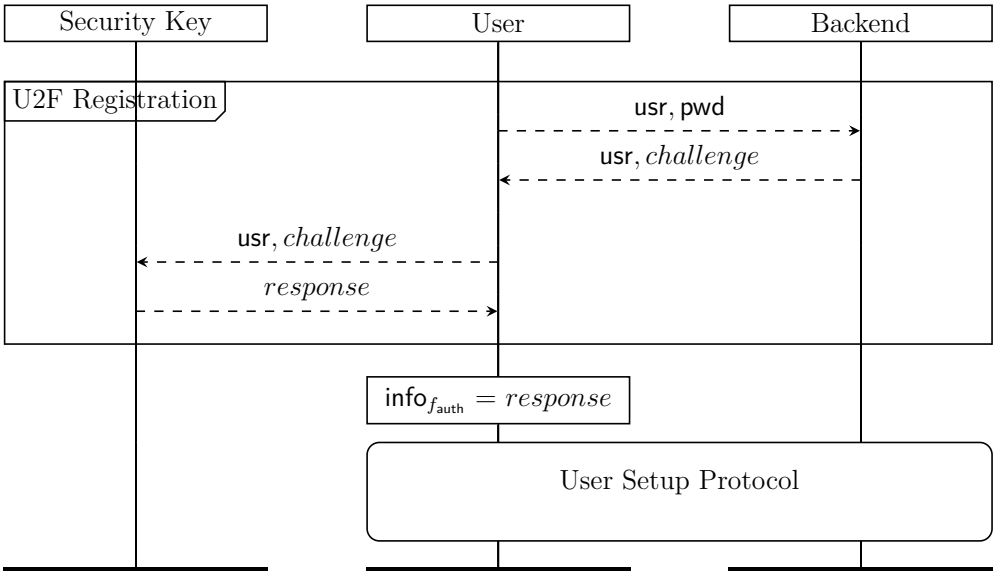

Figure 5.1: Integrating a Security Key (with U2F Protocol) within the User Setup Protocol.

To integrate this mechanism into the Nakula architecture, the user will need to register the authentication device during User Setup: before the User Setup protocol proceeds, the user requests to register an authentication device with the backend, along with a username and password. The backend will forward the username, along with a challenge back the the user, which will be forwarded to the authentication device. After some computations, the authentication device publishes a response, which the user will include within $\mathsf{info}_{f_{\text{auth}}}$ in the User Setup protocol, and the backend will save the information for verification. This is shown in Figure 5.1, and further details about the challenge and reponse components in U2F registration can be found in [154].

After the user completes user setup, the device is left at the safe location, and the user can start gathering data, and proceed to Data Lock as stated in 4.

After adversary control, when the user wishes to obtain back data, they have to first proceed to the key recovery protocol, integrated with U2F. The user simply plugs in the authentication device and request a U2F Authentication protocol to the backend, using the username and password ($\mathsf{usr}, \mathsf{pwd}$) that was registered during user setup. The backend replies with a challenge, which the user forwards to the authentication device, obtaining a response. The user sets the response from the

authentication device as auth, and proceed with the Key Recovery Protocol as described in 4.4. The full protocol can be seen in Figure 5.2.

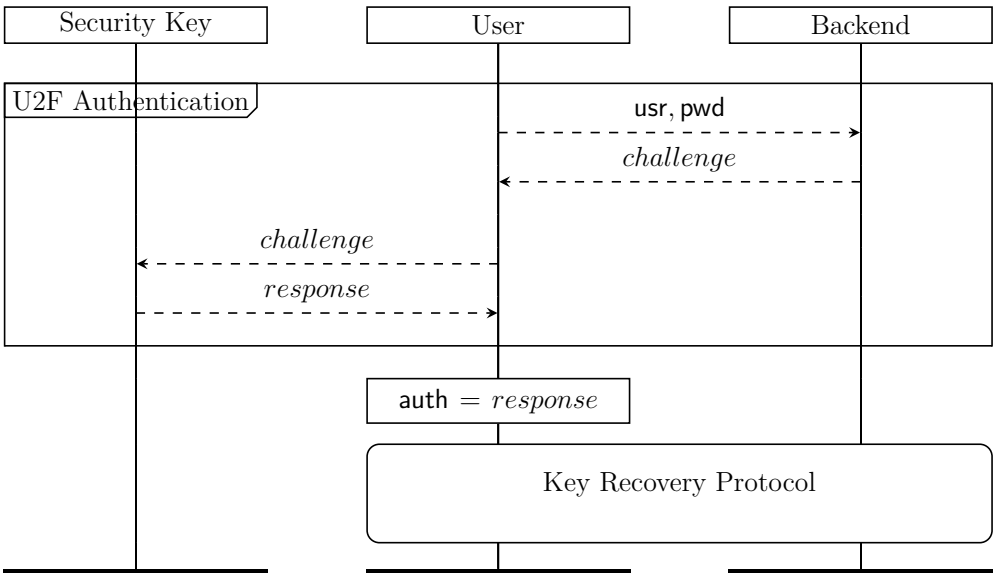

Figure 5.2: Integrating a Security Key (with U2F Protocol) within the Key Recovery Protocol.

Upon receiving auth = *challenge* from the user, the backend verifies that $f_{\mathsf{auth}}(\mathsf{auth}) = 1$ if and only if *response* is correct according to *challenge* using the U2F protocol. Indeed, the security of the authentication mechanism reduces to the security of U2F.

The drawback of this method is that the user have to physically be in a predetermined location and is inflexible to users that are mobile; this means that the user cannot unlock and see the data until after the user is physically in such a position. This is also insufficient against adversaries that also have the extra capability of having access to these spaces (e.g., via collusion or through legal means).

Note that the user setup needs to take place physically away from any location where the adversary may be present, so there is the limitation of location inflexibility not only during key recovery, but also during user setup. Should the authentication device be handed to a trusted person, this may give more flexibility in location both during user setup and key recovery, as the trusted person can be mobile.

5.3.2 Verification Points

Should the infrastructure consist of one or more physical spaces where the adversary does not have access to (e.g,. global or regional company offices with a strict physical access control mechanism) then location flexibility for the user during key recovery can be increased.

For this section, we assume the existence a verification hub, where there exists a trusted *verifier*. The hub can be a part of the same infrastructure as the backend, with an additional assumption that the communications network between the verifier and backend are trusted and secure. This is a standard assumption for enterprise devices connected to each other through an internal network; alternatively for a smaller setup, the connection can be through a direct wired connection with guaranteed physical security. Examples of verifiers may be an NFC or RFID reader. Unfortunately, the use of distance bounding may require additional hardware [60] should it not be built-in on the device; we assume that the device includes the right hardware to make the connection[4].

The high level idea is as follows: if the user can prove that they are near a verifier then they are indeed the user and not the adversary. To prove the property that the user is 'near enough', we utilise distance bounding protocols as part of the Key Recovery protocol. Once the user is ready to complete Key Recovery, the user proceeds with their physical access control and enters the verifier's vicinity (within the acceptable distance bound) and proceeds to send the public key pk to the verifier, and starts the distance bounding protocol. The verifier proceeds with the challenge-response part of the protocol, and when it is verified whether the user is indeed within the distance bound, the verifier forwards pk, n_A and the time of verification to the backend. The user immediately proceeds with the Key Recovery Protocol with $\mathsf{auth} = n_A$, so that the key recovery protocol is tied with the verification session. This is shown in Figure 5.3.

[4]Alternatively, we can consider the user to carry a separate device (e.g. an RFID card) to authenticate themselves to the verifier. Either way, this adds cost.

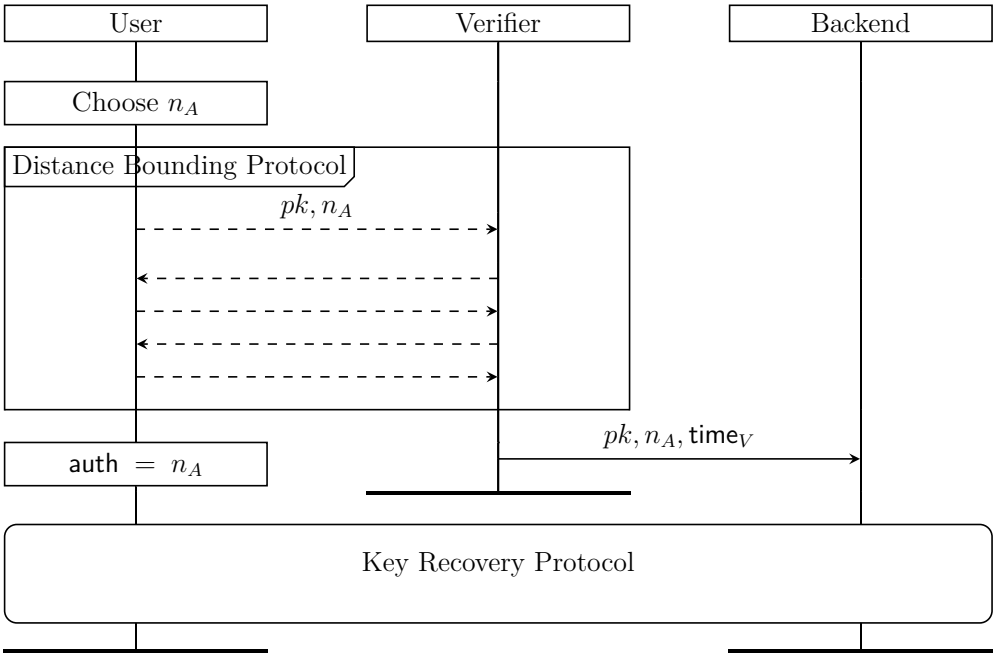

Figure 5.3: Integrating a distance bounding protocol before the Key Recovery Protocol. For simplicity, we assume that the user and verifier has a shared key, obtained during user setup.

Of course, there needs to be registration of the user's device and the verifier before any distance bounding protocol can take place, to exchange security keys. This can be done during User Setup, where the backend acts as a key distributor. We do not specify the specifics, but we simply assume that the user and verifier share a secret key after User Setup. We also assume the acceptable distance bound is chosen to be small enough for the adversary to not be able to physically be within this bound, and is depending on the channel which the distance bounding protocol takes place (e.g. bluetooth, NFC, or RFID).

Upon verification of f_{auth}, the backend verifies that n_A received from the verifier and the user are the same, and that time_V is recent.

Security

Recall that we assume that the adversary doesn't physically have access to the verifier's vicinity. Note that the adversary, who is far away from the verifier, has cloned the user's device, and has all of the building blocks from the user's device, including the shared key between the device and the verifier. The adversary's

goal is to convince the verifier that they are the legitimate user, while not being outside of the accepted bounds.

Within the distance bounding terminology, the adversary can also be considered as a dishonest prover, due to the adversary's access to the device secrets. Indeed, in the verifier's eyes, the user's device and the cloned device are the same—however, the legitimate device is close, and the cloned device is far. However, we assume that the adversary isn't able to modify the user's device, and in particular, cannot install any programme in the user's device; that is, the user's device remains honest. Hence, we require our distance bounding mechanism to be secure against the following attacks:

- distance fraud, where the adversary (as a dishonest prover) convinces the verifier that they are at a different distance than actual, and
- distance hijacking, where the adversary (as a dishonest prover) convinces the verifier that it is a different distance than it actually is, by exploiting the user's device in the verifier's vicinity.

Note that given our assumption that the adversary does not have access to the verifier's vicinity, the adversary is not able to commit impersonation fraud, mafia fraud, or terrorist fraud attacks, as introduced in Section 2.2.4.

Protocols that may be suitable to use include Hancke and Kuhn [81] and those derived from it. With distance fraud being a more standard security guarantee in distance bounding protocols, other proposed protocols that have been tested against distance hijacking attacks can be found in [18, 49].

5.3.3 Remote Authentication With Trusted Entity

In this section, we want to consider a high flexibility mechanism for the user that may be required, or prefer to be, mobile and with potentially changing plans. We have mentioned that human trust should be preferred. We introduced a mechanism in Section 5.3.1 where a trusted individual (the *verifier*) carries an authentication device, and can return the device upon a face-to-face meeting showing the absence of adversary. We assume that the trusted individual has had a history of trust with the

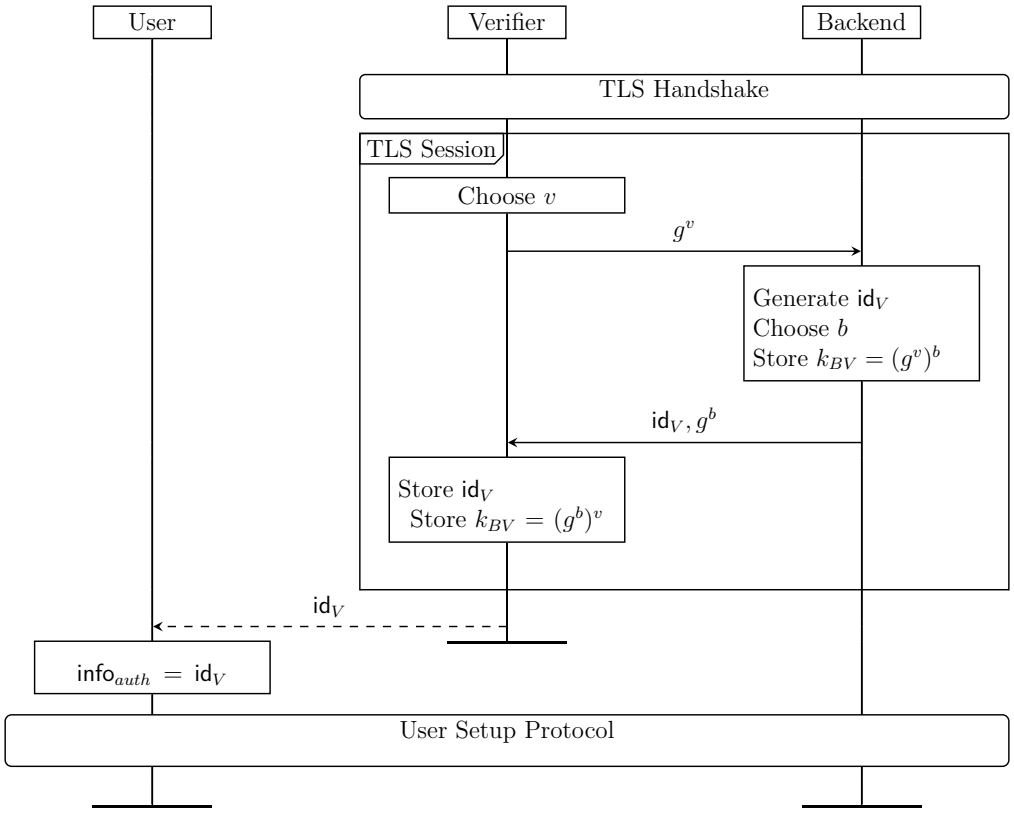

Figure 5.4: User setup with a trusted verifier. In this protocol, the verifier and backend agree on a symmetric key k_{BV} and a verifier identification id_V. The user obtains id_V out of band and proceeds with the User Setup Protocol with $\mathsf{info}_{\mathsf{auth}} = \mathsf{id}_V$.

user: that is, they may have previously met in person and/or exchanged personal or professional credentials. Indeed, the user can choose their trusted individual based on how trustworthy the individual is; or if this is through an enterprise setting with systematic roles, this can be part of the responsibility within the verifier's role.

