Cryptographic Protocols for Privacy Enhancing Technologies

From Privacy Preserving Human Attestation to Internet Voting

by

Iñigo Querejeta-Azurmendi

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in

Computer Science and Technology

Universidad Carlos III de Madrid

Advisors:

Luis Hernández Encinas
Instituto de Tecnologías Físicas y de la Información (ITEFI)
Consejo Superior de Investigaciones Científicas (CSIC)
Jorge López Hernández-Ardieta
Universidad Carlos III de Madrid (UC3M)

Tutor:

Jorge López Hernández-Ardieta
Universidad Carlos III de Madrid (UC3M)

June, 2022

ACKNOWLEDGMENTS

These are probably the lines of the thesis that bring me more excitement. They represent the ending of a long way being left behind. I am very grateful of the people I have met and worked with throughout this journey.

First of all, I would like to thank Jorge López Hernández Ardieta and Luis Hernández Encinas, my advisors, for giving me this opportunity. Your support has been key for the success of this project, and your time and dedication to this thesis has turned it into a result that I am proud to present. Your trust on my capabilities, sometimes higher than what I had of myself, pushed me to believe in the worthiness of my research.

I would also like to thank Wouter Lueks and Carmela Troncoso. Your view of how research should be conducted, and how the world of academia should be defined made you a role for who I wanted to become as a researcher. I am very greatful to have had the chance to work with you.

Jorge Linde and David Nevado, I have learned more from you than what I can recall. Those breakfast sessions have been very educational. I am very happy to know that, even if digitally, we still manage to find time for those important moments.

Antonio Nappa, if today I'm part of the Web3 movement it is in big part thanks to you. You opened me the door to this great world by giving me the opportunity of being your PhD intern at Brave.

I would like to thank Minsait and Brave for hosting me as a researcher and giving me the financial support to pursue my research. I would also like to thank Instituto de Tecnologías Físicas y de la Información (ITEFI) from the Consejo Superior de Investigaciones Científicas (CSIC) for hosting me as a PhD student throughout my thesis and financing my research efforts.

Ana, you have been my companion in the hardest moments along the way. Thanks for being there, and for managing to keep my motivation high till the end of the way.

And finally, to my family. From you I have received the crucial support throughout all stages of my life to reach this point and beyond. Part of what I am today is thanks to you.

PUBLISHED AND SUBMITTED CONTENT

We now present a list of the published content throughout the thesis. We divide this list into two main blocks. First we discuss the internet voting related content. Then we finish with the human attestation related content.

Internet Voting The design of the internet voting protocols, presented in Chapter 4 of this thesis, started as a research project at the Cybersecurity department of Minsait. In a team lead by Jorge López Hernández-Ardieta, I was assigned the task of designing, jointly with Jorge, an internet voting proposal with two particularly important properties: (i) designing a **usable** and **scalable** scheme, and (ii) presenting a coercion-resistant construction. The initial design of the protocol was conducted by Jorge and myself, for which we received support of Luis Hernández Encinas, David Arroyo Guardeño and Victor Gayoso Martínez to validate the work, modify it slightly, and perform an evaluation experiment. This resulted in the first publication of the thesis [QA+17]:

> Iñigo Querejeta-Azurmendi, Jorge L. Hernandez-Ardieta, Víctor Gayoso Martínez, Luis Hernandez Encinas, and David Arroyo. "A coercion-resistant and easy-to-use Internet e-voting protocol based on traceable anonymous certificates". In: *III Jornadas Nacionales de Investigación en Ciberseguridad*. Selected **Best Research Article**. May 2017

This work was then forked into two lines of improvement. Firstly, an effort guided by myself resulted in an improved version for which a short version was published at CISIS'19 [QA+19]:

> Iñigo Querejeta-Azurmendi, Luis Hernández Encinas, David Arroyo Guardeño, and Jorge L. Hernandez-Ardieta. "An Internet Voting Proposal Towards Improving Usability and Coercion Resistance." In: *CISIS-ICEUTE*. vol. 951. Advances in Intelligent Systems and Computing. Springer, 2019, pp. 155–164. DOI: 10.1007/978-3-030-20005-3_16

and its extended version in the journal Mathematics [QA+20]:

> Iñigo Querejeta-Azurmendi, David Arroyo Guardeño, Jorge L. Hernández-Ardieta, and Luis Hernández Encinas. "NetVote: A Strict-Coercion Resistance Re-Voting Based Internet Voting Scheme with Linear Filtering". In: *Mathematics* 8.9 (2020). DOI: 10.3390/math8091618

These two pieces of work were co-authored with Jorge, Luis and David. Formal analysis of the system was done by myself, and the co-authors helped in the conceptualization, investigation, methodology, and writingâ€“review and editing.

Secondly, the initial work was modified and improved in a joint work with Wouter Lueks and Carmela Troncoso. This work was published at USENIX Security'20 [LQAT20]:

> Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. "VOTEAGAIN: A Scalable Coercion-Resistant Voting System". In: *Proceedings of the 29th USENIX Conference on Security Symposium*. USA: USENIX Association, 2020

The design of the scheme was co-authored by Wouter, Carmela and myself. Formal analysis was led by Wouter, while I performed the proof of concept (PoC) implementation of the scheme. These two efforts have different trade-offs, making both solutions of interest depending on the setting.

Bot detection In this thesis, particularly in Chapter 3, we present one main contribution with respect to bot detection. We introduce a candidate replacement to the more intrusive Google ReCAPTCHA method currently used. This work was jointly brainstormed with all co-authors, namely Jiexin Zhang, Panagiotis Papadopoulos, Matteo Varvello, Antonio Nappa, Benjamin Livshits and myself. Initially the project was led by Jiexin which covered the machine learning (ML) related research. It was finally led by myself when the main contributions were in the field of cryptography. It is published in PoPETs'21 [Que+21]

> Iñigo Querejeta-Azurmendi, Panagiotis Papadopoulos, Matteo Varvello, Antonio Nappa, Jiexin Zhang, and Benjamin Livshits. "ZKSENSE: A Friction-less Privacy-Preserving Human Attestation Mechanism for Mobile Devices". In: *Proc. Priv. Enhancing Technol.* 2021.4 (2021), pp. 6–29. DOI: 10.2478/popets-2021-0058

All contributions in the field of Cryptography (the main topic of the paper) in this paper are of myself. The work on the system design and ML model are by Jiexin, Panagiotis and Antonio. The experimental analysis is set up by Matteo. Finally, the implementation of the ML model and Cryptographic primitives were performed by Jiexin and myself respectively.

OTHER RESEARCH MERITS

We list four pieces of work that were published during the time of this research which are not presented in this thesis. In the following two works we were looking for an untrusted way to transmit voting codes, which is a technique commonly used under the assumption of a trusted side channel (such as postal voting). The first paper was published at the PhD Colloquium of E-Voting-ID [Que17]:

> Iñigo Querejeta. "A different approach to code voting". In: *PhD Colloquium of Electronic Voting*. Ed. by Robert Krimmer, Melanie Volkamer, Nadja Braun Binder, Norbert Kersting, Olivier Pereira, and Carsten Schürmann. Springer International Publishing, 2017

The second paper was published as a short paper in JNIC'18 [QAHEHA18]:

> Iñigo Querejeta-Azurmendi, Luis Hernández Encinas, and Jorge L. Hernández Ardieta. "Don't shoot the messenger, How a trusted channel may not be a necessary assumption for remote code-voting". In: *IV Jornadas Nacionales de Investigación en Ciberseguridad*. 2018

None of these techniques resulted successful, and were therefore dismissed from the main body of the thesis.

Then, I was part of a project for designing a decentralised VPN. This work was co-authored with Matteo Varvello, Antonio Nappa, Panagiotis Papadopoulos, Gonçalo Pestana and Ben Livshits. It has been published in IFIP Networking '21 [Var+21]:

> Matteo Varvello, Iñigo Querejeta Azurmendi, Antonio Nappa, Panagiotis Papadopoulos, Gonçalo Pestana, and Benjamin Livshits. "VPN-Zero: A Privacy-Preserving Decentralized Virtual Private Network". In: *2021 IFIP Networking Conference (IFIP Networking)*. 2021, pp. 1–6. DOI: 10.23919/IFIPNetworking52078.2021.9472843

My involvement in this work was in building (and implementing) the cryptographic protocols that allowed for a verifiable hop in a distributed network.

As part of my internship at Brave, we built a Decentralized and Trustless Ad Platform with Reporting Integrity. This project, named THEMIS, was co-authored with Gonçalo Pestana, Panagiotis Papadopoulos and Benjamin Livshits [Pes+20]:

> Gonçalo Pestana, Iñigo Querejeta-Azurmendi, Panagiotis Papadopoulos, and Benjamin Livshits. *THEMIS: Decentralized and Trustless Ad Platform with Reporting Integrity*. 2020. arXiv: 2007.05556 [cs.CR]

The initial, and centralised, design was performed by myself. The decentralisation efforts where then led by Gonçalo, with whom we jointly performed the

conceptualization, investigation and implementation of the system. Panagiotis and Benjamin collaborated in the original writing and initial design of the work. This effort was then modified to be used with more performant cryptographic primitives. The latest update versions of THEMIS are described in two technical reports, *Brave Ads x THEMIS RFC&C*[1] and *Black Box Accumulators in the Context of THEMIS*[2]

Finally, I have been part of a research effort to improve the current definition of a Verifiable Random Function Standard. This work was co-authored with Christian Badertscher, Peter Gaži and Alexander Russell.

> Christian Badertscher, Peter Gaži, Iñigo Querejeta-Azurmendi, and Alexander Russell. *On UC-Secure Range Extension and Batch Verification for ECVRF*. Technical report. `https://iohk.io/en/research/library/papers/on-uc-secure-range-extension-and-batch-verification-for-ecvrf/`. 2021

[1] `https://github.com/brave-intl/themis-rfcc/blob/main/rfcc-themis-tech-report-v1.0.pdf`

[2] `https://github.com/brave-intl/themis-rfcc/blob/main/rfcc-themis-bbas-v1.0.pdf`

SUMMARY

Desire of privacy is oftentimes associated with the intention to hide certain aspects of our thoughts or actions due to some illicit activity. This is a narrow understanding of privacy, and a marginal fragment of the motivations for undertaking an action with a desired level of privacy. The right for not being subject to arbitrary interference of our privacy is part of the universal declaration of human rights (Article 12) and, above that, a requisite for our freedom. Developing as a person freely, which results in the development of society, requires actions to be done without a watchful eye. While the awareness of privacy in the context of modern technologies is not widely spread, it is clearly understood, as can be seen in the context of elections, that in order to make a free choice one needs to maintain its privacy. So why demand privacy when electing our government, but not when selecting our daily interests, books we read, sites we browse, or persons we encounter? It is popular belief that the data that we expose of ourselves would not be exploited if one is a law-abiding citizen. No further from the truth, as this data is used daily for commercial purposes: users' data has value. To make matters worse, data has also been used for political purposes without the user's consent or knowledge. However, the benefits that data can bring to individuals seem endless and a solution of not using this data at all seems extremist. Legislative efforts have tried, in the past years, to provide mechanisms for users to decide what is done with their data and define a framework where companies can use user data, but always under the consent of the latter. However, these attempts take time to take track, and have unfortunately not been very successful since their introduction.

In this thesis we explore the possibility of constructing cryptographic protocols to provide a technical, rather than legislative, solution to the privacy problem. In particular we focus on two aspects of society: browsing and internet voting. These two events shape our lives in one way or another, and require high levels of privacy to provide a safe environment for humans to act upon them freely. However, these two problems have opposite solutions. On the one hand, elections are a well established event in society that has been around for millennia, and privacy and accountability are well rooted requirements for such events. This might be the reason why its digitalisation is something which is falling behind with respect to other acts of our society (banking, shopping, reading, etc). On the other hand, browsing is a recently introduced action, but that has quickly taken track given the amount of possibilities that it opens with such ease. We now have access to whatever we can imagine (except for voting) at the distance of a click. However, the data that we generate while browsing is extremely sensitive, and most of it is dis-

closed to third parties under the claims of making the user experience better (targeted recommendations, ads or bot-detection).

Chapter 1 motivates why resolving such a problem is necessary for the progress of digital society. It then introduces the problem that this thesis aims to resolve, together with the methodology. In Chapter 2 we introduce some technical concepts used throughout the thesis. Similarly, we expose the state-of-the-art and its limitations.

In Chapter 3 we focus on a mechanism to provide private browsing. In particular, we focus on how we can provide a safer, and more private way, for human attestation. Determining whether a user is a human or a bot is important for the survival of an online world. However, the existing mechanisms are either invasive or pose a burden to the user. We present a solution that is based on a machine learning model to distinguish between humans and bots that uses natural events of normal browsing (such as touch the screen of a phone) to make its prediction. To ensure that no private data leaves the user's device, we evaluate such a model in the device rather than sending the data over the wire. To provide insurance that the expected model has been evaluated, the user's device generates a cryptographic proof. However this opens an important question. Can we achieve a high level of accuracy without resulting in a noneffective battery consumption? We provide a positive answer to this question in this work, and show that a privacy-preserving solution can be achieved while maintaining the accuracy high and the user's performance overhead low.