Though the trusted individual can also be mobile and provide more flexibility (compared to leaving the authentication device in a predetermined location), this still limits immediate data access as such meeting will need to be arranged, and the physical travel to do so may take time.

In order to be registered as the user's verifier, the verifier will have to first register with the backend, so they can agree on a secret key and a verifier identifier id_V. To ensure that the verifer is legitimate, the verifier provides id_V off-band to the user. We assume that this channel is secure, without adversary access; for example, it could be through a face-to-face meeting. The user then proceeds with

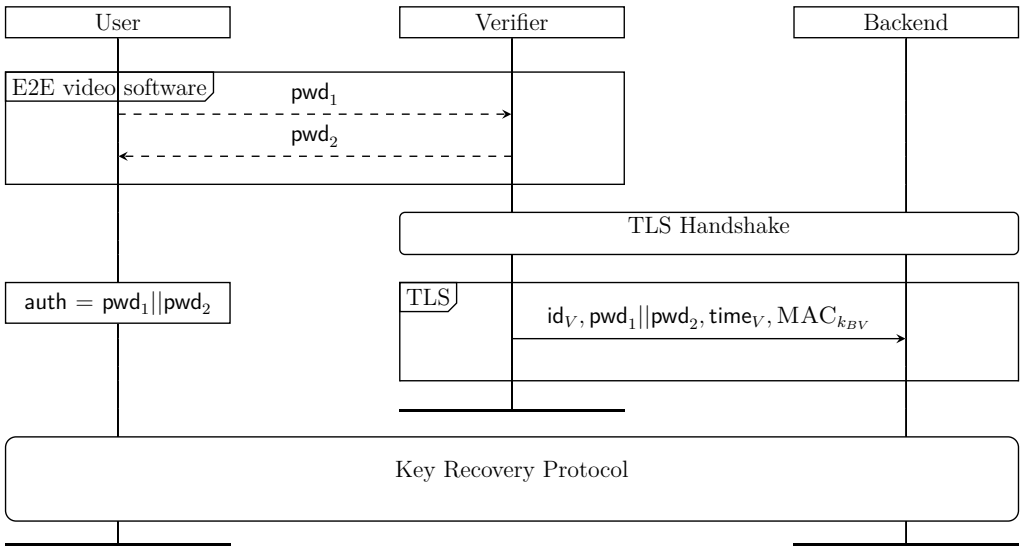

Figure 5.5: Verification using E2EE video software. The user and verifier exchange one-time-passwords that act as an authentication secret during the user's Key Recovery Protocol.

their own User Setup with $\mathsf{info}_{\mathsf{auth}} = \mathsf{id}_V$, to tie the verifier with the user. Note that one user can have multiple verifiers, and one verifier can have multiple users it can verify for. This is shown in Figure 5.4.

After adversary control, when the user is ready to perform key recovery, the user connects with the verifier through an end-to-end-encrypted (E2EE) video software, now commonly available, over the internet. During the video call, the user provides a one-time-password pwd_1 to the verifier, and if the verifier is satisfied that the user is not under adversary control, then they provide another one-time-password pwd_2. As both these agents are trusted, we assume these passwords are indeed single-use (alternatively, we can consider the use of nonces, but given the communication channel is through video, specifically explicitly voiced by each party, a password may be more usable). The verifier then proceeds to send the following information, through a TLS handshake: $\mathsf{id}_V, \mathsf{pwd}_1||\mathsf{pwd}_2, \mathsf{time}_V, \mathrm{MAC}_{k_{BV}}$. The user sends through $\mathsf{auth} = \mathsf{pwd}_1||\mathsf{pwd}_2$ during Key Recovery protocol, and the this is shown in Figure 5.5.

The backend performs the following checks during f_{auth} verification:

1. pk is associated with id_V, which has been completed during User Setup.

2. $\mathsf{pwd}_1||\mathsf{pwd}_2$ provided by the verifier matches with auth provided by the user
3. time_V is recent
4. $\mathrm{MAC}_{k_{BV}}$ is correct, proving that the verifier does have access to the shared key k_{BV} agreed during verifier setup.

We note that the exact protocol can copied without the requirement of an E2EE video software, to allow a face-to-face authentication, as an alternative to using a physical token.

Security

We discuss the security of our online video verification mechanism, and sketch the reasonings behind our design. We assume, due to the nature of the relationship between the user and the verifier, that there exists a secure out-of-band channel for user setup (e.g. through an in-person meeting), and that they have previously exchanged keys for an end-to-end encrypted video communications software; hence, we assume this channel is also secure even post-compromise. We also do not consider the possibility of fake videos, using tools such as deepfake or generative AI [105], where the adversary may be generating a video feed of the user or the verifier which looks realistic, and can respond cohesively within the communication.

First, in the user and verifier setup, the verifier obtains a symmetric shared key with the backend, as well as an identifier id_V. The identifier is then 'linked' with the user through a secure out-of-band channel with the user. Now, should an adversary pretend to be a legitimate user or a legitimate verifier, this will be detected during the out-of-band communication. So based on the out-of-band channel before the user setup, at the end of the user setup, we assume a (legitimate) user has access to a legitimate verifier id id_V for a verifier V that they trust.

To start the key recovery protocol, the user and the verifier needs to communicate over an E2EE video communications platform, which we have assumed to be secure and hence creates a secure channel. During this communication, the verifier has to be convinced that the user is not within adversary control—this is not straightforward

to judge for a human, but there are specific properties that the verifier may look at, or request the user to show:

1. whether there is any other person in the user's video background
2. the user's body language: do they seem relaxed, or in duress?
3. if the video background is consistent with the claimed (or known) location of the user, e.g. if they are in a timezone or climate where it is supposed to be sunny and in daylight, then that is expected.

Indeed, we have mentioned that the adversary has access to the user's authentication secrets. Based on our wording, it remains vague whether or not body language and very specific actions are considered an *authentication secret* per se. If the adversary is able to compel the user to be making the actions, there is a huge amount of psychological pressure placed on the user to do so, and it equally puts the verifier at a difficult position to have to pick up inconsistencies in the user's body language. Should we only consier law-abiding adversaries, the (il)legality of an adversary compelling the user into making false statements is not immediate, and may be classifed under fraud [2]. The security of this authentication method reduces to the ability of the verifier to distinguish between the user freely requesting recovery, or doing so under duress; and hence, this is not a method we recommend for high-risk scenario, or for high security requirement.

Should the verifier accept the verification, they orally exchange one-time-passwords pwd_1 and pwd_2, chosen by the user and verifier respectively. This will act as a one-time shared secret which each party passes on to the backend. In addition, the verifier also passes on id_V, time_V and $\mathrm{MAC}_{k_{BV}}$, allowing the backend to look up the verifier's id_V as well as correlating pk, while ensuring integrity and freshness of the request.

We note that this method opens up the possibility of threat of harm by the adversary towards the user, which might not be suitable in some cases of deployment.

5.3.4 Dedicated Servers

Lastly, we may wish to design a mechanism that provides a high level of flexibility for the user, where similarly as above, the user does not have to plan to be in a specific location in order to perform key recovery. For example, in the situation where the user is travelling through a country that is considered high-risk, then the adversary may only be considered within the bounds of their jurisdiction; that is, we want to show that the user is no longer inside the high-risk country where the adversary is, while also having control over the public communications channels. Note that this assumes a static adversary location coverage. While aiming for flexibility, we also want to design an mechanism with high scalability and guaranteed availability, and to do that we can utilise existing communications infrastructure.

Firstly, the canonical way of determining one's location is the use of Global Positioning System (GPS)[5], which uses a network of (at the time of writing) 24 satellites owned by the United States government, and is available free to use in a wide range of applications. Given existence of this tool, we can be tempted into designing a system where the device generates its location using GPS, and forwards it via the internet to the backend. However, GPS positioning scalability come at a cost: it does not work in an adversarial setting; by default, it is unencrypted and unauthenticated. Multiple attacks including GPS spoofing, where an adversary interferes with the GPS signal [161], resulting in the device thinking that it is somewhere other than the actual location. Secondly, we note that the triangulation calculations are done within the device; the device, as a GPS receiver, will estimate its location based on which four satellites it is in range with, as the satellites operate on a deterministic path and time. However, in our system model, the adversary has access to this device during adversary control, and hence the device, who forwards the location to the backend, cannot be trusted.

Now, as we want to move trust away from the device, we have to involve other parties in the infrastructure. For companies that are able to dedicate server space

[5]without use of generality, we use GPS, but any other global navigation satellite systems (GNSS), e.g., Galileo or GLONASS, can be used.

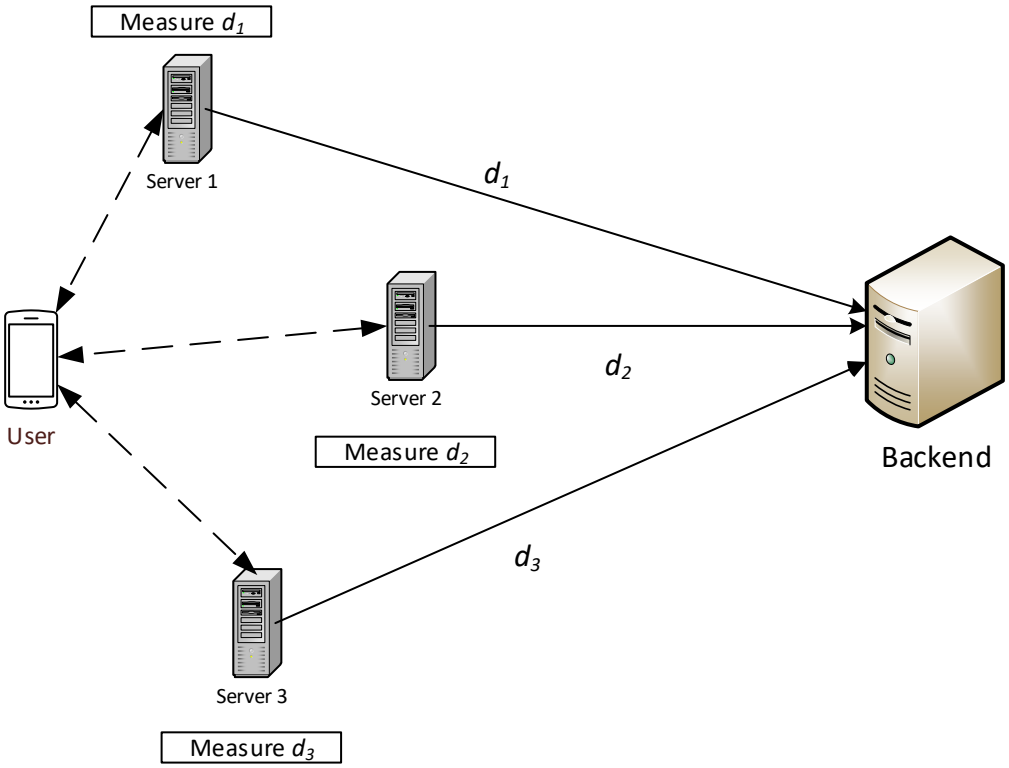

Figure 5.6: A high level functional overview of an initial design of location verification using dedicated servers. Each server estimates the distance between it and the user, and forwards this to the backend. As the backend knows the location of each server and the distance between each server and the user, the backend can perform triangulation calculation to estimate the region of possible location of the user.

across a large geographical area, we can utilise the existence of these servers as our verifiers. The idea is similar to GPS and other location verification systems: if multiple trusted servers can measure the distance between them and the device (say, through computing the time between challenge-response messages), then the user's location can be estimated using triangulation of the aggregate information. We show this idea in Figure 5.6.

We also note that distance bounding mechanisms, though have been introduced to allow location verification [150] do not work well over the internet: first, long-range communications will have larger, less stable timings, and second, in a message exchange, there are many intermediaries and different possible routes. These reasons reduce the accuracy of the distance measurement.

Though there has been plenty work in the literature that focuses on measuring internet distance [52, 70, 115], most don't consider adversarial situations. As

we were at the starting point in designing a new protocol to satisfy our security goals, we came across work by Abdou et al. [4] that does exactly this. The authors propose a mechanism called Client Presence Verification (CPV) where the use of dedicated servers reachable over the internet can provide verification of a client's claim of their presence within a geographical area, using time-delay measurements between the servers and the user. In addition, CPV is designed to be adversarial. In our terminology, the adversary (with the actual device during adversary control, or a cloned device after) may try and claim that they are within the accepted region, by rejecting timestamp messages, unsynchronise their clocks or falsifying their timestamp, or forge calculations in their device. For full details of the protocol, we refer the reader to [4].

Note that an adversary, who has cloned the device, may send the data of the cloned device to an accepted region (without physically being in the accepted region, as confinement is already required) and request CPV verification. Hence, for this method to be secure, we require an additional assumption of Trusted Platform Module (TPM) use: that is, the request mechanism must be tied with the keys stored within the device's TPM.

Hence, an immediate solution for this section is to integrate the CPV mechanism, with the dedicated servers being the CPV verifier. Instead of what we sketched in Figure 5.6, the user claims a location l that they are in, and sends this along with a nonce n_A and pk to the backend. The backend then looks up the three verifiers V_1, V_2, V_3 in response to the the user's claim l, so that the they can check whether l is within the triangle constructed between V_1, V_2, V_3 [4, Algorithm 1]. If each verifier approves the claim, they forward pk, n_A, and current time time_V. The user then proceeds with Key Recovery Protocol with $\mathsf{auth} = n_A$. This is shown in Figure 5.7. The security of this mechanism reduces to the security of the CPV protocol.

Note that the higher the number of verifiers are, the higher the accuracy of the CPV mechanism. In the experiment performed in the paper, experimental results show that the CPV can achieve granularity equivalent to an area of a circle with radius 400km. This area is larger than some countries (e.g. Poland, South Korea)

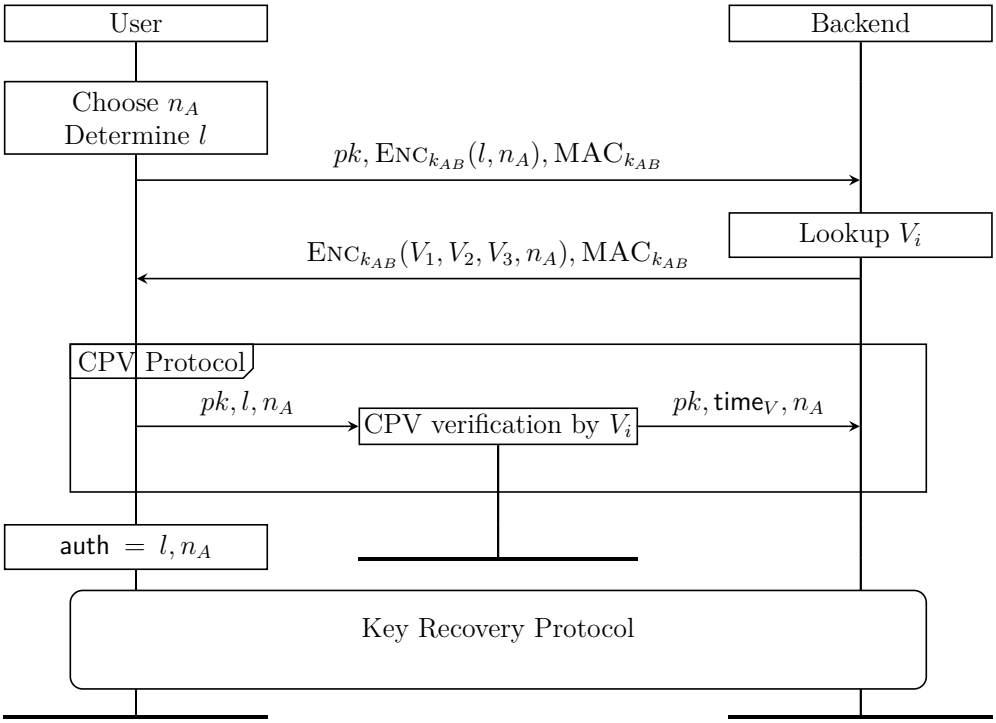

Figure 5.7: Integrating the CPV protocol [4] before the Key Recovery Protocol.

and is unlikely to cater verification of border crossing of neighbouring countries, however be believe that this is of potential when considering adversaries that are significantly further away than the user's current location.

5.4 Choosing f_{auth}

In Section 5.2, we discuss architectural considerations that a system designer may consider when choosing or implementing an authentication mechanism for the Nakula ecosystem: scale of infrastructure, flexibility, and adversary confinement. In this section, we discuss each authentication method with respect to those considerations. We denote by the following: security token (M1), verification point (M2), video authentication (M3), and dedicated servers (M4). We summarise this in Table 5.1, but discuss in detail below, and give examples of multiple deployment cases.

Mechanism	**Scale of Architecture**	**Flexibility**	**Adversary confinement**	**Other Considerations**
Security Token (M1)	Any	Low	Any	Possibility of human-to-human authentication
Verification Point (M2)	Med-High	Med	Any	
Video authentication (M3)	Any	High	Any	Possibility of human-to-human authentication; Potential risk of harm; Requires additional assumption of secure channel post-compromise
Dedicated Servers (M4)	Any except individual deloyment	High	Required	Requires a global network of trusted servers; requires additional assumption of TPM use.

Table 5.1: Comparison of the authentication mechanisms mentioned in this section. We compare them on three main factors: scale of architecture, flexibility (both time and location), and whether or not it requires a static adversary location region.

Scale of Architecture

First, we start with following question: what is the scale of architecture that we are designing for? Is this an single user ecosystem, where an individual wishes to deploy their own mechanism, or a part of a larger existing architecture? Are we building a dedicated service where any user is able to sign up, or is this a closed system within an already existing architecture with their own access control mechanisms?

For an individual deployment, to allow a low cost for the individual, then the user can consider the use of a security token (M1) or use video authentication (M3); the latter does not cost, however the user needs to be able to find a trusted individual who can be reached online, and who is comfortable at making a decision on whether to approve or reject the request.

For the design of a dedicated Nakula service (many users, public use, single backend), then this is similar to the case of individual deployment with (M1) and (M2), as options however with the extra possibility of using dedicated servers (M4) around the world. In the last case, Nakula here also acts as a location verification network.

If the system designer wishes to integrate this into a large existing establishment, with clear identification and access control policies, all mechanisms including using verification points (M2) are possible. In terms of (M1) or (M3), the organisation can assign an individual (relative to the user) to act as verifiers; this may include their line manager, or a dedicated verifier across the organisation (though this doesn't seem like a very exciting job description!). Should they choose (M1) with a static location, the token can be left at a dedicated space for the user that is access controlled (e.g. their desk), and (M2) can be deployed in the organisation's offices, potentially around the world to increase flexibility. Lastly, should the organisation have global offices, they can use the server space from multiple places around the world for (M4); alternatively, they can purchase server space from a third party supplier, which will add cost. Note that the user of M4 is not appropriate for adversaries with mobility.

Flexibility Requirement

Now, we discuss flexibility requirements. For maximum flexibility, of being able to unlock the data at any time or anywhere, (M4) provides the most flexibility, however this assumes a static adversary control region, and due to the high granularity, may carry high rejection rate if multiple adversaries over a large adversary control regions are considered. For (M3), the high location flexibility still holds, however high time flexibility option only holds with the assumption that the human verifier is online and able to verify at the time that the user wishes to do so.

If there is the possibility of deploying multiple verifying hubs across different places, then (M2) provides medium flexibility as the user can choose between different hubs that hey are able to go to. Method (M1) provides the least flexibility, as there is only one predetermined space that the user has to access in order to authenticate themselves, though arguably this provides more security than other mechanisms due to a smaller attack surface.

Example Uses

Taylor is a journalist for a multinational media company, Waystar. They are planning a visit to a high-risk country Cordinia and meet potential informants. However, due to their occupation and recent travel history to other high-risk countries, Waystar classifies this journey as high-risk, requiring Taylor to use Nakula in preparation for the possibility that a border agent, as they wish to leave the country, will wish to search through Taylor's company phone. Taylor will place sensitive information within the Nakula app, and lock it before they leave for the airport at the end of the trip. Waystar prioritises security over flexibility, and Taylor is required to be located and verified, through their organisation access control mechanism, within one of the company offices outside of Cordinia before they can have access to the data again. In this scenario, (M2) is deployed by Waystar, where verification hubs are located in all of their offices around the world, building upon an architecture that they already own.

On the other hand, Blake is a private citizen who carries in his device some medical data that they may require to show in very specific situations, but do not wish for anyone to be able to read it without their consent. To have better control of their data, Blake has registered as a user in the recently deployed Nakula mobile application, which aims to deploy the service to everyone who wishes to use it. Blake's medical data is always locked by default, however in the rare situation that they consent to their data being shared, they ask their partner as a trusted verifier to unlock the data, through (M3): the default is to confirm this face-to-face, but should they be physically far away, they are open to verification over their favourite end-to-end encrypted video calling.

Lastly, we consider a password management software Mataram. As part of its commitment to password confidentiality, it wants to develop a new feature for users travelling across borders which locks some sensitive passwords, and wishes to implement the Nakula architecture within theirs as an option for their users. They purchase (secure) servers across the world for their mechanism, to allow for (M3) to take place during key recovery. Before data lock, the user can choose a blocklist of regions; if the request seems to be coming from there, then the backend rejects this request. As a backup, the user is able to perform data lock using a physical token as well, which they are instructed to leave at a trusted place.

5.5 Conclusion

In this chapter, we presented four high-level methods of authentication f_{auth} as described in Chapter 4, where the user, after a period of adversary control where the adversary has full access to the device and any authentication secrets, wishes to authenticate themselves and showing that they are no longer within adversary control. We discussed why both time and location together are necessary properties for the user to verify during authentication. Our four mechanisms are based on the user of a security token, verification points, dedicated servers, and through human verification (remote or otherwise). We presented how system designers can consider different factors when deciding on authentication methods we proposed,

based on scale of architecture and flexibility requirements, and provide two example cases. Indeed, the user of dedicated servers provide the most flexibility at a cost of maintaining multiple server spaces, and high false negatives. The use of physical tokens provide the least flexibility, however is straightforward to use and has less attack surface. We will discuss potential future work in Chapter 7.

6
Related Work

In this section, we summarise work primarily, but not exclusively in, the academic literature related to our thesis. We focus on four separate sections: strong(er) adversary models and the security against them, undetectable communications, coercion attacks (including plausible deniability and distress signalling), and secure deletion. Though we do our best in categorising the related work, note that there are connections between different sections, and we do not claim exclusive containment of the works listed within each section.

6.1 Strong Adversary Models

Some work has been done exploring the topics of more 'unconventional' attack capabilities. Zhao [178] addresses these threats – including crowd-sourcing, coercion, substantial computational resources, malicious insiders and proposed some mitigations focusing on hardware security. Levy and Schneier [104] presented *intimate threats*, a class of privacy threats arising within intimate relationships, including families, friendships, romantic partnerships, and caregiving relationships, and presented design recommendations against these risks.

The term *UI-bound adversary* was introduced in [72], where perpetrators of Intimate Partner Violence (IPV), through knowledge of the survivor's authentication

secrets, interacts with the survivor's system with the same privileges as the survivor, without any escalation of privilege. This is similar to $\mathcal{A}_{\text{SUR}}$ and $\mathcal{A}_{\text{INT}}$, with minor differences: we give $\mathcal{A}_{\text{SUR}}$ an extra capability of application layer data information, and $\mathcal{A}_{\text{INT}}$ simply has full device access, and can perform any analysis not usually available to an average user.

Ahmed-Rengers et al. proposed a game-theoretic model capturing power dynamics involved in whistleblowing [8], while discussing the issue of hard and soft power involved. In a similar follow-up work, CoverDrop [9], a secure low-latency way for initial contact and trust establishment during a whistleblowing process, was proposed. Within this paper the authors described a strong adversary, both on the network but also on the infrastructure (e.g. ISPs, cloud providers), as well as assuming compromised devices.

Some readers might correlate our adversaries to those discussed under *endpoint compromise*; in such a case, the adversary would be considered to have full access to device secrets. On a more theoretical-level, Basin and Cremers [20, 21] considered the compromise of both session keys and long-term keys of an agent, at multiple points in the protocol. The authors then presented a framework for analysing security protocols in the presence of adversaries with a wide range of compromise capabilities, and presented a hierarchy of protocol security against those compromise. A local adversary in an IoT system was discussed in [6].

In [174], the proposed adversary model considers an adversary who is able to periodically compromise a device. Here, the adversary considers two device states (compromised and secure), and uses the assumption that the user can manually revoke the key and generate a new one when a persistent attacker is detected.

When a secret key is obtained by an adversary, there are ways to recover from such compromise. *Forward secrecy* [79] is precisely defined for this – essentially, being forward secure means that an adversary having access to one secret does not mean that they have access to *future* secrets. One method to obtain forward security is by *key ratcheting*, that is, regular update to the key; ratcheted encryption in key exchange was proposed in [28], and further methods to obtain forward security

include self healing cryptosystems [155] and time-controlled escrow [37]. A survey of key evolving cryptosystems in the public key setting is described in [71]; this is an interesting scenario as the public key remains the same, yet the secret key changes. Post-compromise security [43] is defined to capture that some practically relevant guarantees can still be achieved after secrets have been compromised. The notion *weak* compromise captures the situation when the adversary can temporarily control the long-term key operations, without actually obtaining the long-term key. *Total* compromise captures the situation when the adversary learns the long-term key. It is shown that some form of post-compromise security can be achieved in a system that was weakly compromised.

An *insider threat* [62, 67] model, for example, considers an adversary who has legitimate, full access to a device. However the goal of such an adversary is usually to exfiltrate data through a monitored and controlled network, usually over a covert channel. This is slightly similar to the goal of the user under $\mathcal{A}_{\text{SUR}}$ as this involves someone (the adversary in the insider threat scenario) to discreetly send data without the knowledge of the network controller. However, an insider threat model, if detected, can usually be mitigated by removing the adversary's access—for the adversaries in our paper, this is not necessarily an obvious, or even possible, task.

Given we consider adversaries that may have a high degree of control of the network, this might seem similar to a censorship resistance system (well reviewed in [100, 165]), where a user wishes to exchange communication with a receiver through a communication channel controlled by a censor who actively attempts to prevent this communication. Censors may have a set of distinguishers which takes in network traffic to be analysed and outputs 'accepted' traffic flows. This is similar to our scenario in Chapter 3 where the adversary wishes to distinguish a distress communication from a normal communication on the network, as we assume communications may take place. However, our adversaries have the additional capability due to their locality to the user.

We summarise these adversaries in Table 6.1.

Adversary type	Local access	Network Control	Power (in relation to user)	Adversary Goal
$\mathcal{A}_{\text{SUR}}$	visual only	limited	high	detect distress
$\mathcal{A}_{\text{INT}}$	✓	✓	high	break confidentiality
UI-bound adversary [72]	✓(limited capability)	✗	high	surveillance
Coercive adversary [179]	✓	✓	high	break confidentiality
Post-compromise adversary [43]	✓	✓	undiscussed	break confidentiality
Burnbox adversary [166]	✓	✓	high	break confidentiality
Whistleblowing adversary	limited	✓	high	detect information
Insider threat [62, 67]	✓	✗	low	exfiltrate data
Censors [100, 165]	✗	✓	usually high	break confidentiality; detect information

Table 6.1: A comparison of strong adversary models in the literature, with elevated levels of access.

6.2 Undetectable Communications

The *Prisoner's Problem* was introduced by Simmons in 1984 [148], which illustrates the scenario in which two prisoners can only communicate through a warden. The warden, in today's world, is an active attacker – they can read all the messages, as well as modify any message before handing it from the sender to the receiver, including stopping the message to be transferred at all. The two prisoners need to create a *subliminal channel*, to be able to communicate their secrets within the restrictions imposed. Simmons further discussed subliminal channels in digital signature schemes [149], and in fact, virtually *all* signature schemes would have the necessary requirements for a subliminal channel to exist.