In Chapter 4 we focus on the problem of internet voting. Internet voting means voting remotely, and therefore in an uncontrolled environment. This means that anyone can be voting under the supervision of a coercer, which makes the main goal of the protocols presented to be that of coercion-resistance. We need to build a protocol that allows a voter to escape the act of coercion. We present two proposals with the main goal of providing a usable, and scalable coercion resistant protocol. They both have different trade-offs. On the one hand we provide a coercion resistance mechanism that results in linear filtering, but that provides a slightly weaker notion of coercion-resistance. Secondly, we present a mechanism with a slightly higher complexity (poly-logarithmic) but that instead provides a stronger notion of coercion resistance. Both solutions are based on a same idea: allowing the voter to cast several votes (such that only the last one is counted) in a way that cannot be determined by a coercer.

Finally, in Chapter 5, we conclude the thesis, and expose how our results push one step further the state-of-the-art. We concisely expose our contributions, and describe clearly what are the next steps to follow. The results presented in this work argue against the two main claims against privacy-

preserving solutions: either that privacy is not practical or that higher levels of privacy result in lower levels of security.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Digital progress has undoubtedly made our lives easier —this makes it far easier for us to overlook the harm that these technologies do in our daily lives. Even for experts in the field, it is hard to sacrifice the benefits of these privacy invasive technologies, and all too often we criticise the harm that Google, Facebook or Twitter can do with our data, but in turn use these apps for our daily routines. Instant gains make us forget long term loses. This is comparable, as presented in a study [VCA04], with the climate emergency. The majority of us understand the danger and harm it might cause, but the instant reward makes invisible the loses. Another similarity with the climate emergency is that this is not a problem that affects a small portion of society —this will produce systemic changes (if is not doing it already) and affect all of us. The power we give companies with big data of all the population allows them to take targeted decisions that can shape the communities ideas, political visions (polarising the community) or even belief of needs that these people have.

There has been several scandals that have positioned part of the community against data gathering corporations, as they have shown what is the power that data gives [Cad18; LH18]. Fake news are an ever increasing problem, and while that might not be directly related to privacy, the information companies have of our lives allow them to target their 'fake news' [Buc17] with best possible outcome - having as many people believe these. Political polarisation has been one of the negative results of targeted information, and if in one thing we might all find consensus is in saying that we want to avoid a polarised, frustrated community. Enforcing privacy might be an important

first step of removing that powerful weapon to data gathering corporations, but how can we achieve it without losing all the facilities that they have brought back?

The first responsibility relies in the experts of the field, and is common to every other problem: the communication. The big importance of how the problem reaches the consumers is what carves their understanding and the feel of emergency. But more importantly, the understanding of how to act against it will shape their implication. The communication from the expert community to the users is not something that should be taken for granted, and is the result of studies [Adj+13; CPJ18; FG20]. The alternatives presented to the community need to have an important goal —they need to be easy to understand. Moreover, not only it is important to understand what and how data is used, but also by whom. It is no longer a matter of Governments following suspects using their GPS signals. Things have evolved, and even third party entities (not related to the services one uses, or the sites one browses) have access to vast amount of data of users' daily actions.

1.1 Motivation

Recent years have brought several legislative efforts to take measures against these intrusive actions [Com18; Jus18], but they have been, in too many senses, absolute failures [Dro20; Hor19; Rya20]. Therefore, we believe that the solution needs to be technical, rather than legislative, as only the former can enforce actions that follow the pace of the technological advances.

Offering privacy-preserving alternatives with considerable lacks is not the path to follow, as the main blocker of such a transition is "I have nothing to hide, why should I deteriorate my services?". That is an ever increasing argument for not caring about privacy, and washing intrusive actions of governments or private corporations. The benefit, to the eyes of the user, can be several. Homeland security, better and free services (advertising), or more usability among others. However, these arguments have been used to surpass limits of surveillance, and even private companies have had to held up the pulse against the US government at the time of requiring to downgrade (or even break) security in their services. Some of these examples are Facebook and Apple [BL16; MIB19]. In the case of the former, the US Attorney General William P. Barr requested from Facebook a back door for the encryption mechanism used in their services. This request was rejected, with a good argument of relevance: "The 'backdoor' access you are demanding for law enforcement would be a gift to criminals, hackers and repressive

regimes". Indeed, a backdoor not only exists for the "good folks" —it can also be exploited with bad intentions. The US has had a long history of trying to limit the use of strong encryption —from the enforcement of weak ciphers, where security had to be deliberately weakened [Fre97], to the creation of a 'standardised backdoor' in an Elliptic Curve proposed by the NIST [BLN16]. As it is to expect, this has not only served the intended purpose (of claimed Homeland Security); it also opened the possibility to perform attacks on the 'claimed' security these ciphers gave [Adr+18; Val+15]. The US have not been the only government 'scared' of private communications, for example UK [Bal15], Europe [Pre20], or Russia [Ily] have considered, at some point of recent history, limiting or banning encryption. The effort has often times been directed in imposing, what we believe to be, a false dilemma; citizens need to choose between security and privacy (one cannot have both). This not only is false, but this misconception presents a considerable danger as the recent pandemic has proved. Tracking applications and systems have been recommended, and sometimes enforced, during the pandemic. The message was clear, these invasive technologies serve for tracking the spread of the virus, which results in the protection and security of all. Unfortunately, this scenario made clear the importance of privacy-preserving solutions. 85% percent of tracking apps leaked data [Hel20], and more worrying, some tracking apps which where used for health reasons are now being used for other purposes [Str21] to those initially claimed. What is more concerning, is that the use of these invasive technologies were enforced when there existed privacy-preserving (and decentralised) alternatives that offered almost the same guarantees [Gol20; Tro+20].

There is the general misconception from law abiding citizens that this data gathering is not targeted to them. They believe they get nothing but the benefits, and because they have nothing but 'normal lives', their information is not of interest. However, nothing further from the truth. Privacy invasions are not always aimed at their benefit, and yes, their profiles are of interest. As a matter of fact, their 'profile' is their main source of income. As a saying pictures it: "if you are not paying for the product, then you are the product". As presented by an article in The Economist, "the world's most valuable resource is no longer oil, but data"[Eco16]. Unfortunately, data is not only used with economical motivations. In the past years, data of the masses has been used to affect voter turnout, or even vote direction in both Trump's presidential election and the Brexit campaign [Cad18; LH18]. To quote Cummings, VoteLeave's campaign manager, "we couldn't have done it without [AggregateIQ]". Whether the use of this data was illegal or immoral falls out of the scope of this thesis. However, what seems clear, is that data is gold, both for online businesses as for success with electoral campaigns.

This brings yet another direction for privacy-preserving technologies: the social pressure of the political polarisation, public disclosure of one self, or targeted electoral campaign make it more important than ever to have safe elections, where not only privacy of vote is essential, but also verifiability that the election outcome is as it is expected. In an ever increasingly digitalised word, it is worth asking the question if an online voting system is possible.

One of the biggest challenge when designing privacy-preserving technologies is that we need to offer similar services to the non-private counterparts. After all, they do bring back to the community. Homeland security, human trafficking, or tax evasion are problems which are often solved with intrusive means, and we certainly want to build a society which is capable of handling these. Advertising is often seen as a burden [Mal19], but it is after all, the fuel of the free internet (except for important exceptions such as Wikipedia). Bot detection mechanisms are an important tool for the survival of our most desired commerces, and targeted recommendations are what makes a Google search more successful than some other engines. Moreover, allowing third party entities to access the furthest corners of our private online lives, has allowed them to provide reliable solutions. Google maps or Waze would not be as reliable if they did not know the location of their users, and current bot detection mechanisms would fail tremendously if they could not follow "legitimate users" in their browsing activities. Technologies have made our lives easier, allowing us to make bank transactions from home, purchases from our sofa, or keeping contact with our friend that moved to the other side of the world. Hence, 'if' our data is only used for things that benefit us (even if it generates income to the companies that exploit our data), then we are all good. Unfortunately, this does not hold true: data breaches have allowed adversaries to access millions of records of users around the world. The 20% of small and medium european companies have suffered data breaches in 2019 [Seg19], they are considered among the top 5 risks of global stability [MW19], and in the first 6 months of 2019, 4.1 billion records where exposed by data breaches [Win19]. This means that the data we believe to be used by private corporations is exploited by unwanted adversaries. Such is the concern over data breaches that companies are starting to adopt a new strategy —data minimisation [Mar16]. The General Data Protection Regulation (GDPR) has tried to enforce this [Com18], but it was a common practice way before the legislation came into place. The idea is simple, minimise the amount of data a company stores to the strictly necessary to provide their services. This, in turn, minimises the damage in case of a data breach.

1.2 Problem statement and objectives

In this thesis we try to push the boundaries further; rather than minimising the data stored to what is strictly needed to provide the service, we try to minimise the data *needed* to provide the service. The question is, can we achieve the benefits of existing solutions without the damage they expose us to? If the dilemma of 'security or privacy' is false, then why is there no widely deployed online solution for something as conceptually simple as elections?

We explore and construct privacy-preserving systems providing three properties simultaneously: (i) usability, (ii) sustainability and (iii) integrity. By *usability* we mean that the solution does not need to require an important effort by the user. Our solutions need to be aimed to the general population, and not exclusively the technical knowledgeable individuals. Not only that, but we need to require as little interaction as possible from the user. By *sustainability* of a system, we consider scalability, or user side computation (e.g. our solutions cannot drain user battery). A successful system needs to be designed in a way that can reach (or at least get close to) the dimensions of the replaced mechanism. Finally, with *integrity* we mean that the system needs to provide cryptographic guarantees of the claimed properties. The specific details of these three properties are specific to each of the presented schemes, and we give a more formal description thereof. In particular the two schemes we principally focus are the following; creating a private, and yet successful, bot detection mechanism, and an internet voting scheme.

While designing a privacy-preserving human attestation mechanism our goals were clear:

P1 Design a completely friction-less attestation via device micro-movements that happen during natural user actions like typing and screen touching.

P2 Avoid sending sensitive data to the server, by performing the entire attestation on the user's very own device enclosing it in zero-knowledge proofs to ensure integrity.

P3 Design a zero-knowledge mechanism that does not drain the user's device battery, while ensuring that the verification server can scale the number of attestations. This is, find a sweet spot between prover and verifier time.

P4 Design a mechanism which does not introduce any additional barrier to people affected by any form of disability.

Similarly, when building an internet voting scheme, our objectives were the following:

V1 Design a user-friendly voting procedure.

V2 Provide maximum security in terms of ballot privacy, verifiability of the correctness of the election, and coercion resistance.

V3 Build a scheme that can handle large-scale elections without requiring massive economical effort to prepare, or realize the election.

We further motivate these two topics in Section 1.2.1 and 1.2.2 respectively. These two topics are opposite, as the former exists due to invasive solutions (either from the perspective of usability as image CAPTCHA, or from the perspective of user privacy, such as reCAPTCHAv3), and the latter is barely used due to lack of private solutions providing usable, scalable and verifiable systems. We focus on these two topics as they entail two very important parts of our daily lives: on the one hand, human attestation (whether we realize it or not) happens constantly while browsing, and therefore presenting a privacy-preserving and usable solution is key. On the other hand, internet voting shapes our society, and therefore a solution that allows everyone to vote in total privacy and liberty, from wherever they are is what ensures an honest representation of the peoples choices. We question the dilemma of "Security OR Privacy", and try to provide both without harming usability. In the following subsections we present an overview of the two problems discussed above, and introduce the solutions presented in this thesis.

1.2.1 Privacy-preserving bot detection

Automated software agents that interact with content in a human-like way, have become more prevalent and pernicious in the recent years. Web scraping, competitive data mining, account hijacking, spam and ad fraud are attacks that such agents launch by mimicking human actions at large scale. According to a study [Hug19], 20.4% of the 2019 internet traffic was fraudulent, associated with *user, albeit not human, activity.*

In the ad market specifically, such type of fraudulent traffic costs companies between $6.5-$19 billions in the U.S. alone, and it is estimated that this will grow to $50 billion by 2025 [MK17]. When it comes to the ever-growing mobile traffic, a study [Thr18] (using data spanning 17 billion transactions) observes 189 million attacks originated specifically from mobile devices; this is an increase of 12% compared to the previous six months.

The "Completely Automated Public Turing tests to tell Computers and Humans Apart" (or just CAPTCHA) is the current state-of-the-art mechanism to assess the humanness of a user. CAPTCHAs are widely deployed across the internet to distinguish between human and non-human users. The major downsides of current CAPTCHA solutions (e.g. Securimage [Phi15], hCaptcha [Int19]) include:

(i) **questionable accuracy**: various past works demonstrate how CAPTCHAs can be solved within milliseconds [Boc+17; Iri18; Ove18; SPK16; YEA08],

(ii) **added friction**: additional user actions are required (e.g. image, audio, math, or textual challenges) that significantly impoverish the user experience [GN16], especially on mobile devices,

(iii) **discrimination**: poor implementations often block access to content [Dzi19], especially to visual-impaired users [Hol+19],

(iv) **serious privacy implications**: to reduce friction, the 3rd generation of Google's reCAPTCHA [Liu18] replaces proof-of-work challenges with extensive user tracking. Google's servers attest user's humanness by collecting and validating behavioral data [Fre19] (i.e. typing patterns, mouse clicks, stored cookies, installed plugins), thus raising significant privacy concerns [AFA19; Cla19; Sch19].