The prisoner's problem has sparked the beginning of many research areas – for example, *kleptography*, the study of stealing information securely and subliminally, was introduced, which would later lead into the constructions of the Dual EC-DRBG, and instigates discussions of cryptographic backdoors. Of more interest to our thesis, the prisoner's problem also allowed covert channels as a research area, as well as steganography – the embedding of a secret message in a cover message – was developed. While confidentiality in cryptography is about making messages unreadable to an unauthorised party, steganography on the other hand, hides the existence of the message entirely. However, it was argued that a "perfectly secure" steganography, as an analogy to the one-time pad, is impossible [14].

In [74], the authors show that it is possible to implement covert channels —both storage and timing-based— in Android platforms which may lead to undetected document leakage. Covert channels within internet protocols have previously been explored in many ways, including in Google Analytic web cookies in HTTP protocol [86], network time protocols [12], IPv6 [106], as well as through cache [111]. The use of the TLS randomness as a covert channel has been used in [27, 77, 172], and a review of covert channels over https is presented in [102].

The notion of *undetectable communication* specifically for the case of Online Social Networks (OSNs) was explored in [24], where formal definitions are introduced. In [25], the authors proposed a system to allow users to exchange data over existing

web-based sharing platform, while maintaining confidentiality as well as hiding from a casual observer that an exchange of confidential data is taking place.

The scenario of an insider threat wanting to covertly send information as well as someone circumventing censorship are usually achieved by a scheme which ensures stealth. There are numerous steganographic ways in which this can be done. In [103], the authors presented a censorship circumvention system that leverages steganography to hide information in VoIP communication – similar to our scheme in Chapter 3, their method is undetectable while maintaining the statistical property in the object information is embedded in.

For steganographic or censorship resistance purposes, there are several work looking into encryption functions that have pseudorandom ciphertexts, both in the private-key setting [84] and public-key setting [168]. In [29], the authors proposed a method to encode elliptic curve points indistinguishable from uniform random strings, which would assist elliptic curve cryptography to be suitable as a censorship circumvention tool. These schemes are extremely useful, given that they provide an IND$-CPA encryption method which we can use in Chapter 3. However, these work only provide that the output strings *look* random, but doesn't focus on the specific size constraint.

A subliminal or covert channel have been considered to be an "abuse" in cryptography [57], as the cryptosystem is able to be used for a different purpose than for which it is originally intended. There is further work in creating abuse-free cryptosystems, however we disagree with the terminology, as we have seen in examples above that covert channels can be created by design, to allow for (in our opinion) nonnegative actions.

6.3 Coercion Attacks

As we have discussed in Section 4.1.1, coercion attack, sometimes called by the term *rubberhose cryptanalysis* is a physical or emotional attack to bypass security, usually authentication, by threatening or forcing the user to reveal their secrets.

6.3.1 Distress Signalling

A practical solution to a coercion attack to bypass authentication in a device is the use of a second authentication secret, which when used when the user is subject to a 'coercion attack', will alert a trusted third-party of the coercion – in other words, the user is using this covert mechanism to signal distress during an authentication process. For example, *distress cash*, a distress PIN mechanism for electronic cash is proposed in [56], where a tamper-resistant device would store two PINs to activate the device: one for normal operation, and the other to send a distress message via a subliminal channel. This, of course, is under the assumption that the attacker does not have access to transcripts of previous authentication sessions, and as such, would not be able to distinguish a distress secret from a legitimate secret.

However, a threat model for panic passwords is formalised in [41], which in addition to Kerckhoff's principle and the adversary's observational capabilities, the adversary is not bound to a single instance of coercion to the user (*iteration*), and they can also eliminate any strategy that the user may employ through the order in which she reveals the passwords she knows (*forced randomisation*). It is shown that a scenario in which a user is given only two passwords is susceptible to iteration and forced-randomisation attacks. Panic passwords have been implemented in real-life applications, including in home security systems [7].

In [83] the authors presented *funkspiel schemes* – a method in which, assuming the ability of the user in altering a hardware secret, can communicate distress after detecting a break-in attempt, in a way that is undetectable by the adversary. In this scenario, the two security properties considered are *stealth*: the inability of the attacker to determine whether or not the sender has modified their internal state, and *unforgeability*: the inability of the attacker create a response that is regarded as valid by the receiver. The techniques used in this scheme are interesting – a symmetric funkspiel scheme was proposed, by applying a pseudorandom generator which results in pseudorandomness, a property in which the unforgeability property rely on.

A further distress signal and detection mechanism for authentication is introduced in [156], for both users and administrators. The authentication server would

have a duress authentication system, separate to their existing authentication system, and where multiple administrators are involved, the authentication server will connect to the monitoring server at every authentication attempt; the monitoring server is responsible for distinguishing between correct and distress credentials. This architecture is slightly different than ours, as we rely on the immediate communication recipient of the user to distinguish between a distress and a normal communication.

6.3.2 Plausible Deniability

In some encrypted flash drives, when a duress code is entered, the encryption key will be deleted [15]. VeraCrypt (previously TrueCrypt), a disk encryption software, allows "plausible deniability" against an adversary who forces to reveal a password by using hidden volumes and hidden operating systems, where the password of the hidden volume is different than the password of the outer volume [167].

Another scheme which allows another form of deniability is the following: deniable authenticated encryption [127] captures a scenario in which a whistleblower discloses confidential information to a journalist. The journalist requires that the communication is authenticated – that is, the message is indeed from the whistleblower; however, when the journalist is later coerced to reveal the whistleblower's identity, a deniable authenticated encryption scheme will allow the journalist to arbitrarily create fake messages as if they are from the whistleblower, hence the whistleblower can always deny their involvement.

In [32], a neuroscience-based defence against coercion attacks was discussed, using the concept of *implicit learning* from cognitive psychology: the 'password' is planted to the user over a training period, and will be detected upon authentication; however, a user cannot explicitly explain what the password is. Though implicit learning can be considered as part of behavioural biometrics (e.g. walking gait [11], eye movements [26], skin conductance [80]), it is different in the way that this defence is trained – further, this approach enables useful properties such as key revocation as well as the possibility of multiple keys per user, for use on different systems.

Further interdisciplinary approach against coercion attacks is considered in [80], in which the user's emotional status, captured by the measure of their skin and voice conductance, is incorporated into the process of key generation. It is assumed that this method is publicly known, as otherwise, this would lead to a dangerous situation for the user.

Non-repudiation as a security goal directly contrast the notion of deniability, and many proposals have been made in the past to allow a user to deny that a particular cryptographic action has been made. For example, deniable encryption [38] was introduced to mitigate a situation in which the adversary can request – physically or otherwise – the sender's secret key k, as well as random choices r, used in encryption after the ciphertext $c = \textsc{Enc}(m, r)$ was sent. A deniable encryption algorithm allows Alice to find a different message m', as well as new random choices r' such that $c' = \textsc{Enc}(m', r')$ *appears* the same as the corresponding ciphertext c. The original work however, revealed that this is far from practical as the key length has to be at least the message length, but this has been improved in recent years [101, 163]. The survey [177] provides an overview of data confidentiality against coercive adversaries via deniable encryption.

On the protocol level, another breakthrough in deniability came with the proposal of Off-the-Record Messaging (OTR) [34] where authorship of a particular message can no longer be linked to the original sender. The trick to this mechanism is short-lived session keys, as well as using a MAC key that is shared with the receiver; with the MAC key later published, the original message can be also signed by the receiver or anyone else. In addition, session keys that are no longer used are deleted, allowing forward secrecy, but also deny authorship. Similarly, deniable authenticated encryption [127] allows the receiver of a particular message to arbitrarily create fake messages as if they were from the original sender, hence the sender can always deny their involvement. A recent work [130] has looked into how humans perceive these 'proofs' on a practical level, through a courtroom situation.

Many attempts at maintaining confidentiality against a coercive adversary comes hand-in-hand with information hiding, which allows *plausible deniability*: a

situation such that there "be no irrefutable evidence concerning a disputed event or action" [132]. For example, Veracrypt [167], a fork of the discontinued Truecrypt project [164], is an open source encryption software that comes with the feature that allows the user to create a hidden operating system whose existence should be impossible to prove, as it lies within an existing VeraCrypt volume, and any free space on a VeraCrypt volume is filled with random data when it is created. In addition, if coerced, the user can input password to the outer volume which reveals non-sensitive information. Thus, you will not have to decrypt or reveal the password for the hidden volume.

Hidden volumes are an example of a steganographic file system [13], a storage mechanism designed to give the user a very high level of protection against being compelled to disclose its contents. This work is followed by other systems [19, 112, 158].

Other work on authenticating under duress include a distress pin [41], which includes a threat model of categorisation of coercion, and the Funkspiel scheme [83], assuming you can alter the hardware of the device.

The methods above are different from Nakula, as we're not trying to deny a particular action has been taken through ambiguity; or worse, knowingly making an incorrect statement or entering an authentication secret that doesn't reflect the true content of the data they hold.

The idea of using a trusted third party to store an encryption key has been previously discussed on both an informal and an academic setting [107, 140], as a response to a capture or a device seizure scenario. In addition, [175] also uses an entity that issues a remote wipe command should the user report that a device is stolen or lost.

6.3.3 Solutions to the Border Crossing Problem

The password manager OnePassword published a feature called Travel Mode [65], to protect ultra-sensitive data when someone crosses a border. A user can turn on travel mode in advance of travel, which removes (not hides) some stored passwords that were marked as sensitive. If they are asked to open their password manager,

they proceed with authentication as normal but the border agents wouldn't be able to tell that there are data that are missing. To recover their removed passwords, the user can then authenticate themselves and turn off travel mode. Of course, this doesn't work with $\mathcal{A}_{\text{INT}}$ that can request the user to authenticate themselves and turn off travel mode when the adversary is present.

The idea to 'forget' a key for border crossing has been mentioned in [140], and upon completing the work in Chapters 4 and 5 we have discovered that the idea of using Shamir Secret Sharing for border crossing was introduced in a position paper [17]. Though these ideas are similar to ours, our architectures are distinct, especially our use of asymmetric cryptography is novel.

Lastly, the concept of using encryption without providing someone the key to deny access to the information is far from recent [173] and has been used maliciously in the form of malware (e.g., [138]) but in this thesis we use this technique to our advantage.

A similar work to Chapter 4, BurnBox [166], was published to propose a solution against compelled access (including, but not limited to, border crossings). The approach of the BurnBox is similar, where a user 'revokes' access to specific files before an adversary is able to have access to the device. BurnBox also uses a combination of asymmetric and symmetric encryption similar to ours, where a file is stored encrypted (with a symmetric key), and revocation is done by encrypting the key (along with other information) with a public key that a revocation party has the secret key to.[1] BurnBox is designed to also work with cloud storage systems, and provide security guarantee against passive cloud adversaries, which is the difference to our work.

However, when looking at the details of the approach, our method is more protocol-centred, focusing on different sessions that a (public) key can generate and allows a user to have multiple sessions and lock the data multiple times. Chapter 5 also discusses in detail how the recovery process can take place, which BurnBox

[1]From this description, you might be noticing the high similarity to that of our work. Indeed, we were not aware of this literature as Chapter 4 was written and published.

does not discuss. BurnBox is one step ahead when it comes to filename revocation and not only file content as we did in Chapter 4.

6.4 Secure Deletion

We note that the adversary model introduced as $\mathcal{A}_{\text{INT}}$ reduces to what is usually considered for secure deletion: the goal of $\mathcal{A}_{\text{INT}}$ is to recover deleted data objects after being given access to a physical medium that contained some representation of the data objects.

In most systems, data is not securely deleted by default. Throughout this thesis, we assumed 'deletion' to mean *secure* deletion, where data is made irrecoverable from a physical medium [128]. There has been many work dedicated to secure deletion: in [58], it is noted that cryptographic protocols forgetting information is usually assumed; in turn, they defined and constructed a *secure erasable memory implementation*, which turns any storage device into a storage device that can selectively forget. In [33], the authors uses a block cipher to forget information rather than protect it. Many other schemes, as well as adversary models of device capture are discussed in [128]. In [179], the authors proposed a method to securely delete data under coercion; this is done through a deletion password, which when entered, will delete the data through deleting the relevant encryption key stored within the Trusted Platform Module (TPM), which can be verified through a TPM quote.

6.5 Distance Bounding and Location Verification

Distance bounding was first introduced in [35] as a solution to the mafia fraud attack. There has been numerous protocols proposed since [81], including one designed for location verification [150] by using multiple verifiers, allowing triangulation. The security of different protocols have been thoroughly analysed [18, 40, 49].

Measuring distance over the internet (for location estimation or otherwise) has been studied in a nonadversarial setting [52, 70, 115]. A distance bounding protocol over the internet with the goal to detect internet hijacking has been

proposed [16]. Though experiments show promise, they were only conducted within a 'small' geographical region, with servers located in one country and it remains an open question how their protocol can perform in a larger context; hence, we utilise the CPV tool in [4].

In [180], the authors propose a mechanism where colocated devices with bluetooth connections can enable mutual location proofs and (privately) uploaded to a server. This is of course a possibility to use in our architecture, however this assumes an even high scalability of the deployment of Nakula, in terms of the number of users.

7 Discussion

In this chapter, we present points of discussion throughout this thesis that we believe warrant more clarification and critical reflection. We start with some self-reflection as we discuss the limitations of this thesis, before we move toward future work in this domain.

7.1 Reflections and Limitations

The clear limitation of this research, though is motivated by concrete situations, does not involve case-studies, interviews, or any collaboration with the groups of users that the systems are designed for. Other than through informal conversations and exposing ourself to a diverse set of work within the different domains – both inside and outside of academia – we note that this research has been completed only within the Department of Computer Science, and we accept that this is a shortcoming of our methodology; however, we believe that this work can be a starting point for discussion and a wider research or implementation where other stakeholders are, not only consulted, but also contribute to the design – a "relational dynamic" with the users is necessary [153].

Our adversary model is a little bit different than the "authenticated but adversarial" UI-bound adversary in [104]; it is weaker, in the sense that $\mathcal{A}_{\text{SUR}}$

does not necessarily have access to the device at any point, but stronger as they are able to obtain application layer data. Indeed, one critique of this model is that it does not necessarily capture the capabilities of an actual human attacker: domestic abusers have full device access, and may limit the user from using devices (see quote from [125] below). We wanted to make an adversary have the strongest capability possible, while being able to reach our security goal (as we believe, this is a common approach in computer security); this is another limitation in our methodology. Similarly as the previous paragraph, future work can build on this, with conversations with relevant stakeholders.