The goal of this thesis with respect to bot detection, and the content of Section 3, is to present a humanness attestation alternative that will answer negatively to the following question: are user experience or privacy required trade-offs for humanness attestation? By leveraging the device's motion sensors we aim at demonstrating that it is possible to build a humanness attestation mechanism on the edge that preserves the privacy of the users and runs seamlessly on the background thus adding zero friction to the mobile user experience. The key intuition behind our research is that whenever a (human) user interacts with a mobile device, the force applied during the touch event generates motion. This motion can be captured by the device's sensors (e.g. accelerometer and gyroscope) and used to uniquely distinguish real users from automated agents. As a matter of fact, this can even be used to provide continuous authentication of users [HA+21a; HA+21b]. Further, this detection can run on the user device, without requiring an impractical secure execution environment (unlike related proposals [Gue+18; JKP10]), but more importantly without sharing private information with any server (unlike state-of-the-art [Liu18]), apart from a *proof* that guarantees the integrity of the attestation result.

1.2.2 Internet voting

Democracy is one of the biggest achievements of our society with its main pillar being elections, and that is why any change in the electoral process needs a very detailed study. The digitalization of polls, while still going slower than any other field of society, is starting to become a developed trend, and even if some countries drawed back lately for fear of not having the ability to have high levels of auditability [Reu17; The17], the list of countries using electronic devices to assist in the ballot cast or tallying process keeps growing, with special focus in the developing world [HIT16]. This is known as electronic voting (e-voting), which comprises not only internet voting (i-voting) schemes, but also presence schemes that use electronic means for voting, tallying or verification purposes.

Traditional presence elections have the control in the environment where the voters cast a ballot, ensuring privacy of vote cast and allowing auditability of the vote cast method, which makes the process of digitalization of presence elections faster in comparison with the one of remote elections. The latter has several differences compared to the former. Firstly, the voter may cast the vote under any circumstances, resulting in a high possibility of coercion or vote-selling. Secondly, the vote cast will go through an uncontrolled channel. Thirdly, remote authentication increases the chances of impersonation, and finally, in the specific case of i-voting, the voter casts a vote from its own device, making it difficult to do a massive-scale, effective security assessment.

The next natural step for absentee voters is i-voting and whether it is because the number of expats grows, people living abroad want to get more politically involved, or simply because voters want to cast their vote from their homes, the number of remote voters is growing [Cha18; EC18]. However, many countries (e.g.: Germany, Great Britain, Spain, Mexico) are reluctant to use i-voting, and prefer, on the contrary, postal voting. A proposal of the former offering security, usability, low deployment requirements and low complexity has still not been presented, and this pushes countries to keep using postal voting as a remote voting system, even if properties such as privacy may be compromised. For example, in some countries [Gov20; Min; Min20], to cast an accepted postal vote, one has to send their sealed vote together with a certificate (which contains identification of the voter) validating their right to vote together in an envelope. This creates a direct relation between identity and vote choice, compromising like that vote privacy. Furthermore, a full chain-of-custody cannot be made, trust has to be given to external entities (postal office) and coercion can easily happen. Hence, i-voting to replace remote paper based elections would not only offer a reduction of tallying time, but mainly, could offer more security while improving usability

towards the voter. Current literature presents many solutions offering coercion resistance, universal and individual verifiability or everlasting privacy (these concepts will become clear in Section 2.1) with very low trust requirements. However these solutions are not usable, either by the complication towards the voter or by the complexity of the solution.

Remote authentication is another main topic in remote voting schemes, both by the difficulty of a proper solution and by its importance in a remote election. Digital authentication still remains a fragile topic, however, it is tenable that using cryptographic certificates is the most secure way of digital authentication. A cryptosystem of 128 bits security[3] is, as specified by the U.S. National Institute of Standards and Technology (NIST) or the SOG-IS crypto working group among others, considered to be secure for the distant future [ANS20; Bar16; BSI21; SOG20]. Some countries (such as Spain, Estonia, Luxembourg or Belgium), have a Public Key Infrastructure (PKI) for national identity cards. Such a key storage mechanism allows easy mobility of cryptographic keys to the user and requires only to memorize a password in order to use its cryptographic key, a key that (usually) never leaves the smartcard. Nevertheless, smartcards have disadvantages. While it may seem that it allows user mobility, it is still necessary to use it with a smartcard reader, limiting its usability. There is a high cost in the deployment of the card itself, which increases once we consider training the electorate or the Total Cost of Ownership (TCO). It is clear that the digitalisation of identification and authentication has to be included in governmental actions, however it is not a straightforward project. The cost of deploying the Spanish electronic identity card was expected to be of €220 million [Min06] and in 2014 (eight years after its introduction), it was used only in a 0,02% of the electronical procedures [GVUSS14]. In the case of Germany, apart from a delay in its deployment, it cost the government a total of €1.5 billion [CSP12] to deploy smartcards for eHealth. This shows that deploying a smartcard-based PKI in short notice cannot be a requirement for i-voting. Moreover, attacks to smartcard-based PKIs were shown [HA+13], with two well known cases; first with the known vulnerability of Taiwan's smartcards [Ber+13], followed with the vulnerability of a chip distributor [Nem+17], which affected, among others, Estonia, a country considered as the pioneer of Europe's digitalization.

The final goal of this thesis, with respect to internet voting, is to present a secure protocol that can be used both by developing countries, and more technologically mature countries. We believe that it is important not to base our protocol on the requirement of the ownership of public key pairs

[3]Note that bits of security does not necessarily mean the size of the keys, and for the cases of asymmetric cryptography, keys are always larger than 128 bits.

by the users. Not only it is expensive, but as we have seen, several weaknesses have been found. Hence, if a requirement to deploy an online voting system is to first deploy individual cryptographic keys per user, several governments/companies/institutions might oppose.

1.3 Methodology

This thesis' research was conducted by first finding an open problem, concerning users' privacy in the digital world, that could be solved by the means of cryptographic tools. The next step in resolving an existing problem was formalising the properties that we want to achieve. The properties are divided in two groups, either security properties or functional properties. By formalising the security properties, we understand exactly what we expect out of, for example, a *private* or *verifiabile* protocol. On the other hand, our functional requirements specified what was, for instance, our target traffic of users, or what level of interaction and computational complexity we could expect out of the user. Next, a design and preliminary security analysis of the design followed. It was common to iterate through this process until the preliminary (and informal) security analysis did not break the security of the protocol. Once a preliminary design was successful, an analysis of the functional properties followed. Again, if this study showed that the solution was unusable, we were back at square one.

Finally, when a design was preliminary functional and secure, a formal study followed (both for the security and functional properties). This consisted in trying to formally prove that the design indeed provides the expected formal properties. In this iterative process we found how, in certain cases, if we relaxed the initial conditions of our cryptographic schemes, important improvements could be achieved in terms of performance. In this scenario, which is clearly pictured in the chapter of Internet Voting, Chapter 4, we presented both of the schemes. To formally ensure that the functional requirements were met, an implementation was produced. This allowed us to study the scalability that the scheme could achieve, as well as the efficiency with respect to the user's device.

If, on the design of the solution, we introduced a new cryptographic primitive, a similar process followed. However, throughout this thesis we focused our efforts in building cryptographic protocols rather than primitives. Moreover, the cryptographic protocols introduced in this thesis considerably simplified the methodology for two reasons. First, the cryptographic protocols are transparent to the user, and therefore do not require a user study. Sec-

ondly, most of the properties expected out of these protocols are already well defined, facilitating the formalisation of the introduced protocols.

1.4 Structure of this thesis

In Chapter 2 we introduce the cryptographic protocols and tools, and the state-of-the-art. This will cover the basic cryptographic primitives that are generic to the work presented in the thesis, such as security notions, asymmetric encryption, or commitments schemes. Then, Chapters 3 and 4 are self-contained and can be read independently. They present the contributions of this thesis. In the former we describe a human attestation mechanism that preserves the privacy of the user. In the latter, we describe two internet voting solutions with different trade-offs. The first protocol provides a better performant filtering phase, while the second scheme provides a higher level of coercion resistance while maintaining scalability for large scale elections. Finally, in Chapter 5 we make the conclusion remarks, discuss the open problems and possible lines of future work.

CHAPTER 2

Background

In this chapter we present the related work and introduce the cryptographic notions used throughout this thesis. For introductory concepts on abstract algebra, we refer the reader to Lee [Lee18]. For a detailed and generic introduction to cryptographic protocols, we refer to Schoenmakers [Sch22]. We begin the chapter, in Section 2.1, by introducing the notation. Then, we proceed in Section 2.2, with a brief introduction to cryptographic primitives. Finally, we conclude the chapter in Section 2.3 with a study of the state-of-the-art for the two topics studied in this thesis: bot detection and internet voting.

2.1 Notation and Definitions

The field of cryptology owns its origins to the Greek language, *kryptós*, meaning "hidden, secret", and *logia*, meaning "study". This field is generally divided into two sub-fields, cryptography and cryptanalysis. The former concerns the study and design of secure processes and schemes related to information security. The latter studies ways to break them. We hereby present the distinction made by Schoenmakers, Definition 1.1 [Sch22], between a cryptographic algorithm, protocol and scheme. A **cryptographic algorithm** is a set of well-defined steps, which given an input value, process an output value. These algorithms achieve certain security objectives. A **cryptographic protocol**, on the other hand, is a distributed algorithm, where two or more parties interact following the defined steps. Similarly, a

protocol is expected to satisfy certain security objectives. Finally, a **cryptographic scheme** is a suite of related cryptographic algorithms and cryptographic protocols, achieving certain security objectives. In this thesis we focus in cryptographic protocols.

Cryptographic protocols require interaction between two or more parties, and therefore, the definition of these protocols require the definition of the types of channels that are available to these users. In a multiple party protocol (multi-party computation, or MPC), the channels are distinguished between *end-to-end* or *broadcast* channels. Another distinction of these channels is made between how the information is exchanged. A *public* channel assumes that observers of the protocol can read the exchanged data. On the other hand, a *private* channel assume that observers of the protocol cannot read the exchanged data. We put special attention to the notion of a *bulletin board*, which is an append only public broadcast channel, where parties of the protocol can post data such that everyone (even third parties) have access to it. A communication model is what defines the type of channels that are available to the parties involved in the protocol.

We use ϵ to denote the security parameter, and 1^ϵ to denote its unary representation. All algorithms of the cryptographic protocols presented in this thesis take as input 1^ϵ, and therefore, we omit it from the list of inputs. We use $\mathbb{Z}_p$ to denote the integers modulo p. Moreover, throughout the thesis we use p to denote a prime number. Let D be a set, we use $x \leftarrow_\$ D$ to denote that x is taken uniformly at random from D. The symbols $\top$ and $\bot$ denote success and failure respectively. We represent vector with bold letters, and use $\boldsymbol{x} \leftarrow_\$ D^n$ to denote that we take a vector of size n uniformly at random from D. To denote exponentiation of two vectors, $\boldsymbol{x}, \boldsymbol{y}$, we use $\boldsymbol{x}^{\boldsymbol{y}} = x_1^{y_1} \cdots x_n^{y_n}$.

When presenting a new cryptographic algorithm, protocol or scheme, one needs to prove that it achieves the expected security objectives. Some cryptographic algorithms have well defined properties to be satisfied (see Public Key Cryptosystems 2.2.1 or zero-knowledge proofs 2.2.3). However, proving security of cryptographic protocols may be a harder task, as one first needs to capture the properties of the protocol, and then prove it satisfies them. In this thesis, when defining a new cryptographic protocol, we use game based proofs, were we first define a game capturing the security properties, and then prove that no adversary (limited by our assumptions) can win the game (and hence, break the security). In general, this thesis assumes that adversaries run in probabilistic polynomial time (except for the property of everlasting privacy that we present in the protocol of Chapter 4).

Definition 2.1.1. *An adversary, $\mathcal{A}$, is said to run in* polynomial time *if*

there exists a polynomial $P(\cdot)$ such that $\mathcal{A}$ running on any input $x \in \{0,1\}^*$ halts within $P(|x|)$ steps.

This definition is what defines the complexity class **P**, which is the class of problems which can be solved by a deterministic polynomial time (DPT) Turing machine. Deterministic, in this context means that the algorithm, or Turing machine, at each step, moves from its current configuration to a *unique* successor. The problem of defining the hardness of a problem using the complexity class **P**, is that the problem is considered in terms of *worst-case* complexity. However, in cryptography it is much more relevant to consider the complexity of the problems in the average-case. To this end, we allow the adversary to access a source of randomness to define the next steps of its computation, and hence define the adversary to run in probabilistic polynomial time.