Though we have abstracted away the motivating situations within the system design and adversary model to allow for a wider range of applications, we second that, as [135] stated, protocol design should begin "with the unique needs of the population the protocol is meant to serve". Designing a distress signalling mechanism for survivors of domestic abuse required significantly different specifications from designing a similar mechanism for whistleblowers or espionage agents. As we have stated in Chapter 5, an authentication mechanism for a multinational company is not comparable to a personal deployment. We have left many holes in the chapter as we aim to generalise the problem as far as possible, at the cost of not being able to address specific issues.

We also mention our positionality[1]: the literature in which we have gathered ideas and information from are concentrated primarily between North America and western Europe, specifically two countries: the United Kingdom and the United States; the former due to the residency of the author, and the latter due to the far-reaching aspect of its politics and media. We did not discuss how these situations may be different in other countries. An Indonesian government research on gender-based violence, following Covid-19, states [125]:

> "many women and girls in many developing and underdeveloped countries may not have the capacity or access to mobile phones, computers, or internet access that would facilitate them in reaching for help through

[1] As Lila Abu-Lughod states, "every view is a view from somewhere and every act of speaking a speaking from somewhere" [5]

> online services in [quarantine times]. Women and girls in many contexts often have less access than men and boys to the internet or other forms of technology, consequently impairing their ability to access remote services. Based on this fact, the issue is even more complex as risks would be further compounded in the cases of women and girls living in households affected by [gender-based violence]. Perpetrators would likely limit the victims' access to various technology forms."

Does designing an online distress reporting system make sense at all, if the demographic it was designed for do not even have access to the means of getting online? At the same time, we are also aware of situations in other countries where women use technology to seek help or escape, allowing them an opportunity they previously did not have [87].

Of course, domestic abuse and coercive control is a pervasive problem around the world. So are the separate problems of data collection, mass surveillance, border crossing, immigration and law enforcement. We are far from claiming to solve any societal problem that have layers of intersectionality. We will never be able to do that using security protocols.

7.2 Future Work

There are plenty of avenues in which the research in this thesis can be developed further. First, our approach in Chapter 3 referred to the UI and enrolment problem as out of scope from this thesis, even though that is where the possibility of detection, and hence harm, can occur. We also note that the methods that we mention for the UI are specifically visual, or audiovisual – we have not covered the possibility of use of the system in some stress cases, including to cater those with visual impairment. An immediate line of work is to present a specification in the UI design for a specific population, so that not only the security of the system can hold, but also the safety of the users. Slupska in [153] calls from a move from security by design towards *safety* by design, especially for vulnerable groups; any follow-up work should be guided by this.

In a more cryptographic domain, our work in Chapter 3 relies on a modified El-Gamal, with a 1-2 ciphertext expansion size. To the best of our knowledge, there is no public key encryption with pseudorandom ciphertext that is close to being implementable in our system. Of course, finding such an encryption function (should it exist) would be another thesis, or a few theses, on its own, but an interesting area is to use other client-side randomness for communication. Note that this technique is not only useful in our case, but also as a wider censorship-resistance tool, as have been noted in [27, 172]. Looking into more recent protocols such as QUIC would be a good place to start, but objects like headers with many random-looking element can also be considered.

An interesting avenue of research is to consider the use of secret sharing mechanisms [145], threshold encryption, or secure multi-party computation, within the Nakula ecosystem. Indeed, this idea has been explored in [17], when considering the border crossing situation, but with a slightly different system model. Can we distribute risk based on distributing trust? Can we increase flexibility for the user by distributing secrets in as many shares as possible without compromising security?

We presented a high-level security analysis for mechanisms we introduce in Chapter 5. Further details about the cryptographic interaction between our protocols and U2F, CVP, or distance bounding can be specified, as devils are usually in the tiniest details, warranting a full security analysis.

Lastly, we also note that the systems in this thesis relies on trusted third parties: the DCP and webservers in Chapter 3, as well as the backend and verifiers in Chapters 4 and 5. To achieve our security guarantees, it is required that they are trusted. However, we note that these points hold immense responsibility, and the system can be targeted by other actors. If the DCP holds any personal information, can they turn adversarial, or be subject to compromise by an adversary? Can this be monetised by a someone having access to these systems? We note that the information they hold is of high sensitivity value due to the nature of the system, as it holds information about a group that is enough in a vulnerable position to require

a distress signalling mechanism. An interesting future work would implement privacy mechanisms on these servers so that no one party has access to sensitive information.

8 Conclusion

This thesis started with answering the following questions: what is an adversary, and what do we mean by privileged access? We focus on two adversaries, a surveillant adversary and an intrusive adversary, characterised by their access to some access to the user's endpoint secrets, full control control of the communications channels, locality to the user, and benefitting from an asymmetric power dynamic.

In Chapter 3, we defined a surveillant adversary who has access to the user's output device, but also application layer contents of the user's TLS communications. We introduced distress signalling as a security goal for the user and formalised the adversary's capabilities in a security game where it aims to detect distress signals. To achieve this, the user utilises participating webservers, who act as to relay the distress between the user and the trusted third party. The distress is embedded within an encoding function whose output is then forwarded via the TLS client nonce towards the webserver. Using TLS and public key encryption allows for scalability, and we showed that even if the webserver has to decrypt every TLS communication that arrives, they can do so with minimal overhead. We performed a full security analysis of the architecture.

Chapter 4 saw us introduce an intrusive adversary, who has full physical access to the device for a finite amount of time, as well as having access to the user's authentication secrets. We designed an architecture to maintain confidentiality

against such adversary, where the user does not have to lie or hide the information, by ensuring the user does not have the means to decrypt the necessary building blocks, as authentication has to depend on physical location that the adversary does not have access to, and not only a password or biometric that the user has. We discuss four location-based authentication mechanisms in Chapter 5 and provided examples of use-cases with varying levels of architecture and security.

We proceeded in discussing related work and where our research lies within the academic landscape in Chapter 6. We critically discuss our work in Chapter 7, along with some reflections and limitations, as we point out that collaboration between different stakeholders is necessary for further research and deployment, and we discuss future directions that this research can continue on.

A

Further Cryptographic Building Blocks

A.1 IND$-CPA Encryption Scheme

Let U_k denote the uniform distribution of bitstrings of size k. The well-known notion of CPA security for public key encryption describes an adversary who, after providing two messages m_0, m_1, aims to distinguish between $\text{ENC}_{pk}(m_0)$ and $\text{ENC}_{pk}(m_1)$, while having access to pk. Though this is a very useful notion, this does not imply that each ciphertext is indistinguishable from a randomly chosen string. To do so, we consider a game which an adversary, after providing a message m is given either $\text{ENC}_{pk}(m)$ or a uniformly chosen string of the same length, as well as pk. This is formalised in the following experiment, as described in [168]:

Definition 4. Let $\Pi = (\text{GEN}, \text{ENC}, \text{DEC})$ be a public encryption scheme. Consider the following experiment ExpIND$-CPA, where an adversary $\mathcal{A}$ is given access to an oracle which is either:

- ENC_{pk} for $(pk, sk) \leftarrow \text{GEN}(1^k)$, in other words, an oracle which, given a message m, returns $\text{ENC}_{pk}(m)$.
- $\$(\cdot) = U_{|\text{ENC}_{pk}(\cdot)|}$, that is, an oracle which returns a uniformly selected output of the appropriate length.

The adversary is also given access to pk used by its oracle to answer queries. The IND$-CPA advantage of $\mathcal{A}$ against Π is defined as

$$\mathrm{Adv}_{\pi}^{\mathrm{CPA}}(\mathcal{A}, n) = | \Pr_{(pk,sk)\leftarrow \textsc{Gen}(1^n), r\leftarrow\{0,1\}^*}[\mathcal{A}_r^{\textsc{Enc}_{pk}}(pk) = 1] - \Pr_{(pk,sk),r}[\mathcal{A}_r^{\$}(pk) = 1]|.$$

where $A(t, q, l)$ be the set of adversaries $\mathcal{A}$ which make $q(n)$ queries to the oracle totalling at most $l(n)$ bits and run for $t(n)$ steps. Then Enc is said to be indistinguishable from random bits under chosen plaintext attack (IND$-CPA) if for every probabilistic polynomial-time adversary $\mathcal{A}$,

$$\max_{\mathcal{A}\in A(t,q,l)} \{\mathrm{Adv}_{\pi}^{\mathrm{CPA}}(\mathcal{A}, n)\} \leq \epsilon(n)$$

for some negligible function ϵ.

The ExpIND$-CPAdescribes a scenario where the adversary is able to submit one message to the $\textsc{Enc}_{pk}$ oracle. Before we look into the multiple message version of IND$-CPA, we look at an extension of the CPA experiment, called the LR-oracle experiment [97, Definition 11.5], which models security when multiple messages are encrypted using the same public key.

Definition 5. Consider the *left or right* oracle $\mathrm{LR}_{pk,b}$ that, on input a pair of equal length messages m_0, m_1 computes the ciphertext $c \leftarrow \textsc{Enc}_{pk}(m_b)$ and returns c. The attacker is allowed to query this oracle many times.

Let $\Pi = (\textsc{Gen}, \textsc{Enc}, \textsc{Dec})$ be a public key encryption scheme and let $\mathcal{A}$ be an adversary. The LR-oracle experiment $\mathrm{PubK}_{\mathcal{A},\Pi}^{\mathrm{LR-CPA}}(n)$:

1. $\textsc{Gen}(1^n)$ is run to obtain (pk, sk). A uniform bit $b \in \{0, 1\}$ is chosen.
2. The adversary $\mathcal{A}$ is given input pk and oracle access to $\mathrm{LR}_{pk,b}(\cdot, \cdot)$ and outputs a bit b'.

The output of the experiment is defined to be 1 if $b' = b$ and 0 otherwise.

Now, a public key encryption scheme $(\textsc{Gen}, \textsc{Enc}, \textsc{Dec})$ is said to have *indistinguishable multiple encryptions* if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function ϵ such that:

$$\Pr[\text{PubK}^{\text{LR}-\text{CPA}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \epsilon(n).$$

We now consider the case where the adversary is able to submit multiple equal-sized messages $m_1, \ldots, m_q$ and receives $\vec{C} = c_1, \ldots, c_q$ where $\vec{C}$ is a distribution that is either obtained by $\textsc{Enc}_{pk}(m_i)$ or obtained from $U_{|\textsc{Enc}_{pk}(m)|}$. That is, upon receiving $\vec{C}$, the adversary cannot distinguish whether or not they are receiving a distribution obtained from $\textsc{Enc}$ using the same public key pk, or a distribution of random numbers. We capture this in the following game:

Definition 6. Let $\Pi = (\textsc{Gen}, \textsc{Enc}, \textsc{Dec})$ be a public key encryption scheme. The indistinguishability against multiple random bits experiment $\text{ExpIND\$-multCPA}_{\mathcal{A},\Pi}(n)$:

Challenger		**Adversary**
$(pk, sk) \leftarrow \textsc{Gen}(1^n)$		
	$\xrightarrow{pk}$	
		$\vec{M} = (m_1, \ldots, m_q)$,
	$\xleftarrow{\vec{M}}$	
$\vec{C_0} = (u_1, \ldots, u_q)$ where $u_i \leftarrow U_{\lvert\textsc{Enc}_{pk}(m_i)\rvert}$		
$\vec{C_1} = (\textsc{Enc}_{pk}(m_i))_{i=1}^{q}$		
$b \leftarrow_R \{0,1\}$		
	$\xrightarrow{\vec{C_b}}$	
		Guess $b' \in \{0,1\}$
	$\xleftarrow{b'}$	

The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise. We define the IND\$-multCPAadvantage of $\mathcal{A}$ against Π to be

$$\text{Adv}^{\text{CPA}}_{\Pi}(\mathcal{A}, n) = |\Pr[\mathcal{A} \text{ outputs } 1 | b = 0] - \Pr[\mathcal{A} \text{ outputs } 1 | b = 1]|.$$

We say that a public key encryption scheme $\Pi = (\textsc{Gen}, \textsc{Enc}, \textsc{Dec})$ is said to have *indistinguishable multiple encryptions from random bits* if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function ϵ such that:

$$\Pr[\text{ExpIND\$-multCPA}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \epsilon(n).$$

The above definition of $\text{ExpIND\$-multCPA}_{\mathcal{A},\Pi}(n)$ is a natural extension of $\text{ExpIND\$-CPA}_{\mathcal{A},\Pi}(n)$, exactly how the $\text{PubK}^{\text{LR}-\text{CPA}}_{\mathcal{A},\Pi}(n)$ experiment in an extension of $\text{PubK}^{\text{CPA}}_{\mathcal{A},\Pi}(n)$ experiment. In the CPA sphere, a CPA-secure public key encryption system automatically guarantees indistinguishability under multiple encryptions:

Theorem 1. *If a public key encryption scheme* $(\textsc{Gen}, \textsc{Enc}, \textsc{Dec})$ *is CPA-secure, then it also has indistinguishable multiple encryptions.*

Proof. Theorem 11.6 [97]. □

Given the natural extension of CPA security to indistinguishability under multiple encryptions, we argue that we can extend the IND\$-CPA to IND\$-multCPA in a similar manner. However, the proof for this is outside the scope of this thesis.