Definition 2.1.2. *An adversary, $\mathcal{A}$, is said to run in* probabilistic polynomial time *(PPT) if and adversary $\mathcal{A}'(\cdot) = \mathcal{A}(\cdot, r)$ runs in polynomial time where r is a string of uniformly random bits.*

Finally, when modelling security in cryptography, one cannot always aim to achieve perfect security. Sometimes there exists a (small) chance that the adversary guesses the secret correctly. When defining a protocol, one needs to take into consideration these events, and define what is the acceptable probability of such events happening without breaking the security of the protocol. We consider a protocol to be secure, if these events happen with *negligible probability.*

Definition 2.1.3. *A nonnegative function $f\colon \mathbb{N} \to \mathbb{R}$ is called negligible if for every $\gamma \in \mathbb{N}$ there exists a $k_0 \in \mathbb{N}$ such that $\forall k \geq k_0, f(k) \geq 1/k^\gamma$.*

In cryptography, to have practical groups and rings, we rely on hardness assumptions. These are usually of the type "a PPT adversary has very low probability in finding X given Y". In the coming chapter we introduce the hardness assumptions used throughout this thesis which rely mainly on the Discrete logarithm (DL) problem in cyclic groups[4]. As previously introduced, hardness of a problem depends on the fact that an adversary has no more than negligible probability of finding a solution. We now introduce the DL related hardness assumptions. We use a cyclic group $G = \langle g \rangle$ of order p.

Definition 2.1.4. *The* discrete logarithm *problem is defined by, given $g, X \in G$, output x such that $X = g^x$, namely the discrete logarithm of X in the base*

[4]For a more thorough list of cryptographic assumptions, see `https://www.ecrypt.eu.org/ecrypt2/documents/D.MAYA.6.pdf`

g. More technically, we say the DL-assumption *holds relative to G if for all PPT adversaries* $\mathcal{A}$*,*

$$\Pr[x \leftarrow \mathcal{A}((G, p, g), X) \wedge X = g^x]$$

is negligible with respect to ϵ*.*

Definition 2.1.5. *The* Diffie-Hellman *(DH) problem is defined by, given* $g^x, g^y \in G$ *for* $x, y \in \mathbb{Z}_p$*, output* $g^{x \cdot y}$*. We say that the* DH-assumption *holds relative to G, if for all PPT adversaries* $\mathcal{A}$*,*

$$\Pr[g^{x \cdot y} \leftarrow \mathcal{A}((G, p, g), X, Y) \wedge X = g^x \wedge Y = g^y]$$

is negligible with respect to ϵ*.*

Definition 2.1.6. *The* Decisional Diffie-Hellman *(DDH) problem is defined by, given* $g^x, g^y, Z \in G$ *for* $x, y \in \mathbb{Z}_p$*, determine whether or not* $Z = g^{x \cdot y}$*. We say that the* DDH-assumption *holds relative to G, if for all PPT adversaries* $\mathcal{A}$*,*

$$|\Pr[\mathcal{A}((G, p, g), g^x, g^y, g^z)] = 1| - |\Pr[\mathcal{A}((G, p, g), g^x, g^y, g^{x \cdot y})] = 1|$$

is negligible with respect to ϵ*, where* $x, y, z \leftarrow_{\$} \mathbb{Z}_p$*.*

Note that DDH assumption can only hold in a group where DH assumption holds, which in turn must be in a group where the DL assumption holds.

2.2 Cryptographic Algorithms and Protocols

In this section we introduce some basic cryptographic algorithms and protocols which are commonly used in the field, and in this work. We first introduce the notions with an analogy, and proceed with a more formal description. All the methods described hereof rely on the hardness assumptions described in Section 2.1. However, this is not, in any way, an extensive list of commonly used cryptographic algorithms and protocols, and only covers those used throughout the thesis.

2.2.1 Public Key Cryptosystems

The most commonly used analogy to describe public key cryptosystems is one of a padlock. Anyone that has access to the padlock may choose which locker to lock, but only the owner of the key is capable of opening the lock. Similarly, a public key (or asymmetric) cryptosystem consists of two keys; the public

key, pk, and the private (or secret) key, sk. As in the analogy, anyone that has access to the public key can encrypt (or "lock") a message, but only the owner of the secret key is capable of recovering the message. This allows two parties to communicate securely by exchanging their public keys and encrypting the corresponding messages. In this subsection we introduce the only encryption scheme used throughout the thesis, namely ElGamal cryptosystem. Finally, we briefly introduce Digital Signatures and Threshold Cryptography, which are important building blocks of public key cryptography.

An encryption scheme is described by three functions: key generation, $\texttt{KeyGen}(1^\epsilon)$, where given a security parameter, outputs a key pair; encryption, $\texttt{Enc}(m, pk)$, where given a message and a public key outputs a ciphertext; and decryption, $\texttt{Dec}(c, sk)$, where given a ciphertext and a secret key, outputs the encrypted message in plaintext. We do not specify the encryption scheme when it is implicit. In general, given an entity $\mathcal{X}$, $pk_\mathcal{X}$ and $sk_\mathcal{X}$ will denote the public and private keys of $\mathcal{X}$, respectively.

ElGamal

The public key cryptosystem used throughout this thesis is one based on the discrete logarithm problem, presented in 1985 by El Gamal [EG85]. This cryptosystem is not so widely used, as its predecessor RSA, but some interesting properties that we present later in this section will make clear why it is our choice of preference. The three functions of ElGamal are defined as follows.

Algorithm 1 ($\texttt{KeyGen}(\epsilon)$)**.** Select a group G generated by g with order p, according to ϵ. Then $sk \leftarrow_\$ \mathbb{Z}_p$ and $pk = g^{sk}$. Return (pk, sk).

Now, any party that has access to the public key pk, can simply encrypt a message $m \in G$ with the following algorithm.

Algorithm 2 ($\texttt{Enc}(pk, m)$)**.** Let $c = (c_1, c_2) = (g^r, m \cdot pk^r)$ for $r \leftarrow_\$ \mathbb{Z}_p$. Return c.

Now only the owner of the private key can decrypt the ciphertext as follows.

Algorithm 3 ($\texttt{Dec}(sk, c)$)**.** Return $m = c_2 \cdot c_1^{-sk}$.

One of the advantages of this encryption scheme is that it works over prime order groups where the DL problem is hard, and can hence be instantiated over elliptic curves. This allows for smaller key or ciphertext sizes, while maintaining the same security level [Bar16]. Moreover, from Bernhard

$\mathtt{Exp}^{\mathtt{nm\text{-}cpa},b}_{\mathcal{A},\mathcal{D}}(\epsilon)$:
  $(pk, sk) \leftarrow \mathtt{KeyGen}(1^\epsilon)$
  $m_0, m_1 \leftarrow \mathcal{A}(\text{“find”}, pk)$
  $c^* = \mathtt{Enc}(pk, m_b)$
  $b' \leftarrow \mathcal{A}^{O_D}(\text{“guess”}, pk, c^*)$
  Output b'

$O_D(\vec{c})$:
  If $c^* \in \vec{c}$ then return $\perp$
  $\vec{m}_i = \mathtt{Dec}(sk, \vec{c}_i)$
  Return $\vec{m}$

Figure 2.1: In the NM-CPA experiment $\mathtt{Exp}^{\mathtt{bpriv},b}_{\mathcal{A},\mathcal{D}}\mathcal{AV}$, the adversary $\mathcal{A}$ finds two messages m_0 and m_1 of which it asks an encryption of the challenger. It is then allowed to ask the decryption of a vector of ciphertexts $\vec{c}$ of its decryption oracle O_D. It may only call this oracle once.

et al.'s work on Helios [BPW12] we know that in the random oracle model under the DDH assumption the ballot encryption scheme based on ElGamal with a non-interactive proof of correct construction is NM-CPA secure, that is,

$$\left|\Pr\left[\mathtt{Exp}^{\mathtt{nm\text{-}cpa},0}_{\mathcal{A},\mathcal{D}}(\epsilon) = 1\right] - \Pr\left[\mathtt{Exp}^{\mathtt{nm\text{-}cpa},1}_{\mathcal{A},\mathcal{D}}(\epsilon) = 1\right]\right|$$

is negligible in ϵ, where $\mathtt{Exp}^{\mathtt{nm\text{-}cpa},b}_{\mathcal{A},\mathcal{D}}$ is as in Figure 2.1.

However, the property that motivates the usage of this scheme is its additive homomorphic property, which allows one to perform additions over encrypted data without decrypting it first. To see this, let $l_1, l_2 \in \mathbb{Z}_p$ be two messages that one wants to encrypt and operate over with. To leverage the homomorphic property, we encode these values as group elements $m_1 = g^{l_1}, m_2 = g^{l_2} \in G$. Then, given a key-pair $(pk, sk) = \mathtt{KeyGen}(G, g, p)$, applying the binary operation that defines the group, $\cdot$, over the corresponding encryptions results in the encryption of the added messages. More precisely,

$$\begin{aligned}\mathtt{Enc}(pk, m_1) \cdot \mathtt{Enc}(pk, m_2) &= (g^{r_1}, m_1 \cdot pk^{r_1}) \cdot (g^{r_2}, m_2 \cdot pk^{r_2}) = \\ &(g^{r_1+r_2}, g^{l_1+l_2} \cdot pk^{r_1+r_2}) = \mathtt{Enc}(pk, m_1 \cdot m_2) = \mathtt{Enc}(pk, m_{12})\end{aligned}$$

with $r_1, r_2 \leftarrow_\$ \mathbb{Z}_p$ and $m_{12} = g^{l_1+l_2}$, the encoding of the addition of the values l_1, l_2. Now, to extract the message one needs to compute the discrete logarithm of $m_{12} = g^{l_1+l_2}$. However, note that while we are in a group where the hardness of the discrete logarithm is assumed to hold, we can restrict

the message space to be quite small, and hence brute force is feasible. The practicality of this property will become clear by the end of this thesis. This message representation is often referred to as the 'exponential ElGamal'.

An example of the use of such a property, is the possibility of randomising a ciphertext. Assume you have the encryption of l_1 under ciphertext $c_1 = (g^{r_1}, g^{l_1} \cdot pk^{r_1})$ with $r_1 \leftarrow_\$ \mathbb{Z}_p$. Now, one can randomise the ciphertext without changing the encrypted value or accessing the private key. Let $c_0 = (g^{r_0}, g^0 \cdot pk^{r_0})$ with $r_0 \leftarrow_\$ \mathbb{Z}_p$. By leveraging the additive homomorphic property, one can compute $c_3 = c_1 \cdot c_0 = (g^{r_1+r_0}, g^{l_1} \cdot pk^{r_1+r_0})$, such that c_3 looks random in comparison with c_1, but $\texttt{Dec}(sk, c_3) = \texttt{Dec}(sk, c_1)$. We refer to this as randomisation of a ciphertext.

Digital signatures

We introduce the notation of digital signatures, as they constitute an important role in some of our protocols. However, we do not present the exact algorithms, as we use them in a more generic fashion. A signature scheme is defined by two algorithms: the signing algorithm $s = \texttt{Sign}(sk, m)$ that signs messages $m \in \{0,1\}^*$; and a verification algorithm $\texttt{SignVerify}(pk, s, m)$ that outputs $\top$ if `Sign` is a valid signature on m and $\perp$ otherwise. There exists both an RSA based signature algorithm [RSA78] as well as a Discrete-log based construction [Sch90].

Threshold Cryptography

Many situations benefit from the fact that not a single entity has the complete ownership of a secret. If you take for example a personal safe in a bank, the lock requires two keys, one by an employee of the bank, and the other by the user. In cryptography, such situations also exist, where one requires the participation of several parties to use a secret. Moreover, in threshold cryptography, it is possible to build schemes where a secret is shared amongst k parties, and the protocol requires at least t parties, where $t \leq k$, to cooperate in order to use the secret. For example, in threshold encryption, a secret key is shared among 10 parties, such that 7 (or more) parties need to cooperate in order to decrypt a ciphertext encrypted with the corresponding key.

In this subsection we introduce a distinct notation than the one presented above to distinguish between standard cryptography and threshold cryptography. Throughout this thesis we only instantiate threshold cryptography for encryption, and hence the notation for other use cases (e.g. signatures) is omitted.

To generate a key, the different parties jointly run the $\texttt{DistKeyGen}(1^\epsilon, t, k)$

protocol where ϵ is the security parameter, t is the number of parties needed to decrypt ciphertexts, and k the total number of parties. They follow the protocol as presented in [Ped91b]. Such a protocol allows a set of trustees to compute a public key pair where the public key is directly computed from the different shares of the private key, meaning that the 'full' private key is never computed. This protocol outputs a public encryption key pk_d and each party obtains a private decryption key $sk_{d,i}$.