B

Notation and Abbreviations

$\mathcal{A}_{\text{INT}}$	intrusive adversary
$\mathcal{A}_{\text{SUR}}$	surveillant adversary
k_{AB}	symmetric key k between two parties A and B
(pk_A, sk_A)	asymmetric keypair for party A with public key pk and secret key sk
$\text{ENC}_k(\cdot)$	encryption function using key k
$\text{DEC}_k(\cdot)$	decryption function using key k
$\text{SIGN}_k(\cdot)$	signature function with key k
$h(\cdot)$	hash function
$\text{MAC}_k(\cdot)$	A MAC function with key k
n_A	nonce chosen by party A
$\mathbb{Z}_p$	the group of integers mod p under multiplication
$\mathbb{F}_q$	a finite field of q elements
U_k	uniform distribution of bitstrings of size k
$\leftarrow_R$	chosen from random

Table B.1: Notation used throughout thesis

List of Abbreviations

AEAD . . . . . Authenticated Encryption with Associated Data

CA . . . . . Certificate Authority

CCTV . . . . . Closed-circuit television

CPA . . . . . Chosen plaintext attack

CPV . . . . . Client Presence Verification

CDH . . . . . Computational Diffie-Hellman

DCP . . . . . Distress Coordination Point

DDH . . . . . Decisional Diffie Hellman

DLP . . . . . Discrete Logarithm Problem

E2EE . . . . . End-to-end Encrypted

GNSS . . . . . Global Navigation Satellite System

GPS . . . . . Global Positioning System

HTTP . . . . . Hypertext Transfer Protocol

HTTPS . . . . Hypertext Transfer Protocol Secure

IPS . . . . . Intimate Partner Surveillance

ISP . . . . . Internet Service Provider

KDF . . . . . Key Derivation Function

MAC . . . . . Message Authentication Code

NIST . . . . . National Institute of Standards and Technology

PPT . . . . . Probabilistic Polynomial Time

RFID . . . . . Radio Frequency Identification

SSL . . . . . Secure Socket Layer

TPM . . . . . Trusted Platform Module

TLS . . . . . Transport Layer Security

U2F . . . . . Universal Second Factor

UI . . . . . User Interface

References

[1] 1984. *Police and Criminal Evidence Act 1984, Section 41.* Retrieved 27 September 2023 from https://www.legislation.gov.uk/ukpga/1984/60/section/41

[2] 2006. *Fraud Act 2006 c. 35.* Retrieved 3 October 2023 from https://www.legislation.gov.uk/ukpga/2006/35/contents

[3] 2015. *Serious Crime Act 2015 c. 9, Section 76.* Retrieved 15 September 2023 from https://www.legislation.gov.uk/ukpga/2015/9/section/76/2015-12-29

[4] AbdelRahman Abdou, Ashraf Matrawy, and P. C. van Oorschot. 2017. CPV: Delay-Based Location Verification for the Internet. *IEEE Transactions on Dependable and Secure Computing* 14, 2 (2017), 130–144. https://doi.org/10.1109/TDSC.2015.2451614

[5] Lila Abu-Lughod. 2012. WRITING AGAINST CULTURE. *Andamios (Mexico City, Mexico)* 9, 19 (2012), 129–157.

[6] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. 2020. Peek-a-Boo: I See Your Smart Home Activities, Even Encrypted!. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20).* Association for Computing Machinery, New York, NY, USA, 207–218. https://doi.org/10.1145/3395351.3399421

[7] ADT. 2018. *Safewatch Pro 3000 System Manual.* Technical Report. ADT.

[8] Mansoor Ahmed-Rengers, Ross Anderson, Darija Halatova, and Ilia Shumailov. 2020. Snitches Get Stitches: On the Difficulty of Whistleblowing. arXiv:2006.14407 [cs.CY]

[9] Mansoor Ahmed-Rengers, Diana Vasile, Daniel Hugenroth, Alastair Beresford, and Ross Anderson. 2022. CoverDrop: Blowing the Whistle Through A News App. *Proceedings on Privacy Enhancing Technologies* 2022 (04 2022), 47–67. https://doi.org/10.2478/popets-2022-0035

[10] Women's Aid. 2022. Cover your tracks online. https://www.womensaid.org.uk/cover-your-tracks-online/

[11] Heikki J. Ailisto, Mikko Lindholm, Jani Mantyjarvi, Elena Vildjiounaite, and Satu-Marja Makela. 2005. Identifying people from gait pattern with accelerometers. In *Biometric Technology for Human Identification II (Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 5779)*, Anil K. Jain and Nalini K. Ratha (Eds.). 7–14. https://doi.org/10.1117/12.603331

[12] Aidin Ameri and Daryl Johnson. 2017. Covert Channel over Network Time Protocol. In *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy (ICCSP '17)*. Association for Computing Machinery, New York, NY, USA, 62–65. https://doi.org/10.1145/3058060.3058082

[13] Ross Anderson, Roger Needham, and Adi Shamir. 1998. The Steganographic File System. In *Information Hiding*, David Aucsmith (Ed.). Vol. 1525. Springer Berlin Heidelberg, Berlin, Heidelberg, 73–82. https://doi.org/10.1007/3-540-49380-8_6 Series Title: Lecture Notes in Computer Science.

[14] R. J. Anderson and F. A. P. Petitcolas. 1998. On the Limits of Steganography. *IEEE Journal on Selected Areas in Communications* 16, 4 (May 1998), 474–481. https://doi.org/10.1109/49.668971

[15] Apricorn. 2017. *Aegis Secure Key 3.0 User's Manual*. Technical Report.

[16] Ghada Arfaoui, Gildas Avoine, Olivier Gimenez, and Jacques Traoré. 2021. How Distance-Bounding Can Detect Internet Traffic Hijacking. In *Cryptology and Network Security*, Mauro Conti, Marc Stevens, and Stephan Krenn (Eds.). Springer International Publishing, Cham, 355–371.

[17] Erinn Atwater and Ian Goldberg. 2018. Shatter Secrets: Using Secret Sharing to Cross Borders with Encrypted Devices. In *Security Protocols XXVI (Lecture Notes in Computer Science)*, Vashek Matyáš, Petr Švenda, Frank Stajano, Bruce Christianson, and Jonathan Anderson (Eds.). Springer International Publishing, Cham, 289–294. https://doi.org/10.1007/978-3-030-03251-7_33

[18] Gildas Avoine, Muhammed Ali Bingöl, Ioana Boureanu, Srdjan čapkun, Gerhard Hancke, Süleyman Kardaş, Chong Hee Kim, Cédric Lauradoux, Benjamin Martin, Jorge Munilla, Alberto Peinado, Kasper Bonne Rasmussen, Dave Singelée, Aslan Tchamkerten, Rolando Trujillo-Rasua, and Serge Vaudenay. 2018. Security of Distance-Bounding: A Survey. *ACM Comput. Surv.* 51, 5, Article 94 (sep 2018), 33 pages. https://doi.org/10.1145/3264628

[19] Austen Barker, Staunton Sample, Yash Gupta, Anastasia McTaggart, Ethan L. Miller, and Darrell D. E. Long. 2019. Artifice: A Deniable Steganographic File System. In *9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19)*. USENIX Association, Santa Clara, CA.

[20] David Basin and Cas Cremers. 2010. Degrees of Security: Protocol Guarantees in the Face of Compromising Adversaries. In *Computer Science Logic*, Anuj Dawar and Helmut Veith (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–18.

[21] David Basin and Cas Cremers. 2010. Modeling and Analyzing Security in the Presence of Compromising Adversaries. In *Computer Security – ESORICS 2010 (Lecture Notes in Computer Science)*, Dimitris Gritzalis, Bart Preneel,

and Marianthi Theoharidou (Eds.). Springer, Berlin, Heidelberg, 340–356. https://doi.org/10.1007/978-3-642-15497-3_21

[22] Andrej Bauer. [n. d.]. *Random Art*. Random Art. Retrieved 27 September 2023 from http://www.random-art.org/

[23] BBC. 2019. *US Domestic Abuse Victim Pretends to Order Pizza to Alert 911*. BBC News. Retrieved 27 September 2023 from https://www.bbc.co.uk/news/world-us-canada-50513706

[24] Filipe Beato, Emiliano De Cristofaro, and Kasper B. Rasmussen. 2014. Undetectable communication: The Online Social Networks case. In *2014 Twelfth Annual International Conference on Privacy, Security and Trust*. IEEE, Toronto, ON, Canada, 19–26. https://doi.org/10.1109/PST.2014.6890919

[25] Filipe Beato, Iulia Ion, Srdjan Čapkun, Bart Preneel, and Marc Langheinrich. 2013. For Some Eyes Only: Protecting Online Information Sharing. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy* (San Antonio, Texas, USA) *(CODASPY '13)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/2435349.2435351

[26] Roman Bednarik, Tomi Kinnunen, Andrei Mihaila, and Pasi Fränti. 2005. Eye-Movements as a Biometric. In *Proceedings of the 14th Scandinavian Conference on Image Analysis (SCIA'05)*. Springer-Verlag, Berlin, Heidelberg, 780–789. https://doi.org/10.1007/11499145_79

[27] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. 2014. Security of Symmetric Encryption against Mass Surveillance. In *Advances in Cryptology – CRYPTO 2014*, Juan A. Garay and Rosario Gennaro (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–19.

[28] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. 2017. *Ratcheted Encryption and Key Exchange: The Security of Messaging.* Technical Report. Cham. 619–650 pages.

[29] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. 2013. Elligator: Elliptic-Curve Points Indistinguishable from Uniform Random Strings. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13).* Association for Computing Machinery, New York, NY, USA, 967–980. https://doi.org/10.1145/2508859.2516734

[30] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. 2016. Time-Lock Puzzles from Randomized Encodings. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science* (Cambridge, Massachusetts, USA) *(ITCS '16).* Association for Computing Machinery, New York, NY, USA, 345–356. https://doi.org/10.1145/2840728.2840745

[31] Jeremiah Blocki, Benjamin Harsha, and Samson Zhou. 2018. On the Economics of Offline Password Cracking. In *2018 IEEE Symposium on Security and Privacy (SP).* IEEE Computer Society, Los Alamitos, CA, USA, 853–871. https://doi.org/10.1109/SP.2018.00009

[32] Hristo Bojinov, Daniel Sanchez, Paul Reber, Dan Boneh, and Patrick Lincoln. 2012. Neuroscience Meets Cryptography: Designing Crypto Primitives Secure Against Rubber Hose Attacks. In *21st USENIX Security Symposium (USENIX Security 12).* USENIX Association, Bellevue, WA, 129–141.

[33] Dan Boneh and Richard J. Lipton. 1996. A Revocable Backup System. In *Proceedings of the 6th Conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6* (San Jose, California) *(SSYM'96).* USENIX Association, USA, 9.

[34] Nikita Borisov, Ian Goldberg, and Eric Brewer. 2004. Off-the-Record Communication, or, Why Not to Use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society* (Washington DC, USA) *(WPES '04)*. Association for Computing Machinery, New York, NY, USA, 77–84. https://doi.org/10.1145/1029179.1029200

[35] Stefan Brands and David Chaum. 1994. Distance-Bounding Protocols. In *Advances in Cryptology — EUROCRYPT '93*, Tor Helleseth (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 344–359.

[36] Simone Browne. 2015. "What Did TSA Find in Solange's Fro"?: Security Theater at the Airport. In *Dark Matters: On the Surveillance of Blackness*. Duke University Press. https://doi.org/10.1215/9780822375302-005

[37] Mike Burmester, Yvo Desmedt, and Jennifer Seberry. 1998. Equitable Key Escrow with Limited Time Span (or, How to Enforce Time Expiration Cryptographically) Extended Abstract. In *Advances in Cryptology – ASIACRYPT '98 (Lecture Notes in Computer Science)*, Kazuo Ohta and Dingyi Pei (Eds.). Springer, Berlin, Heidelberg, 380–391. https://doi.org/10.1007/3-540-49649-1_30

[38] Rein Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. 1997. Deniable Encryption. In *Advances in Cryptology: CRYPTO '97*, Burton S. Kaliski (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 90–104.

[39] R. Chandramouli and N. Memon. 2001. Analysis of LSB Based Image Steganography Techniques. In *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*, Vol. 3. IEEE, Coimbatore, India, 1019–1022 vol.3. https://doi.org/10.1109/ICIP.2001.958299

[40] Tom Chothia, Joeri de Ruiter, and Ben Smyth. 2018. Modelling and Analysis of a Hierarchy of Distance Bounding Attacks. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 1563–1580.

[41] Jeremy Clark and Urs Hengartner. 2008. Panic Passwords: Authenticating under Duress. In *Proceedings of the 3rd Conference on Hot Topics in Security* (San Jose, CA) *(HOTSEC'08)*. USENIX Association, USA, Article 8, 6 pages.

[42] Google Code. 2011. *SecreTwit*. Google.

[43] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. 2016. On Post-compromise Security. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. IEEE, Lisbon, Portugal, 164–178. https://doi.org/10.1109/CSF.2016.19

[44] Path Community. 2022. *Path – Comunity Safety*. https://www.pathcommunity.co/

[45] Cambridgeshire Constabulary. 2023. *Robber walked victim to ATM and demanded cash*. Cambridgeshire Constabulary. Retrieved 19 September 2023 from https://www.cambs.police.uk/news/cambridgeshire/news/2023/august/robber-walked-victim-to-atm-and-demanded-cash/

[46] Sophia Cope, Amul Kalia, Seth Schoen, and Adam Schwartz. 2017. *Digital Privacy at the U.S. Border: Protecting the Data On Your Devices*. Technical Report. Electronic Frontier Foundation. https://www.eff.org/wp/digital-privacy-us-border-2017

[47] Joseph Cox. 2019. China Is Forcing Tourists to Install Text-Stealing Malware at Its Border. https://www.vice.com/en/article/7xgame/at-chinese-border-tourists-forced-to-install-a-text-stealing-piece-of-malware.

[48] Sadie Creese, Michael Goldsmith, and Bill Roscoe. 2003. The attacker in ubiquitous computing environments: formalising the threat model. In *Formal Aspects of Security and Trust*. Springer-Verlag, Pisa, Italy. https://api.semanticscholar.org/CorpusID:61815860

[49] Cas Cremers, Kasper B. Rasmussen, Benedikt Schmidt, and Srdjan Capkun. 2012. Distance Hijacking Attacks on Distance Bounding Protocols. In *2012 IEEE Symposium on Security and Privacy*. IEEE, San Francisco, CA, 113–127. https://doi.org/10.1109/SP.2012.17

[50] Mike Crittenden. 2019. How to Use Shorkeys. https://github.com/crittermike/shortkeys/wiki/How-To-Use-Shortkeys

[51] US Customs and Border Protection. 2018. CBP Directive No 3340-049A. Subject: Border Search of Electronic Devices. https://www.cbp.gov/document/directives/cbp-directive-no-3340-049a-border-search-electronic-devices

[52] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. 2004. Vivaldi: A Decentralized Network Coordinate System. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Portland, Oregon, USA) *(SIGCOMM '04)*. Association for Computing Machinery, New York, NY, USA, 15–26. https://doi.org/10.1145/1015467.1015471

[53] Quynh Dang. 2012. Recommendation for Applications Using Approved Hash Algorithms. NIST Special Publication 800-107 Revision 1 (Aug. 2012).

[54] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. 2014. The Tangled Web of Password Reuse. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014*. The Internet Society, San Diego, California, USA. https://www.ndss-symposium.org/ndss2014/tangled-web-password-reuse

[55] Data Center Knowledge. 2017. Google Data Center FAQ.