2.2.2 Commitment Schemes

A commitment scheme is usually introduced with the following analogy. Suppose that Alice wants to 'commit' to a sentence, without (initially) disclosing it. Then she writes the sentence in a piece of paper, and introduces it in a locked box, which she keeps the key. She gives this box to Bob. Whenever Alice wants to disclose the initially committed sentence, she simply gives the key to Bob. In this way, Bob is certain that Alice did not change her mind from the moment he received the box. In commitment schemes, this key is named the 'opening key'.

Commitment schemes consist of three protocols. The key generation, the commit, and the reveal protocols. In many cases the definition of these properties require no interaction amongst the parties, and are hence called *non-interactive* commitment schemes.

Definition 2.2.1. *A non-interactive commitment scheme involves a sender and a receiver, and is formed by the following three algorithms* $\texttt{Com} = (\texttt{Setup}, \texttt{Commit}, \texttt{VerifyCommit})$. $\texttt{Setup}(\epsilon)$ *takes the security parameter and outputs the commitment key,* ck. $\texttt{Commit}(ck, m)$ *takes the commitment key* ck *and a message* m*, and outputs a commitment* c*, and its corresponding opening,* o*. Finally,* $\texttt{VerifyCommit}(ck, c, o, m)$ *takes as input the key, commitment, opening and message, and accepts,* $\top$*, or rejects,* $\bot$*. This tuple of algorithms need to have, in addition, the following properties:*

Correctness *If the sender and receiver follow the protocol, then the receiver always accepts.*

Binding *Any polynomial-time adversary,* $\mathcal{A}$*, can extract two different messages,* m_1, m_2 *such that* $\texttt{VerifyCommit}(ck, c, o, m_1) = \texttt{VerifyCommit}(ck, c, o, m_1) = \top$ *with negligible probability.*

Hiding *For any polynomial-time adversary,* $\mathcal{A}$*, given a commitment,* c*, finding a message* m *such that* $\texttt{VerifyCommit}(ck, c, o, m) = \top$ *happens with negligible probability.*

Moreover, one makes the following distinctions of the binding and hiding properties. If the adversary is defined to be computationally unbounded, we say that the commitment scheme is information-theoretically binding or information-theoretically hiding. It is straightforward to prove that a commitment scheme cannot be both information-theoretically binding and hiding simultaneously. Assume that we have a commitment scheme, $(\texttt{Commit}, \texttt{VerifyCommit})$, which is information-theoretically hiding. This means than a computationally unbounded adversary can not get any information about the committed value. Given that `Commit` is public, the (unbounded) adversary could try all possible openings, and still not be able of determining which is the opening which the prover committed to. This means that, for a given commitment c, there exists $o_1 \neq o_2$ and $m_1 \neq m_2$ such that $\texttt{VerifyCommit}(ck, c, o_1, m_2) = \texttt{VerifyCommit}(ck, c, o_2, m_2) = \top$. However, this precludes the possibility of the scheme to be information-theoretically binding, as an unbounded adversary would find these two distinct openings, that pass verification.

The commitment scheme mostly used throughout this thesis is an information-theoretically hiding commitment scheme. This means that no matter how powerful an adversary is, it will never be able to extract the committed message without the knowledge of the o. More precisely, we use the Pedersen commitment scheme, introduced below.

Pedersen Commitment Scheme

In this thesis we use two variants of the Pedersen Commitment scheme. The first was the original work by Pedersen in 1991 [Ped91a]. The algorithms of the Pedersen Commitment scheme are defined as follows. We consider an instantiation on a group finite cyclic G of order a prime p.

Algorithm 4 (Pedersen Commitment)**.** Let $m \in \mathbb{Z}_p$.

$\texttt{Setup}(\epsilon)$: Let $g, h \leftarrow_\$ G$. Output $ck = (g, h)$.
$\texttt{Commit}(ck, m)$: Let $o \leftarrow_\$ \mathbb{Z}_p$. Output $(c, o) = (g^m \cdot h^o, o)$.
$\texttt{VerifyCommit}(ck, c, o, m)$: Output $\top$ if $c = g^m \cdot h^o$, else return $\bot$.

The second variant was presented by Groth in 2009 [Gro09] and functions similarly to the original construction, with the difference that one can commit a vector of values, rather than a single one. The algorithms of the Pedersen Vector Commitment scheme for vector messages, $\boldsymbol{m}$, of size n are defined as follows.

Algorithm 5 (Pedersen Vector Commitment)**.** Let $\boldsymbol{m} \in \mathbb{Z}_p^n$.

$\texttt{Setup}(\epsilon)$: Let $\boldsymbol{g} \leftarrow_\$ G^n, h \leftarrow_\$ G$. Output $ck = (\boldsymbol{g}, h)$.
$\texttt{Commit}(ck, \boldsymbol{m})$: Let $o \leftarrow_\$ \mathbb{Z}_p$. Output $(c, o) = (\boldsymbol{g}^{\boldsymbol{m}} \cdot h^o, o)$.
$\texttt{VerifyCommit}(ck, c, o, m)$: Output $\top$ if $c = \boldsymbol{g}^{\boldsymbol{m}} \cdot h^o$, else return $\bot$.

This particular instantiation of commitment schemes is information-theoretically hiding, and computationally binding. Moreover, these commitment schemes are additively homomorphic, similar to the ElGamal encryption scheme presented above. Consider two commitments, c_1, c_2, for messages $\boldsymbol{m}_1, \boldsymbol{m}_2 \in \mathbb{Z}_p^n$,

$$c_i = \boldsymbol{g}^{\boldsymbol{m}_i} h^{o_i} \text{ for } i \in \{0, 1\}.$$

Then, we have that applying the group operation to the commitments equals the commitment of the addition of the messages, whose opening is the sum of the openings. More specifically

$$c_1 \cdot c_2 = \boldsymbol{g}^{\boldsymbol{m}_1} h^{o_1} \cdot \boldsymbol{g}^{\boldsymbol{m}_2} h^{o_2} = \boldsymbol{g}^{\boldsymbol{m}_1 + \boldsymbol{m}_2} h^{o_1 + o_2}.$$

2.2.3 Zero-knowledge proofs

Throughout this thesis, we rely on a well known class of cryptographic protocols, namely ZKP. They were introduced by Goldwasser *et al.* [GMR85], and they enable one party (prover) to convince another (verifier) about the validity of a certain statement. The statement being proved must include the assertion that the prover has such knowledge, without revealing any other information about the knowledge itself. Let $\{\mathcal{R}_j\}_{j \in \mathbb{N}}$ be a family of polynomial-time decidable relations $\mathcal{R}$ on pairs (v, w). The relation is constituted by the common input, v, and the private input of the prover, or witness, w. To denote that a relation holds, we write $\mathcal{R}(v, w) = 1$, and use $\mathcal{R}(v, w) = 0$ otherwise. Both the common input and witness may be formed by one or more elements.

In this thesis we take particular interest in ZKPs where the verifier is honest, and generates honest randomness, which we replace by a random function modeled as a random oracle [FS87]. Despite this limitation, they turn out to be very versatile tools. They consist of several messages, where initially the prover sends a message, a_1, the verifier responds with a challenge, c_1, prover proceeds with another message a_2, and so on and so forth, until they reach the final round and the prover ends with the response, r. At the end of the interaction, the verifier can perform checks to ensure that the statement being proven is true.

A zero-knowledge proof system is defined by three probabilistic polynomial time algorithms, $(\mathcal{K}, \mathcal{P}, \mathcal{V})$, the generator, prover and verifier. The generator takes as input a security parameter written in unary form, 1^ϵ, and

builds the common input of a proof, $pp \leftarrow \mathcal{K}(1^\epsilon)$. In our research, we use only common inputs that do not need to be honestly generated, meaning that the output of $\mathcal{K}$ can be publicly verified. The $\mathcal{P}$ and $\mathcal{V}$ algorithms take as input (pp, u, w) and (pp, u) respectively. We denote the interaction between prover and verifier, and the latter's output (0 if valid or 1 otherwise) by $\langle \mathcal{P}(pp, u, w) || \mathcal{V}(pp, u) \rangle$. We consider relations $\mathcal{R}$ that consist of a three element tuple (pp, u, w), which we refer as the common input, instance and witness respectively. We define the set of all valid instances as $\mathcal{L}_\mathcal{R} = \{(pp, u) | \exists\, w \colon (pp, u, w) \in \mathcal{R}\}$. The protocol $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ is called a zero-knowledge proof system if it has perfect completeness, knowledge soundness and special honest-verifier zero-knowledge as described below. We now proceed with a formal definition of the properties a proof system needs to have to be considered a zero-knowledge proof.

Definition 2.2.2 (Perfect Completeness)**.** *A proof system is perfectly complete if for all PPT adversaries* $\mathcal{A}$

$$\Pr\left[\begin{array}{r} pp \leftarrow \mathcal{K}(); (u, w) \leftarrow \mathcal{A}(pp) : \\ (pp, u, w) \notin \mathcal{R} \vee \langle \mathcal{P}(pp, u, w) || \mathcal{V}(pp, u) \rangle = 1 \end{array}\right] = 1$$

Paraphrasing, this means that whenever prover and verifier proceed with the protocol, the verifier will always validate the proof.

Definition 2.2.3 (Knowledge soundness)**.** *A proof system has (strong black-box) computational knowledge soundness if for all DPT* $\mathcal{P}^*$ *there exists a PPT extractor* $\mathcal{E}$ *such that for all PPT adversaries* $\mathcal{A}$

$$\Pr\left[\begin{array}{r} pp \leftarrow \mathcal{K}(1^\epsilon); (u, s) \leftarrow \mathcal{A}(pp); w \leftarrow \mathcal{E}^{\langle \mathcal{P}^*(s) || \mathcal{V}(pp,u) \rangle}(pp, u) : \\ b = 1 \wedge (pp, u, w) \notin \mathcal{R} \end{array}\right]$$

is negligible with respect to ϵ*.*

Here, the oracle $\langle \mathcal{P}^*(s) || \mathcal{V}(pp, u) \rangle$ *runs a full protocol execution from the state* s*, and if the proof is successful it returns a transcript of the prover's communication. The extractor* $\mathcal{E}$ *can ask the oracle to rewind the proof to any point and execute the proof again from this point on with fresh challenges from the verifier. We define* $b \in \{0, 1\}$ *to be the verifier's output in the first oracle execution, i.e., whether it accepts or not, and we think of* s *as the state of the prover. If the definition holds also for unbounded* $\mathcal{P}^*$ *and* $\mathcal{A}$ *we say the proof has statistical knowledge soundness.*

The definition can then be paraphrased as saying that if the prover in state s makes a convincing proof, then we can extract a witness.

Definition 2.2.4 (Special Honest-Verifier Zero-Knowledge)**.** *A proof system is computationally special honest-verifier zero-knowledge (SHVZK) if there exists a PPT simulator S such that for all state-full interactive PPT adversaries $\mathcal{A}$ that output (u, w) such that $(pp, u, w) \in \mathcal{R}$ and randomness ϕ for the verifier*

$$\Pr\left[\begin{array}{r} pp \leftarrow \mathcal{K}(1^\epsilon); (u, w, \phi) \leftarrow \mathcal{A}(pp); \\ \texttt{view}_\mathcal{V} \leftarrow \langle \mathcal{P}(pp, u, w) || \mathcal{V}(pp, u, \phi) \rangle : \\ \mathcal{A}(\texttt{view}_\mathcal{V}) = 1 \end{array}\right] - \Pr\left[\begin{array}{r} pp \leftarrow \mathcal{K}(1^\epsilon); (u, w, \phi) \leftarrow \mathcal{A}(pp); \\ \texttt{view}_\mathcal{V} \leftarrow S(pp, u, \phi) : \\ \mathcal{A}(\texttt{view}_\mathcal{V}) = 1 \end{array}\right]$$

is negligible with respect to ϵ. We say the proof is statistically SHVZK if the definition holds against unbounded adversaries, and perfect SHVZK if the probabilities are equal.

This definition can be paraphrased as saying that for every valid protocol run, a simulator can generate simulated random view which is indistinguishable from the original run.

Blum *et al.* [BFM88] introduced non-interactive Zero-Knowledge Proofs (NIZKPs), which enable the prover to prove the validity of a statement without interacting with the verifier. This, in turn, allows any party to act as a verifier. The prover simply sends one message to the verifier, and the verifier either accepts or rejects it. The challenge step is then replaced by a source of randomness. This is known as the Fiat-Shamir heuristic [FS87].

We adopt the Camenisch-Stadler notation [CS97] to denote such proofs and write, for example,

$$\Pi = SPK\{(x) \colon X = g^x\}, \tag{2.1}$$

to denote the non-interactive zero-knowledge proof that the prover knows the discrete logarithm x of X with base g. To represent the verification procedure, we use $\Pi.\texttt{Verify}$.

As described in the introduction, we are interested in use cases where users might be behind a device with low bandwidth network or with constraints in the amount of data to be transmitted (e.g. mobile devices). Similarly, we study scenarios where the server needs to scale to millions of users, and in the context of ZKPs, to millions of proofs verifications. It is relevant, then, to introduce a concept that has been of high interest for the academic and

industrial community, namely Succint Non-interactive ARguments of Knowledge (SNARK) [Bit+12]. A SNARK is also defined by three algorithms, $(\mathcal{K}, \mathcal{P}, \mathcal{V})$, which in contrast to ZKPs, satisfy the properties of completeness, succinctness and knowledge soundness. The notion of succinctness, the only not described above, is defined as follows:

Definition 2.2.5 (Succintness)**.** Π *is said succinct if the running time of* $\mathcal{V}$ *is* $poly(\epsilon)(\epsilon + |x| + \log|w|)$ *and the proof size is* $poly(\epsilon)(\epsilon + \log|w|)$*, where* $|x|$ *is the size of the computation (with respect to the number of gates), and* $|w|$ *is the size of the witness.*

Note that there is no requirement of the zero-knowledge property for a system to be considered a SNARK. There exists, however, the ZK-SNARKs, which do require the zero-knowledge property. There has been a huge amount of advances in the field in the past years [Bün+18; Chi+20; Gro16; GWC19; Mal+19] just to cite a few. Of particular interest in this thesis is the construction of Groth [Gro16], which has an existing framework to easily generate ZKPs out of code, namely ZoKrates [ZoK19].

Some examples

To introduce the power of ZKP, in this subsection we present a few constructions. To begin, one could consider the example of equation (2.1), and use it as a zero-knowledge proof that the prover is the owner of an ElGamal key-pair. In this way a verifier is convinced about the statement that the prover knows the private key, but the latter does not disclose anything about the secret.

This can also be used for the decryption procedure. Assume a scenario where several parties encrypt their secret under a public key. Then, the homomorphic property of the ElGamal encryption scheme is leveraged to compute the addition. Finally, the key-pair owner proceeds with the decryption to show to all the participants what the sum of the secrets is. However, given that only the key-pair owner knows the secret, it could cheat the rest of the participants, by simply outputting a random number. To avoid this, all participants would require the key-pair owner to compute a proof of correct decryption. Let $c = (c_1, c_2), mg, pk$ be the ciphertext, plaintext, generator and public key respectively, and let sk with $pk = g^{sk}$ be the secret key. Then, the key-pair owner, when disclosing the plaintext m, also provides the following proof:

$$\Pi_D = \{(sk) \colon pk = g^{sk} \wedge m = c_2 \cdot c_1^{-sk}\}$$

As we can see, one can compose several conditions in the construction of a ZKP. As a matter of fact, one can encode any NP-program as a zero-knowledge proof, and prove correct computation. Another representative example can be instantiated over the Pedersen commitment scheme. Assume that a prover commits to two values, and publishes them somewhere. Later, it wants to prove ownership of the commitment by showing that it knows the opening, but without disclosing it. Hence, it can use the following ZKP,

$$\Pi_{Open} = \{(m_1, o_1, m_2, o_2) \colon c_1 = g^{m_1} h^{o_1} \wedge c_2 = g^{m_2} h^{o_2}\}$$

to prove it knows ownership without disclosing the message or the opening. More generally, we can have such a proof for vector commitments.

$$\Pi_{Op} := \{(\boldsymbol{m}, r) \colon c = \boldsymbol{g}^{\boldsymbol{m}} \cdot h^r\}\,.$$

We use the work presented by Hoffmann *et al.* [HKR19] to achieve a proof with logarithmic complexity with respect to the size of the input vector.

One can also instantiate ZKPs to prove statements about the opening, without having to disclose the latter. For example, one can prove that two commitments share the same opening,

$$\Pi_{Eq} := \{(\boldsymbol{m}, r_1, r_2) \colon c_1 = \boldsymbol{g}^{\boldsymbol{m}} \cdot h^{r_1} \wedge c_2 = \boldsymbol{h}^{\boldsymbol{m}} \cdot h^{r_2}\}\,.$$

Again, we use the work presented in [HKR19] to achieve logarithmic communication with respect to the vector opening.

Similarly, one can provably exchange any element in the committed vector by a zero, and prove we did so correctly,

$$\Pi_0 := \big\{(\boldsymbol{m} \in \mathbb{Z}_p^N, r) \colon c = \boldsymbol{g}^{\boldsymbol{m}} h^r \wedge E = g_j^{m_j} \wedge c/E = g_1^{m_1} \cdots g_{j-1}^{m_{j-1}} \cdot g_{j+1}^{m_{j+1}} \cdots g_N^{m_N} h^r\big\}\,.$$

Moreover, the prover can prove some algebraic relation amongst the two openings, for instance that the first is the square of the second [CM99],

$$\Pi_{Sq} := \big\{(m_1, m_2, r_1, r_2 \in \mathbb{Z}_p) \colon c_1 = g^{m_1} \cdot h^{r_1} \wedge c_2 = g^{m_2} \cdot h^{r_2} \wedge m_1 = m_2^2\big\}\,.$$

To prove that a number is within a range, we leverage Bulletproofs [Bün+18]:

$$\Pi_{GE} := \big\{(m, r \in \mathbb{Z}_p) \colon c = g^m h^r \wedge m \in [0, 2^l]\big\}$$

for some positive integer l. We combine these two proofs, Π_{Sq} and Π_{GE}, to prove the correct computation of the floor of the square root of a committed

value. Let $c_1 = g^{m_1} \cdot h^{r_1}$, $c_2 = g^{m_2} \cdot h^{r_2}$ and $c_1^{+1} = c_1 \cdot g = g^{m_1+1} \cdot h^{r_1}$. We want to prove that $m_1 = \lfloor \sqrt{m_2} \rfloor$. It suffices to prove the following two statements:

(i) The square of the committed value in c_1 is smaller or equal than c_2, and

(ii) The square of the committed value in c_1^{+1} is greater than c_2.

We denote this proof by:

$$\Pi_{sqrt} := \{(m_1, m_2, r_1, r_2 \in \mathbb{Z}_p) : c_1 = g^{m_1} \cdot h^{r_1} \wedge c_2 = g^{m_2} \cdot h^{r_2} \wedge m_1 = \lfloor \sqrt{m_2} \rfloor\}.$$

2.3 State of the art

The usage of cryptographic constructions for improving the privacy of the user has been an active area of research since the introduction of public key cryptography in 1976 by Diffie and Hellmann [DH76]. In the coming decade after the introduction of the key exchange mechanism, important foundations where introduced, such as public key encryption [EG85; RSA78], Zero-Knowledge Proofs (ZKPs) [BFM88; GMR85], commitment schemes [EGL85; SRA81], blind signatures [Cha83] or mix-nets [Cha81]. However, it has been in the last decade where we have seen a boom of mainstream usage of some of these technologies such as zero knowledge proofs, blind signatures or mix-nets, which were only used in very particular scenarios. These technologies are, nowadays, used by millions of users through internet voting systems, Zcash [Mie+13], Privacy Pass [Dav+18], Signal [Sig21] or the incentivised test-net of the Nym network [Net20]. In this section we do not aim to cover all recent advances of cryptography aimed at improving user privacy. Instead we focus primarily in sub-linear zero-knowledge proofs (which we use to improve the state of the art in bot detection mechanisms), privacy-preserving advertising and internet voting schemes.

We now provide the state-of-the-art in the two topics that are covered in this thesis, namely bot detection, and the mechanisms used to achieve it, and internet voting.

2.3.1 Sub-linear zero-knowledge proofs - Bot detection

Assessing Humanness To prevent automated programs from abusing online services, the widely adopted solution is to deploy a CAPTCHA system. However, text-based CAPTCHA schemes have been proven to be insecure as

machines achieved 99.8% success rate in identifying distorted text [Che+21; CSJ09; Dou21; Goo+13; YEA08; Zi+20]. Audio-based CAPTCHAs have also been used to assist visually impaired people, but they are difficult to solve, with over half of users failed during their first attempt [Tas+12]. Therefore, CAPTCHA service providers started to test image-based CAPTCHA schemes, which require users to select images that match given description [Goo14]. Nevertheless, in [SPK16; Zho+18] authors demonstrated that more than 70% of image-based Google and Facebook CAPTCHAs can be efficiently solved using deep learning.

In [WKY10], authors designed a multi-level data fusion algorithm, which combines scores from individual clicks to generate a robust evidence. Nevertheless, these CAPTCHA systems require users to perform additional tasks and deliver worse user experience, especially when running on mobile devices [RC13]. Google reCAPTCHA v2 use a risk analysis engine to avoid interrupting users unnecessarily [Goo19]. This engine collects and analyses relevant data during click events. The latest reCAPTCHA v3 no longer requires users to click but instead it studies user interactions within a webpage and gives a score that represents the likelihood that a user is a human [Dev18]. A similar method has been performed by Süzen [Sü21], which also uses mouse movements, keyboard gestures, or web behavior among others. Although these CAPTCHA schemes are transparent to users, a plethora of sensitive data, including cookies, browser plugins, and JavaScript objects, is collected [O'R15] that could be used to fingerprint the user.

With the proliferation of smartphones, more humanness attestation schemes proposed leveraging the variety of available sensors. Most of these schemes require users to perform additional motion tasks. In [SSH13], authors showed that waving gestures could be used to attest the intention of users. In [Fen+20; Gue+15], authors designed a bot detection system that asks users to tilt their device according to the description to prove they are human. In [HKH16], authors presented a movement-based CAPTCHA scheme that requires users to perform certain gestures (e.g. hammering and fishing) using their device. In [DL+12], authors exploited touch screen data during screen unlocking to authenticate users. In [Gue+16], authors suggested a brightness-based bot prevention mechanism that generates a sequence of circles with different brightness when typing a PIN; users will input misleading lie digits in circles with low brightness. In [BGC17], authors proposed a behavioural-based authentication scheme, which uses timing and device motion information during password typing.

Finally, there exists the Invisible CAPTCHA [Gue+18]. In this work, authors leveraged the different device acceleration appearing on a finger touch and a software touch to make a decision about whether a user is a bot.

However, Invisible CAPTCHA is not fully implemented as it requires a secure execution environment and its accuracy is low when device is stable on a table. In addition, it only considers simple tap and vibration events; its accuracy on more complicated touch events (e.g. drag, long press, and double tap) is unclear.

In summary, the state-of-the-art is limited in the three main properties that we expect out of a bot detection mechanism.

A) **Be friction-less:** Most existing mechanisms require the user to solve mathematical quizzes or image/audio challenges [Int19; Phi15], thus severely hampering the user experience. According to studies [All13; Bur+10]: (i) humans only agree on what the CAPTCHA says 71% of the time, (ii) visual CAPTCHAs take 9.8 seconds (on average) to complete, (iii) audio CAPTCHAs take 28.4 seconds (and 50% of the Audio CAPTCHA users quit). The profound degradation of the user experience forces service providers to perform user attestations only sporadically [Fou16; Kah20; Loz19] in an attempt to save their declining conversion rates. Related research works [Gue+15; HKH16; SSH13] reduce user friction by requiring, for example, the user to tilt their phone during humanness verification, which still requires the user to perform a non-natural action.

B) **Be privacy-preserving**: To mitigate the above, mechanisms like reCAPTCHA v3 [Liu18] (i) track the user while browsing a webpage, (ii) send raw tracking data to third party attestation servers, where (iii) a "risk score" representative of humanness is computed and then (iv) shared with the webmasters. Of course, this pervasive behavioral tracking raises significant privacy concerns [Cla19; Sch19]. Similarly, there are sensor-based approaches [Gue+15; Gue+18; HKH16] that transmit raw mobile sensor data to remote attestation servers; something that as reported, may reveal keystrokes, gender, age, or be used to fingerprint users [Dav+17; DBC16; Mal+18; Pap+17; RO+16; SS+18; ZBS19].

C) **Be broadly accessible:** The majority of existing CAPTCHA solutions are not accessible to all users [Con19; Hol+19]. According to a survey [Min17], CAPTCHAs constitute the first source of difficulty for visually impaired users. Meanwhile, human attestation mechanisms designed for visually impaired people (like audio-based reCAPTCHA) have been exploited to bypass CAPTCHAs by providing automatic responses [Seb19; Tam+09].

Privacy-Preserving and Provable ML A potential approach to offer privacy-preserving machine learning is to evaluate the model locally, avoiding data to be sent to the server. However, if such approach is taken without

proving correct evaluation of the model, then verification may be lost [BR11; GCF11; TVD17]. In cases such as bot detection the user's interest might be of faking the evaluation model, and this may be vulnerable to user attacks.

Another approach consists in encrypting the data on the client, and run ML model on such encrypted data at the server. This can be achieved via Fully Homomorphic Encryption (FHE)[5] [BLN14; Dow+16; GLN12]: clients encrypt their data with their own keys and send the ciphertext to the server to evaluate an ML model. Next, the server sends the outcome of the homomorphic computation back to the user who would provably decrypt it and send back to the server. FHE has received increased interest in the past years. However, it is studied in scenarios where a single client encrypts a large dataset which is evaluated in the server. For the case of millions of users, each with their own encryption keys, FHE has not seen a wide interest due to its computational overhead.