[56] George Davida, Yair Frankel, Yiannis Tsiounis, and Moti Yung. 1997. Anonymity Control in E-cash Systems. In *Financial Cryptography (Lecture*

Notes in Computer Science), Rafael Hirschfeld (Ed.). Springer, Berlin, Heidelberg, 1–16. https://doi.org/10.1007/3-540-63594-7_63

[57] Yvo Desmedt. 1990. Abuses in Cryptography and How to Fight Them. In *Advances in Cryptology — CRYPTO' 88*, Shafi Goldwasser (Ed.). Springer New York, New York, NY, 375–389.

[58] Giovanni Di Crescenzo, Niels Ferguson, Russell Impagliazzo, and Markus Jakobsson. 1999. How To Forget a Secret. In *STACS 99 (Lecture Notes in Computer Science)*, Christoph Meinel and Sophie Tison (Eds.). Springer, Berlin, Heidelberg, 500–509. https://doi.org/10.1007/3-540-49116-3_47

[59] W. Diffie and M. Hellman. 1976. New directions in cryptography. *IEEE Transactions on Information Theory* 22, 6 (1976), 644–654. https://doi.org/10.1109/TIT.1976.1055638

[60] Christos Dimitrakakis and Aikaterini Mitrokotsa. 2015. Distance-Bounding Protocols: Are You Close Enough? *IEEE Security & Privacy* 13, 4 (2015), 47–51. https://doi.org/10.1109/MSP.2015.87

[61] D. Dolev and A. Yao. 1983. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory* 29, 2 (March 1983), 198–208. https://doi.org/10.1109/TIT.1983.1056650

[62] Simon Eberz, Kasper B. Rasmussen, Vincent Lenders, and Ivan Martinovic. 2016. Looks Like Eve: Exposing Insider Threats Using Eye Movement Biometrics. *ACM Trans. Priv. Secur.* 19, 1, Article 1 (June 2016), 31 pages. https://doi.org/10.1145/2904018

[63] Megan L. Evans, Margo Lindauer, and Maureen E. Farrell. 2020. A Pandemic within a Pandemic — Intimate Partner Violence during Covid-19. *New England Journal of Medicine* 383, 24 (2020), 2302–2304. https://doi.org/10.1056/NEJMp2024046 arXiv:https://doi.org/10.1056/NEJMp2024046

[64] Roy T. Fielding, Mark Nottingham, and Julian Reschke. 2022. HTTP Semantics. RFC 9110. https://doi.org/10.17487/RFC9110

[65] Rick Fillion. 2017. *Introducing Travel Mode: Protect your data when crossing borders.* 1Password. https://blog.1password.com/introducing-travel-mode-protect-your-data-when-crossing-borders/

[66] Keith Fisher. 2020. *Update on Border Searches of Electronic Devices.* American Bar Association. https://www.americanbar.org/groups/business_law/publications/blt/2020/04/border-searches/

[67] Gina Fisk, Mike Fisk, Christos Papadopoulos, and Joshua Neil. 2003. Eliminating Steganography in Internet Traffic with Active Wardens. In *Information Hiding*, Fabien A. P. Petitcolas (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 18–35.

[68] Google for Developers. 2014. *Android Studio.* Google Developers. Retrieved March 16, 2023 from https://developer.android.com/studio

[69] Internet Engineering Task Force. 2018. *Transport Layer Security.* Internet Engineering Task Force. Retrieved 5 October 2023 from https://datatracker.ietf.org/group/tls/about/

[70] P. Francis, S. Jamin, Cheng Jin, Yixin Jin, D. Raz, Y. Shavitt, and L. Zhang. 2001. IDMaps: a global Internet host distance estimation service. *IEEE/ACM Transactions on Networking* 9, 5 (2001), 525–540. https://doi.org/10.1109/90.958323

[71] Matt Franklin. 2006. A Survey of Key Evolving Cryptosystems. *International Journal of Security and Networks* 1, 1/2 (2006), 46. https://doi.org/10.1504/IJSN.2006.010822

[72] Diana Freed, Jackeline Palmer, Diana Minchala, Karen Levy, Thomas Ristenpart, and Nicola Dell. 2018. "A Stalker's Paradise": How Intimate

Partner Abusers Exploit Technology. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3174241

[73] Nicolas Gailly, Kelsey Melissaris, and Yolan Romailler. 2023. tlock: Practical Timelock Encryption from Threshold BLS. Cryptology ePrint Archive, Paper 2023/189. https://eprint.iacr.org/2023/189 https://eprint.iacr.org/2023/189.

[74] Wade Gasior and Li Yang. 2012. Exploring Covert Channel in Android Platform. In *2012 International Conference on Cyber Security*. IEEE, 173–177.

[75] Lorena Gazzotti. 2021. (Un)making illegality: Border control, racialized bodies and differential regimes of illegality in Morocco. *The Sociological Review* 69, 2 (2021), 277–295. https://doi.org/10.1177/0038026120982273 arXiv:https://doi.org/10.1177/0038026120982273

[76] Ilana Gershon. 2020. The Breakup 2.1: The ten-year update. *The Information Society* 36, 5 (2020), 279–289. https://doi.org/10.1080/01972243.2020.1798316 arXiv:https://doi.org/10.1080/01972243.2020.1798316

[77] Eu-Jin Goh, Dan Boneh, Benny Pinkas, and Philippe Golle. 2003. The Design and Implementation of Protocol-Based Hidden Key Recovery. In *Information Security*, Colin Boyd and Wenbo Mao (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 165–179.

[78] Shafi Goldwasser and Silvio Micali. 1984. Probabilistic encryption. *J. Comput. System Sci.* 28, 2 (1984), 270–299. https://doi.org/10.1016/0022-0000(84)90070-9

[79] Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Mihir Bellare, and Sara K. Miner. 1999. A Forward-Secure Digital Signature Scheme. In *Advances*

in Cryptology – CRYPTO '99, Michael Wiener (Ed.). Vol. 1666. Springer Berlin Heidelberg, Berlin, Heidelberg, 431–448. https://doi.org/10.1007/3-540-48405-1_28

[80] Payas Gupta and Debin Gao. 2010. Fighting Coercion Attacks in Key Generation Using Skin Conductance. In *Proceedings of the 19th USENIX Conference on Security* (Washington, DC) *(USENIX Security'10)*. USENIX Association, USA, 30.

[81] Gerhard P. Hancke and Markus G. Kuhn. 2005. An RFID Distance Bounding Protocol. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*. IEEE, 67–73. https://doi.org/10.1109/SECURECOMM.2005.56

[82] Drew Harwell. 2022. *Customs officials have copied Americans' phone data at massive scale*. The Washington Post. https://www.washingtonpost.com/technology/2022/09/15/government-surveillance-database-dhs/

[83] Johan Høastad, Jakob Jonsson, Ari Juels, and Moti Yung. 2000. Funkspiel Schemes: An Alternative to Conventional Tamper Resistance. In *Proceedings of the 7th ACM Conference on Computer and Communications Security - CCS '00*. ACM Press, Athens, Greece, 125–133. https://doi.org/10.1145/352600.352619

[84] Nicholas J. Hopper, John Langford, and Luis von Ahn. 2002. Provably Secure Steganography. In *Advances in Cryptology — CRYPTO 2002*, Moti Yung (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 77–92.

[85] Safe Horizon. 2021. SafeChat.

[86] William Huba, Bo Yuan, Daryl Johnson, and Peter Lutz. 2011. A HTTP Cookie Covert Channel. In *Proceedings of the 4th International Conference on Security of Information and Networks (SIN '11)*. Association for Computing

Machinery, New York, NY, USA, 133–136. https://doi.org/10.1145/2070425.2070447

[87] Ben Hubbard and Richard C. Paddock. 2019. *Saudi Women, Tired of Restraints, Find Ways to Flee*. The New York Times. https://www.nytimes.com/2019/01/11/world/middleeast/saudi-arabia-women-flee.html

[88] Troy Hunt. [n. d.]. . Retrieved 4 October 2023 from https://haveibeenpwned.com

[89] Saira Hussain and Sophia Cope. 2019. *Harvard Student's Deportation Raises Concerns about Border Device Searches and Social Media Surveillance*. Electronic Frontier Foundation. Retrieved 22 September 2023 from https://www.eff.org/deeplinks/2019/08/harvard-students-deportation-raises-concerns-about-border-device-searches-a

[90] Hayyu Imanda and Kasper Rasmussen. 2023. Nakula: Coercion Resistant Data Storage against Time-Limited Adversary. In *Proceedings of the 18th International Conference on Availability, Reliability and Security* (Benevento, Italy) *(ARES '23)*. Association for Computing Machinery, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.1145/3600160.3600175

[91] Apple Inc. 2020. *Device and Data Access when Personal Safety is At Risk*. Technical Report. Apple Inc. https://manuals.info.apple.com/MANUALS/1000/MA1976/en_US/device-and-data-access-when-personal-safety-is-at-risk.pdf

[92] Internet Live Stats. 2021. Google Search Statistics.

[93] Charlie Jacomme and Steve Kremer. 2018. An Extensive Formal Analysis of Multi-factor Authentication Protocols. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 1–15. https://doi.org/10.1109/CSF.2018.00008

[94] Rhett Jones. 2017. *Border Agent Demands NASA Scientist Unlock Phone Before Entering the Country.* Gizmodo. https://gizmodo.com/border-agent-demands-nasa-scientist-unlock-phone-before-1792275942

[95] David Kahn. 1974. *The codebreakers* ([abridged] ed. ed.). Weidenfeld and Nicolson, London.

[96] Paul Karp. 2018. *Coalition's surveillance laws give police power to access electronic devices.* The Guardian. https://www.theguardian.com/australia-news/2018/aug/14/coalitions-surveillance-laws-give-police-power-to-access-electronic-devices

[97] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography, Second Edition* (second ed.). Chapman & Hall/CRC, Boca Raton, FL.

[98] Joseph 'Jofish' Kaye. 2011. Self-Reported Password Sharing Strategies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 2619–2622. https://doi.org/10.1145/1978942.1979324

[99] Auguste Kerckhoffs. 1883. La cryptographie militaire. *Journal des Sciences Militaires* IX (1883), 5–38.

[100] Sheharbano Khattak, Tariq Elahi, Laurent Simon, Colleen M. Swanson, Steven J. Murdoch, and Ian Goldberg. 2016. SoK: Making Sense of Censorship Resistance Systems. *Proceedings on Privacy Enhancing Technologies* 2016, 4 (Oct. 2016), 37–61. https://doi.org/10.1515/popets-2016-0028

[101] Marek Klonowski, Przemysław Kubiak, and Mirosław Kutyłowski. 2008. Practical Deniable Encryption. In *SOFSEM 2008: Theory and Practice of Computer Science*, Viliam Geffert, Juhani Karhumäki, Alberto Bertoni, Bart

Preneel, Pavol Návrat, and Mária Bieliková (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 599–609.

[102] Konstantin G. Kogos, Elena I. Seliverstova, and Anna V. Epishkina. 2017. Review of Covert Channels over HTTP: Communication and Countermeasures. In *2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. 459–462. https://doi.org/10.1109/EIConRus.2017.7910590

[103] Katharina Kohls, Thorsten Holz, Dorothea Kolossa, and Christina Pöpper. 2016. SkypeLine: Robust Hidden Data Transmission for VoIP. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCS '16)*. Association for Computing Machinery, New York, NY, USA, 877–888. https://doi.org/10.1145/2897845.2897913

[104] Karen Levy and Bruce Schneier. 2020. Privacy threats in intimate relationships. *Journal of Cybersecurity* 6, 1 (2020), 1–31. https://doi.org/10.1093/cybsec/tyaa006 arXiv:https://academic.oup.com/cybersecurity/article-pdf/6/1/tyaa006/33328242/tyaa006.pdf

[105] Changjiang Li, Li Wang, Shouling Ji, Xuhong Zhang, Zhaohan Xi, Shanqing Guo, and Ting Wang. 2022. Seeing is Living? Rethinking the Security of Facial Liveness Verification in the Deepfake Era. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 2673–2690.

[106] Norka B. Lucena, Grzegorz Lewandowski, and Steve J. Chapin. 2006. Covert Channels in IPv6. In *Privacy Enhancing Technologies*, George Danezis and David Martin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 147–166.

[107] Philip MacKenzie and Michael K. Reiter. 2003. Networked cryptographic devices resilient to capture. *International Journal of Information Security* 2, 1 (Nov. 2003), 1–20. https://doi.org/10.1007/s10207-003-0022-8

[108] Elisabeth Mahase. 2020. Covid-19: EU States Report 60% Rise in Emergency Calls about Domestic Violence. *BMJ (Clinical research ed.)* 369 (2020), 1. https://doi.org/10.1136/bmj.m1872 arXiv:https://www.bmj.com/content/369/bmj.m1872.full.pdf

[109] Diogo Marques, Tiago Guerreiro, Luis Carriço, Ivan Beschastnikh, and Konstantin Beznosov. 2019. Vulnerability & Blame: Making Sense of Unauthorized Access to Smartphones. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300819

[110] Morgan Marquis-Boire. 2015. *How Governments Are Using Spyware to Attack Free Speech.*

[111] Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, and Carlo Alberto Boano. 2017. Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In *Proceedings of the 2017 Network and Distributed Systems Security*. Internet Society, San Diego, California.

[112] Andrew D. McDonald and Markus G. Kuhn. 2000. StegFS: A Steganographic File System for Linux. In *Information Hiding*, Andreas Pfitzmann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 463–477.

[113] Alfred J. Menezes and Scott A. Vanstone. 1993. Elliptic Curve Cryptosystems and Their Implementation. *J. Cryptol.* 6, 4 (sep 1993), 209–224. https://doi.org/10.1007/BF00203817

[114] UK SAYS NO MORE. [n. d.]. *Ask for Ani.* UK SAYS NO MORE. Retrieved 19 September 2023 from https://uksaysnomore.org/get-involved/ask-for-ani/

[115] T.S.E. Ng and Hui Zhang. 2002. Predicting Internet network distance with coordinates-based approaches. In *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 1. IEEE, 170–179. https://doi.org/10.1109/INFCOM.2002.1019258

[116] Alexandra Nisenoff, Maximilian Golla, Miranda Wei, Juliette Hainline, Hayley Szymanek, Annika Braun, Annika Hildebrandt, Blair Christensen, David Langenberg, and Blase Ur. 2023. A Two-Decade Retrospective Analysis of a University's Vulnerability to Attacks Exploiting Reused Passwords. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 5127–5144.

[117] United Nations Office of Counter-Terrorism. [n. d.]. *On Human Rights and Screening in Border Security and Management.* Technical Report. https://www.un.org/sites/www.un.org.counterterrorism/files/1806953-en-ctitf-handbookhrscreeningatborders-for-web2.pdf

[118] National Institute of Standards and Technology. 2012. *INFORMATION SECURITY.* Technical Report Special Publication 800-30.