To the best of our knowledge, there are few papers aiming to provide provable machine learning local evaluation without a trusted execution environment. Davidson *et al.* [DFL14] try to solve a similar problem, where personalization of a user device is done by evaluating a model locally on the user's machine. This work uses Bayesian classification, for which they need from 100-300 feature words. The generation of correct model evaluation for such range of feature words ranges from 30 to 80 seconds in a laptop. Moreover, this study uses standard techniques for constructing zero-knowledge proofs, which give a big overhead to the prover and verifier. Danezis *et al.* [Dan+12] propose a solution where after the evaluation of Random Forest and Hidden Markov models, the user generates a zero-knowledge proof of correct evaluation. However, this paper misses an evaluation study or availability of the code, which makes a study of the scalability of their approach inaccessible.

The available zero-knowledge schemes are, however, not ideal for our use case. We need a proof system with low prover computational overhead, while maintaining the communication complexity and verifiable computation low. A recent paper by Groth [Gro16] presented a quasi-optimal solution for the last two properties. Such a scheme falls in the category of Succinct Non-interaction ARgument of Knowledge (SNARK), where any nondeterministic polynomial time (NP) statement can be proven with minimal proof size and verification time. However, this construction comes with some caveats. A trusted setup is required, and to make matters worse, for our particular scenario, it was over 1GB of data that the user needed to download. Moreover,

[5]Similar to the additive homomorphic property described in Chapter 2, FHE can perform both addition and multiplication over encrypted texts.

the complexity of the user is considerably affected, with proof generation time reaching around 174.5 seconds in a 2018 Samsung S9. There has been a high number of improvements in the past years with proposals reducing the assumption requirements or prover time, however they all fall short for our goals. An important observation is that our scheme does not require a general computation zero-knowledge proof, but we can build an *ad-hoc* proof for our statement, improving on setup size and prover time. As a matter of fact, an inner product proof shows sufficient for a bot-detection mechanism. Groth [Gro09] presented the best current solution that offered simultaneously zero-knowledge and soundness. Recent advances [Boo+16; Bün+18] have presented sub-linear inner product proofs which are only sound. However, they fail to present an explicit definition and proof of a zero-knowledge version.

2.3.2 Coercion resistance in internet voting

Coercion resistance comes at a high cost, as can be seen in the existing literature (we forward the reader to Section 4.4 for a formal definition of the security properties). In the present moment there are three methods to achieve coercion resistance in remote elections: fake credentials, re-voting, and, a recently introduced notion, vote-flipping. Other mechanisms to provide a weaker form of coercion resistance exist. Such weaker notions assume that a voter cannot prove its vote to a coercer. This requires the adversary to not be present during vote cast. This is a very strong assumption that we exclude from our analysis of the state-of-the-art. One recent example of such a mechanism was presented by Dimitriou [Dim20], which not only assumes that the coercer is not present during vote cast, but also assumes anonymous communications, and secure forgetting of a secret value.

In particular, the works that cover the three methods to prevent coercion in electronic elections are the following. First comes the use of fake credentials (or fake passwords) introduced in the work by Juels, Catalano and Jakobsson [JCJ05] (referred to as the JCJ protocol), and used in several new constructions [Ara+16; BGR12; CH11; Gro+19; MCC08; Ye+21]. In these schemes, during the registration phase, the voter receives a bunch of credentials out of which some are correct and will cast a valid vote; while others are invalid, and cast a non countable vote. When a voter is coerced, it uses a fake credential to cast a vote, making this vote invalid during the tallying phase. In a moment where the coercer is absent, the voter can cast a vote using the correct credential. These type of solutions assume the coercer to be absent during registration and at a given moment throughout the election. Moreover, they require the voter to privately and securely store

cryptographic material (being able to hide it from the coercer) and to lie convincingly under the pressure of the coercer, which may indeed be a challenge for some. Finally, in order to provide coercion resistance, they do not provide feedback to the user of whether the vote was cast with a valid credential, resulting in a high dependence of the human memory and usage of the correct credentials at the right time.

Secondly, comes a recently introduced mechanism, named vote flipping. Chaum *et al* [Cha+21] present a new strategy to avoid coercion, which consists in providing the voters with a 'code' that allows them to flip their vote before the end of the election. However, the problem of such a solution is that it requires the voter to maintain cryptographic state. In order to flip the vote, the user needs to provide a zero-knowledge proof of knowledge of a secret key related to a particular public key. As with fake credentials, this mechanism requires the user to safely store and keep cryptographic material.

Thirdly, coercion can be mitigated by the use of re-voting. Voters can cast multiple votes, and the last vote is counted. Contrary to the previous approach, this solution requires the voter to be able to cast a vote after being coerced and before the election closes. However, there is no registration process where the coercer must be absent, the user may not necessarily need to store cryptographic material, and the voter can suffer from 'perfect coercion': the coercer may indicate exactly how the user must act, without the latter needing to lie about its actions while coercion is taking place.

We choose the latter solution as we believe that its core assumptions are more realistic for real world scenarios. In Section 4.1 we give an intuition of why these assumptions are more realistic than the ones assumed in fake credentials based solutions. Re-voting has been used in several constructions proposed in current literature. The main challenge that one finds when allowing multiple voting for coercion resistance is filtering. In order to mitigate coercion attacks, voting must be *deniable*, meaning that the adversary must not be able to determine whether a specific voter re-cast her ballot, or more generally, not be able to know which of the voters re-voted. Otherwise, and adversary can perform what is known as the the 1009 attack [Smi05] in which the coercer forces a voter to cast a specific number of ballots and looks for a group of that size in the filtering step. If such group does not exist, the coerced voter has revoted. Concurrently, auditability that the process happened as expected must be provided.

The JCJ protocol [JCJ05] allows multiple voting, and the filtering stage is not deniable, i.e., one can determine whether a given vote has been filtered (note that this is not how JCJ achieves coercion resistance, but with fake credentials). The authors achieve this by using a cryptographic tool, Plaintext Equivalence Texts (PETs), allowing an entity to compare two ci-

phertexts without the need of decrypting any of them. With this tool, they are capable of comparing every pair of credentials used to cast a vote. If two votes are cast by the same voter, they take the last. The complexity of comparing each pair of credentials results in a computation of $\mathcal{O}(N^2)$ PETs, with N being the total number of votes cast, making it an unusable scheme even for small scale elections. This complexity was later reduced by Araújo *et al.* [Ara+16], however, still not offering filter deniability.

The work presented by Spycher *et al.* [SHD10] allows the voter to cast a vote in an electoral school, and therefore overwrite any previously cast votes. However, this results in the requirement of presence accessibility of the voter. Another used method is to do the filtering as a black box protocol. Trust is given to a server which will filter all votes and publish all re-encrypted votes [Gjø10]. This avoids any tracking or knowledge of which votes have been filtered, but the verifiability is totally dispensed.

To the best of our knowledge, current filtering schemes that offer a trustless deniable voting scheme which, at the same time have public verifiability, are the ones proposed by Achenbach *et al.* [Ach+15] and Locher *et al.* [LHK16] which use similar solutions. In a protocol where a Public Bulletin Board (PBB) is used in order to allow verifiability, the filtering process must be done after the mixing, else wise, a voter (and thus the coercer) would know whether her vote was filtered or not. However, after the mixing, it is not longer possible to know which is the order of the votes, and therefore, before inserting the votes in the mix-net there must be some kind of reference of their order. This solution faces this by performing Encrypted PETs (EPET) on the credentials for all votes against all lately cast votes before mixing. This consists in performing a PET whose output is encrypted, and if any of the comparisons among the credentials is equivalent, then the output of the EPET hides a random number (alternatively, the encrypted number will be a one). Votes are then included in a mix-net. The filtering stage happens after mixing by decrypting the EPET, and all votes which output a random number will be filtered out. This achieves deniability with no trust in any external entity. However, there is a high increase in the complexity as the EPET have to be performed for each pair of votes, resulting in a complexity of $\mathcal{O}(N^2)$ distributed (among several servers) EPET calculations prior to the mixing, and in $\mathcal{O}(N)$ distributed decryptions and zero knowledge proofs of correct decryption during the filtering. Again, this makes these solutions unusable even for elections of tens of thousands of voters.

Note that all these filtering schemes presented above are implemented by comparing the voting credentials, needing like that for voters to maintain a cryptographic state throughout the election.

Everlasting privacy is a concept that was introduced by Moran *et al.*

[MN06]. In this scheme perfectly hiding commitment schemes are used to hide vote intention, and the hiding values of the commitment schemes are exchanged through private channels. This idea can be used in any scheme based in homomorphic tallying. In [Gro+19], an everlasting privacy scheme is presented in the JCJ setting, giving to users the burden of having to handle several credentials, but with the improvement on previous schemes that it only assumes the existence of private channels. Locher *et al.* [LHK16] present an everlasting privacy scheme in an information-theoretical sense, with the main drawback of having a quadratic proposal which makes it unusable even for medium sized elections.

CHAPTER 3

zkSENSE—Private Human Attestation

In this chapter, we introduce zkSENSE, a new humanness verification mechanism which was published at the 21st Privacy Enhancing Technologies Symposium (PETS 2021) [Que+21]. In Section 3.1 we define the model we select to perform the attestation. Next, in Section 3.2 we introduce the computation we need to prove in zero-knowledge to maintain the privacy of the user. Section 3.3 presents an ad-hoc construction respectively. In the latter, we introduce a new zero-knowledge proof to reduce the complexity of the humanness proof. We then present zkSVM, the proof of humanness, in Section 3.4. We then provide a security analysis, a description of the implementation and a performance analysis in Sections 3.5, 3.6 and 3.7. For the implementation and performance we include details on how we build a SNARK based approached, that showed to be less performant. We conclude the chapter in Section 3.8 with an improvement to the data consumption by batching the proofs we introduced in Section 3.3.

3.1 Human attestation

The key intuition behind zkSENSE, shown in Figure 3.1, is that whenever a (human) user interacts with the mobile's display, the force applied during the touch event generates motion. This motion is captured by the embedded IMU (Inertial Measurement Unit) sensors (e.g.: accelerometer and gyroscope). By contrast, when there is automated user activity (e.g.: simulated touches) there is no external force exerted by fingers, and thus there is no noticeable

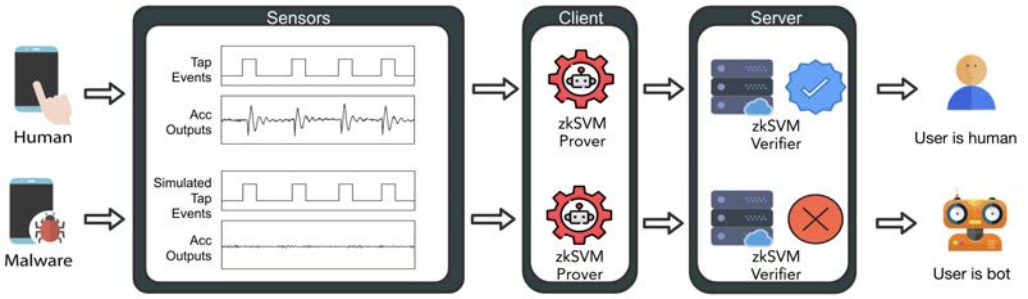


Figure 3.1: High-level overview of the zkSENSE architecture. An integrated ML-based classifier studies the patterns of sensor outputs right before, during, and shortly after a click event. To avoid leaking sensitive sensor output outside the device, the classification appears on the user side and the client has to prove the integrity of the reported result to the server.

change in the output of the above sensors. The sensor data is then used in the client's device to generate a proof of humanness, which is finally verified by the server.

Figure 3.2 shows a snapshot of the output produced by IMU sensors during click events performed both by a human (left plots) and an automated agent (right plots). During the click events (highlighted in red) the accelerometer (top plots) senses a max rate of change of 0.6 in case of human and 0.07 for an automated agent, i.e. 8.5× greater maximum linear acceleration movement. Similarly, the gyroscope (bottom plots) senses a max rate of change of 0.024 in case of human click and 0.0049 when there is automation, i.e. 4.9× greater maximum angular rotational velocity.

Figure 3.3 shows a snapshot of the same sensors' output in the case of automated clicks coupled with two artificial device movements: vibration

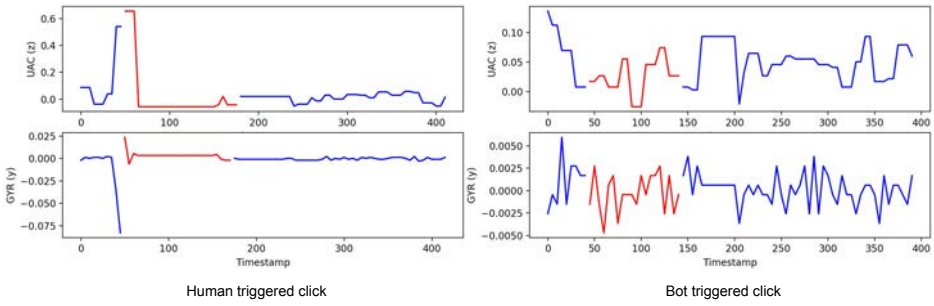


Figure 3.2: Output of gyroscope and accelerometer motion sensors during human and automatically triggered click (in red, the click event). The maximum linear acceleration movement is up to 8.5× greater and the angular rotational velocity is up to 4.9× greater in case of a human triggered click.