[119] Office for National Statistics. 2020. *Domestic Abuse during the Coronavirus (COVID-19) Pandemic, England and Wales: November 2020.* Retrieved 16 April 2024 from https://www.ons.gov.uk/peoplepopulationandcommunity/crimeandjustice/articles/domesticabuseduringthecoronaviruscovid19pandemicenglandandwales/november2020

[120] Joint Committee on Human Rights. 2019. *Immigration detention: Sixteenth Report of Session 2017-–19.* Technical Report.

[121] Hilary Osborne and Sam Cutler. 2019. *Chinese Border Guards Put Secret Surveillance App on Tourists' Phones.* https://www.theguardian.com/world/2019/jul/02/chinese-border-guards-surveillance-app-tourists-phones

[122] Pallets. 2010. *Flask: web development, one drop at a time*. Pallets. Retrieved March 16, 2023 from `https://flask.palletsprojects.com/en/2.2.x/`

[123] Olivier Pereira, Florentin Rochet, and Cyrille Wiedling. 2018. Formal Analysis of the FIDO 1.x Protocol. In *Foundations and Practice of Security*, Abdessamad Imine, José M. Fernandez, Jean-Yves Marion, Luigi Logrippo, and Joaquin Garcia-Alfaro (Eds.). Springer International Publishing, Cham, 68–82.

[124] Oxford University Press. 2023. *Oxford English Dictionary*. Oxford University Press. Retrieved 13 September 2023 from `https://www.oed.com/`

[125] Widya Puspitasari and Fauziah Mayangsari. 2022. *COVID-19 and Gender-Based Violence in Indonesia: The Urgency of Prevention and Mitigation Framework*. BRIN Publishing, Chapter 6, 97–122. `https://doi.org/10.55981/brin.536.c463`

[126] Jie Qin. 2015. Hero on Twitter, Traitor on News: How Social Media and Legacy News Frame Snowden. *The International Journal of Press/Politics* 20, 2 (2015), 166–184. `https://doi.org/10.1177/1940161214566709` arXiv:https://doi.org/10.1177/1940161214566709

[127] Kasper Rasmussen and Paolo Gasti. 2018. Weak and Strong Deniable Authenticated Encryption: On Their Relationship and Applications. In *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE Computer Society, Belfast, 1–10. `https://doi.org/10.1109/PST.2018.8514181`

[128] Joel Reardon, David Basin, and Srdjan Capkun. 2013. SoK: Secure Data Deletion. In *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society, San Francisco, California, USA, 301–315. `https://doi.org/10.1109/SP.2013.28`

[129] Joel Reardon, Hubert Ritzdorf, David Basin, and Srdjan Capkun. 2013. Secure Data Deletion from Persistent Media. In *Proceedings of the 2013*

ACM SIGSAC Conference on Computer & Communications Security (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 271–284. https://doi.org/10.1145/2508859.2516699

[130] Nathan Reitinger, Nathan Malkin, Omer Akgul, Michelle L. Mazurek, and Ian Miers. 2023. Is Cryptographic Deniability Sufficient? Non-Expert Perceptions of Deniability in Secure Messaging. In *2023 IEEE Symposium on Security and Privacy (SP) (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 274–292. https://doi.org/10.1109/SP46215.2023.00095

[131] Ronald Linn Rivest, Adi Shamir, and D. A. Wagner. 1996. *Time-Lock Puzzles and Timed-Release Crypto*. Technical Report. USA.

[132] Michael Roe. 2010. *Cryptography and evidence*. Technical Report. University of Cambridge.

[133] Phillip Rogaway. 2016. The Moral Character of Cryptographic Work. USENIX Association, Austin, TX.

[134] Bill Roscoe. 2014. Computer Security: 16 Lectures, Michaelmas Term 2014.

[135] Leah Namisa Rosenbloom and Seny Kamara. 2023. Crypgraphy for Grassroots Organization. In *Real World Cryptography Symposium*.

[136] SafeLives. 2015. Getting it right first time. https://safelives.org.uk/sites/default/files/resources/Getting%20it%20right%20first%20time%20-%20complete%20report.pdf

[137] SafeLives. 2017. Insights Idva England and Wales dataset 2016-17: Adult independent domestic violence advisor (Idva) services. https://safelives.org.uk/sites/default/files/resources/Insights%20national%20dataset%20-%20Idva%202016-17%20-%20Final.pdf

[138] M. Satheesh Kumar, Jalel Ben-Othman, and K.G. Srinivasagan. 2018. An Investigation on Wannacry Ransomware and its Detection. In *2018 IEEE Symposium on Computers and Communications (ISCC)* (25-28 June 2018). IEEE Computer Society, Natal, Brazil, 1–6. https://doi.org/10.1109/ISCC.2018.8538354

[139] Bruce Schneier. 1998. *The Secret Story of Nonsecret Encryption.* https://www.schneier.com/essays/archives/1998/04/the_secret_story_of.html

[140] Bruce Schneier. 2009. *Protect Your Laptop Data From Everyone, Even Yourself.* Wired. https://www.wired.com/2009/07/protect-your-laptop-data-from-everyone-even-yourself/

[141] Bruce Schneier and Phil Sutherland. 1995. *Applied Cryptography: Protocols, Algorithms, and Source Code in C* (2nd ed.). John Wiley & Sons, Inc., USA.

[142] Oxford Secure. [n. d.]. *Stay safe on the move.* University of Oxford. Retrieved 19 September 2023 from https://www.infosec.ox.ac.uk/stay-safe-move

[143] Semrush. 2022. Top websites. https://www.semrush.com/website/top/

[144] Congressional Research Service. 2021. *Searches and Seizures at the Border and the Fourth Amendment.* Technical Report R46601. https://sgp.fas.org/crs/homesec/R46601.pdf

[145] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (nov 1979), 612–613. https://doi.org/10.1145/359168.359176

[146] Hung-Jr Shiu, Bor-Shing Lin, Bor-Shyh Lin, Po-Yang Huang, Chien-Hung Huang, and Chin-Laung Lei. 2018. Data Hiding on Social Media Communications Using Text Steganography. In *Risks and Security of Internet and Systems*, Nora Cuppens, Frédéric Cuppens, Jean-Louis Lanet, Axel Legay, and Joaquin Garcia-Alfaro (Eds.). Springer International Publishing, Cham, 217–224.

[147] Joseph H. Silverman. 2009. *The Arithmetic of Elliptic Curves, Second Edition.* Springer, New York, NY. https://doi.org/10.1007/978-0-387-09494-6

[148] Gustavus J. Simmons. 1984. The Prisoners' Problem and the Subliminal Channel. In *Advances in Cryptology: Proceedings of Crypto 83*, David Chaum (Ed.). Springer US, Boston, MA, 51–67. https://doi.org/10.1007/978-1-4684-4730-9_5

[149] Gustavus J. Simmons. 1985. The Subliminal Channel and Digital Signatures. In *Advances in Cryptology (Lecture Notes in Computer Science)*, Thomas Beth, Norbert Cot, and Ingemar Ingemarsson (Eds.). Springer, Berlin, Heidelberg, 364–378. https://doi.org/10.1007/3-540-39757-4_25

[150] Dave Singelee and Bart Preneel. 2005. Location verification using secure distance bounding protocols. In *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005.* IEEE, Washington, DC, USA, 7 pp.–840. https://doi.org/10.1109/MAHSS.2005.1542879

[151] Supriya Singh, Anuja Cabraal, Catherine Demosthenous, Gunela Astbrink, and Michele Furlong. 2007. Password Sharing: Implications for Security Design Based on Social Practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '07)*. Association for Computing Machinery, New York, NY, USA, 895–904. https://doi.org/10.1145/1240624.1240759

[152] Julia Słupska. 2019. Safe at Home: Towards a Feminist Critique of Cybersecurity. *St Antony's International Review* 15, 1 (2019), 83–100. https://www.jstor.org/stable/27027755

[153] Julia Słupska and Angelika Strohmayer. 2022. Networks of Care: Tech Abuse Advocates' Digital Security Practices. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 341–358. https://www.usenix.org/conference/usenixsecurity22/presentation/slupska-networks

[154] Sampath Srinivas, Dirk Balfanz, Eric Tiffany, and Alexei Czeskis. 2017. *Universal 2nd Factor (U2F) Overview.* FIDO Alliance. https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.html

[155] J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin, and D. Dean. 2002. Self-Healing Key Distribution with Revocation. In *Proceedings 2002 IEEE Symposium on Security and Privacy.* IEEE, San Francisco, CA, 241–257. https://doi.org/10.1109/SECPRI.2002.1004375

[156] Emil Stefanov and Mikhail Atallah. 2010. Duress Detection for Authentication Attacks against Multiple Administrators. In *Proceedings of the 2010 ACM Workshop on Insider Threats (Insider Threats '10).* Association for Computing Machinery, New York, NY, USA, 37–46. https://doi.org/10.1145/1866886.1866895

[157] Gianluca Stringhini. 2021. *The Cyber Security Body of Knowledge v1.1.0, 2021.* University of Bristol, Chapter Adversarial Behaviours.

[158] Kian-Lee Tan, Hwee Hwa Pang, and Xuan Zhou. 2004. Hiding Data Accesses in Steganographic File System. In *Proceedings. 20th International Conference on Data Engineering.* IEEE Computer Society, Los Alamitos, CA, USA, 572. https://doi.org/10.1109/ICDE.2004.1320028

[159] The Met Police. 2016. *Ask for Angela.* Retrieved 16 April 2024 from https://www.met.police.uk/AskforAngela

[160] Elise Thomas. 2018. *Sydney airport seizure of phone and laptop 'alarming', say privacy groups.* The Guardian. https://www.theguardian.com/world/2018/aug/25/sydney-airport-seizure-of-phone-and-laptop-alarming-say-privacy-groups

[161] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. 2011. On the Requirements for Successful GPS Spoofing

Attacks. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '11)*. Association for Computing Machinery, New York, NY, USA, 75–86. https://doi.org/10.1145/2046707.2046719

[162] Amar Toor. 2013. *UK border police can seize and download your phone's data for no reason at all.* The Verge. https://www.theverge.com/2013/7/15/4524208/uk-border-police-seize-download-mobile-phone-data-under-anti-terror-law

[163] Ari Trachtenberg. 2014. Say It Ain't so - an Implementation of Deniable Encryption. In *Blackhat Asia*.

[164] Truecrypt. 2023. *Truecrypt*. TrueCrypt Foundation. https://truecrypt.sourceforge.net

[165] Michael Carl Tschantz, Sadia Afroz, Anonymous, and Vern Paxson. 2016. SoK: Towards Grounding Censorship Circumvention in Empiricism. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, 914–933. https://doi.org/10.1109/SP.2016.59

[166] Nirvan Tyagi, Muhammad Haris Mughees, Thomas Ristenpart, and Ian Miers. 2018. BurnBox: Self-Revocable Encryption in a World Of Compelled Access. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 445–461. https://www.usenix.org/conference/usenixsecurity18/presentation/tyagi

[167] VeraCrypt. 2022. *Veracrypt*. IDRIX. https://www.veracrypt.fr/en/Hidden%20Volume.html

[168] Luis Von Ahn and Nicholas J Hopper. 2004. Public-Key Steganography. In *International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT 2004*, Christian Cachin and Jan L. Camenisch (Eds.). Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 323–341.

[169] Wikipedia. [n. d.]. *Adversary (cryptography)*. Wikipedia: The Free Encyclopedia. Retrieved 16 April 2024 from https://en.wikipedia.org/wiki/Adversary_(cryptography)

[170] Wikipedia. 2023. *Rubber-hose Cryptanalysis*. Wikipedia: The Free Encylopedia. Retrieved 14 September 2023 from https://en.wikipedia.org/wiki/Rubber-hose_cryptanalysis

[171] Ben Wolford. 2018. *How to protect your phone or computer when crossing borders*. Proton. https://proton.me/blog/border-crossing-protect-electronics

[172] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. 2011. Telex: Anticensorship in the Network Infrastructure. In *Proceedings of the 20th USENIX Conference on Security* (San Francisco, CA) *(SEC'11)*. USENIX Association, USA, 30.

[173] Adam Young and Moti Yung. 1996. Cryptovirology: Extortion-Based Security Threats and Countermeasures. In *Proceedings of the 1996 IEEE Conference on Security and Privacy* (Oakland, California) *(SP'96)*. IEEE Computer Society, USA, 129–140.

[174] Jiangshan Yu, Mark Ryan, and Cas Cremers. 2018. DECIM: Detecting Endpoint Compromise In Messaging. *IEEE Transactions on Information Forensics and Security* 13, 1 (Jan. 2018), 106–118. https://doi.org/10.1109/TIFS.2017.2738609

[175] Xingjie Yu, Zhan Wang, Kun Sun, Wen Tao Zhu, Neng Gao, and Jiwu Jing. 2014. Remotely wiping sensitive data on stolen smartphones. In *Proceedings of the 9th ACM symposium on Information, computer and communications security (ASIA CCS '14)*. Association for Computing Machinery, New York, NY, USA, 537–542. https://doi.org/10.1145/2590296.2590318

[176] Kexiong Curtis Zeng, Yuanchao Shu, Shinan Liu, Yanzhi Dou, and Yaling Yang. 2017. A Practical GPS Location Spoofing Attack in Road Navigation Scenario. In *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications* (Sonoma, CA, USA) *(HotMobile '17)*. Association for Computing Machinery, New York, NY, USA, 85–90. https://doi.org/10.1145/3032970.3032983

[177] Qionglu Zhang, Shijie Jia, Bing Chang, and Bo Chen. 2018. Ensuring Data Confidentiality via Plausibly Deniable Encryption and Secure Deletion –a Survey. *Cybersecurity* 1, 1 (2018), 1. https://doi.org/10.1186/s42400-018-0005-8

[178] Lianying Zhao. 2018. *Authentication and Data Protection under Strong Adversarial Model*. Ph. D. Dissertation. Concordia University.

[179] Lianying Zhao and Mohammad Mannan. 2015. Gracewipe: Secure and Verifiable Deletion under Coercion. In *Proceedings of the 2015 Network and Distributed System Security*. Internet Society, San Diego, California, USA, 16 pages.

[180] Zhichao Zhu and Guohong Cao. 2011. APPLAUS: A Privacy-Preserving Location Proof Updating System for location-based services. In *2011 Proceedings IEEE INFOCOM*. IEEE, 1889–1897. https://doi.org/10.1109/INFCOM.2011.5934991