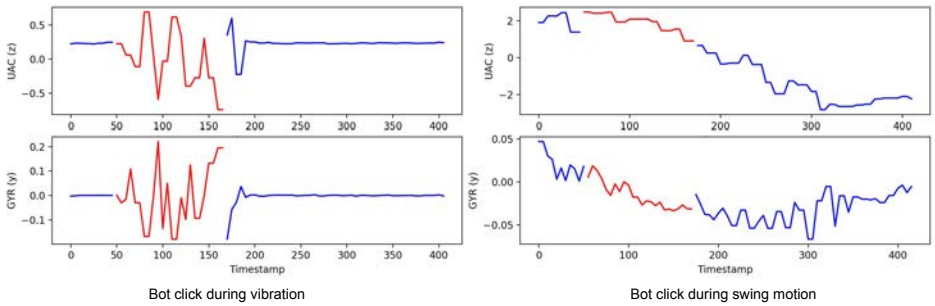


Figure 3.3: Motion sensors output during automatically triggered click with artificial device movement (in red, the click event): (i) during device vibration (on the left) and (ii) when device is docked on a swing (on the right).

and swing motion. In presence of vibration (left plots), we see that the motion generated is comparable with the case of the human click depicted in Figure 3.2. Note that the y-axis of the triggered click is around an order of magnitute smaller than the human triggered click, which implies that the movements detected in the right are micro movements detected by the sensors when the device is not in motion. We see that the accelerometer senses the same force with the case of the human click (this verifies the observations of [Gue+18]) but for a longer time. The gyroscope though senses greater angular rotational velocity and for longer time than in the case of a human's click. In presence of swing motion (right plots), the gyroscope senses similar angular rotational velocity as with the case of the human click, while the accelerometer senses greater linear acceleration movement (up to 3.8×) for a long period.

3.1.1 Classification of Humanness

Building upon the above observations and as depicted in Figure 3.1, zkSENSE uses an ML-based classifier to study the pattern of sensor outputs before, during, and shortly after a click event. Based on this information, the model decides about whether the action was triggered by a human or not.

In zkSENSE, we pre-train a model on a server and we move the classifier to the edge by running it on the user side and only report the result to an attestation server responsible for auditing the humanness of the user. This way, zkSENSE ensures that private sensor data never leaves the user's device. In Figure 3.1, we present the high level overview of our approach. As we can see, an attestation starts with a click (screen touch). The motion sensor outputs generated during this event are used as input to the zkSVM Prover

Data	Data amount
Volunteering Users	10 users
Duration of collection	22 days
Android Devices tested	Google Pixel 3, Realme X2 Pro, Samsung Galaxy S9/S8/S6, Honor 9, Huawei Mate 20 Lite, OnePlus 6
Human events collected	7736 clicks
Artificial events collected	25921 clicks

Table 3.1: Summary of the collected dataset.

module, which runs a trained model to classify if the action was conducted by a human or not.

To collect the necessary ground truth to train the various tested models, we instrumented the open source browser Brave [Bra20] for Android to capture click events (and their corresponding motion sensor traces) performed during browsing. Then, we recruited 10 volunteers who used our instrumented browser for 22 consecutive days for their daily browsing[6]. The device models used are: Google Pixel 3, Samsung Galaxy S9, S8 and S6, OnePlus 6, Realme X2 Pro, Huawei Mate 20 Lite and Honor 9. Volunteers were well-informed about the purpose of this study and gave us consent to collect and analyse the motion sensor traces generated during their screen touch events. We urged volunteers to use their phone as normal.

To generate artificial user traffic, we used `adb` [Dev20] to automate software clicks on 4 of the volunteering devices. To test different attack scenarios, during the automation, we generated software clicks with the device being in 4 different states:

1. Resting on a platform (desk/stand).
2. Being carried around in pocket.
3. Being placed on a swing motion device.
4. While device is vibrating (triggered by `adb`).

As summarized in Table 3.1, by the end of the data collection, we had 7736 human-generated clicks and 25921 artificially generated clicks.

During data collection, accelerometer and gyroscope sensors were sampled at 250Hz. For each click event, we not only consider the device motion

[6]During data collection the instrumented browser was running on the users' personal device so we could not control the services running in the background. The data collected included only the raw sensor data during a user click.

Classifier	$\mathbf{F_1}$ (weighted)	**Recall**
SVM	0.92	0.95
Decision Tree (9 Layers)	0.93	0.95
Random Forest (8 Trees, 10 Layers)	0.93	0.95
KNN	0.92	0.93
Neural Network (Linear Kernel)	0.86	0.95
Neural Network (ReLU Kernel)	0.91	0.96

Table 3.2: Accuracy of the various tested classifiers, where we study Support Vector Machines (SVM), Decision Trees, Random Forests, K-Nearest Neighbor (KNN) and Neural Networks with Linear and ReLU kernels.

during the touch, but also the device motion right before and shortly after the touch. In particular, we consider that the period starts 50ms before the finger touches the screen and finishes 250ms after it. Then, we split each period into two segments: (i) *before releasing finger* and (ii) *after releasing finger*. For each axis (x, y, z) in accelerometer and gyroscope, we calculate the average and standard deviation of its outputs in each segment. In addition, we calculate the consecutive difference of sensor outputs in each segment and use the average and standard deviation of these differences as features.

Using the above features, we test several ML classifiers via 10-fold cross validation. Table 3.2 presents the weighted $\mathbf{F_1}$ score and recall of the different classifiers we tested [Chi92]. Recall is the number of correctly detected positives divided by all positives of the dataset. In this context, *recall* means the proportion of correctly identified artificial clicks over all artificial clicks. In other words, recall indicates the ability to capture artificial clicks. The $\mathbf{F_1}$ score is the harmonic mean of the precision (where the precision is determined by dividing the number of true positives over the number of all positive results) and the recall represents the number of correctly identified bots out of all bots in the dataset. We choose weighted $\mathbf{F_1}$ score as an evaluation index because our dataset is unbalanced.

3.2 Privacy-Preserving and Provable ML

To preserve user privacy, human attestation in zkSENSE is performed in the user's device and only the *result* of the attestation is shared with the server. To ensure that the server can verify the integrity of the transmitted result, the user includes a commitment of the sensors, together with a proof that the result corresponds to the model evaluated over the commited values.

We build the two ML evaluation proving components of zkSENSE: (i) zkSVM Prover and (ii) zkSVM Verifier. The zkSVM Prover checks on the client whether a user is a human based on a model we pre-trained (Sec-

tion 3.1.1), and generates a proof to ensure its proper execution. The zkSVM Verifier, on the server's side, checks that the proof is correctly generated. If the verification is successful, the server will know that (a) the ML-based humanness detection model classifies the user as human or non-human based on the committed sensor outputs, and that (b) the used model is the genuine one, without though learning the value of sensor outputs.

Table 3.2 shows that the different classifiers tested achieve similar accuracy. While decision trees, random forests, or neural networks provide slightly higher $\mathbf{F_1}$ accuracy than SVM (see Table 3.2), in zkSENSE we choose SVM as the underlying model due to its simplicity at evaluation time and its suitability with zero-knowledge proofs. Hence, our zkSENSE's accuracy in assessing the humanness of a user is 92%. Contrarily, neural networks need to perform non-linear operations, while decision trees require several range proofs, which are expensive operations to prove in zero-knowledge. As mentioned above, the SVM model we trained uses as features the average (μ) and standard deviation (σ) of sensor outputs, together with the average and standard deviation of the consecutive difference vector. On top of that, before applying the SVM model, the extracted features need to be normalised. The goal of normalisation is to change data values to a common scale, without distorting differences in the ranges of values. Then, trained SVM weights are assigned to each normalised feature to calculate the SVM score.

Suppose for each feature f_i, the normalisation mean, normalisation scale, and SVM weight and intercept are M_i, S_i, w_i and c respectively. Then, the SVM score s can be calculated with the equation:

$$s = \frac{1}{e^{-(c+\sum_{i=1}^{N} \frac{(f_i - M_i)}{S_i} w_i)} + 1}. \tag{3.1}$$

Since only the value of f_i is secret, we only need to provably compute $\sum_{i=1}^{N} f_i \frac{w_i}{S_i}$. Given only integer values can be processed in the underlying group arithmetic of Pedersen commitments, we instead prove $\sum_{i=1}^{N} f_i \left\lfloor \frac{w_i}{S_i} 10^d \right\rfloor$ and effectively use the parameter d to preserve d-digits after the decimal points of $\frac{w_i}{S_i}$.

The model trained in this work had as a goal the exploration of the feasibility of generating a zero knowledge proof of correct execution of the model. Training a model using only 10 users with their respective mobile devices is not representative of the whole population, and if this solution is reused, this should be taken into account to train a more accurate model. However, we note that the way that zkSENSE is built allows the server to change and update the model as soon as it gets obsolete. zkSENSE is not bound to a particular trained model, but only to SVM in general. We would

like to emphasize that a model with more data, users, or devices would not affect the proof generation, nor the evaluation performed in Section 3.7.

In order to perform the provable computation of the ML model, we tested two approaches:

- General computation ZKPs.
- Ad-hoc ZKP.

As introduced in Section 2.2, there exists ZKP systems that allow one to encode and prove any type of NP-statement. This seemed ideal at first for our construction, in particular with the existing tools and compilers that abstract the low level knowledge away from the developer. However, as will become clear by the end of the section, this was not the best solution for our scenario. As it turns out, building an ad-hoc proof for our computation resulted in a considerable improvement on the performance on prover side.

3.3 zkSVM—proving humanness with logarithmic complexity

The development of the general computation ZKP approach was ideal for prototyping, but as we will show in Section 3.7, it induces a considerable overhead in the client side. To avoid using a general computation ZKP, in this section we present an ad-hoc model that considerably reduces prover time. However, to achieve this, it is not sufficient with ZKPs presented in Section 2.2 —we require a zero-knowledge inner product proof. However, as presented in Section 2.3.1, such a solution, with logarithmic complexity has not been yet formalised. We begin by formalising such a proof, before presenting the description of zkSVM.

3.3.1 IP-ZKP

Inner product proofs are an important building block of zkSENSE, as it is what allows us prove correctness of average computation and standard deviation. However, existing constructions which have sub-linear complexity only provide the soundness properties of proofs, and not the zero-knowledge. The latter is key for our applications and hence we need to extend the existing solutions to provide honest-verifier zero-knowledge.

Protocol 3.3.1. IP-ZKP This protocol presents the sub-linear Inner Product Zero-Knowledge Proof (IP-ZKP). Our construction is based on the non

zero-knowledge version presented in [Bün+18]. For their use cases, the zero-knowledge property is not required, and hence the lack of an explicit definition and study of a zero-knowledge proof. In this section we make it explicit, and prove (in Section 3.8) that it provides completeness, knowledge soundness and special honest-verifier zero-knowledge properties. For the introduction of notation, refer to Chapter 2. In our construction the prover has a commitment, $A = h^{\alpha}\boldsymbol{g}^{\boldsymbol{a}}\boldsymbol{h}^{\boldsymbol{b}}$, of two vectors $\boldsymbol{a}, \boldsymbol{b}$ with blinding factor α, and a second commitment, $V = g^{ic}h^{\gamma}$, of a value, ic, with blinding factor γ. The prover convinces the verifier that $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = ic$ holds. In particular, the prover proves the following statement:

$$SPK\{(\alpha, \gamma, \boldsymbol{a}, \boldsymbol{b}, ic) \colon A = h^{\alpha}\boldsymbol{g}^{\boldsymbol{a}}\boldsymbol{h}^{\boldsymbol{b}} \ \wedge \ V = g^{ic}h^{\gamma} \ \wedge \ \langle \boldsymbol{a}, \boldsymbol{b} \rangle = ic\}$$

The prover, $\mathcal{P}$, and verifier, $\mathcal{V}$, interact as follows[7]:

$\mathcal{P}$: It computes blinding vectors, $\boldsymbol{s}_L, \boldsymbol{s}_R$ for each vector of the inner product and commits to it:

$$\boldsymbol{s}_L, \boldsymbol{s}_R \leftarrow_{\$} \mathbb{Z}_p^N, \tag{3.2}$$

$$\rho \leftarrow_{\$} \mathbb{Z}_p, \tag{3.3}$$

$$S = h^{\rho}\boldsymbol{g}^{\boldsymbol{s}_L}\boldsymbol{h}^{\boldsymbol{s}_R} \in G. \tag{3.4}$$

Now the prover defines two linear vector polynomials, $l(X), r(X) \in \mathbb{Z}_p^N[X]$, where X is the indeterminate, and a quadratic polynomial as follows:

$$l(X) = \boldsymbol{a} + \boldsymbol{s}_L \cdot X \in \mathbb{Z}_p^N[X], \tag{3.5}$$

$$r(X) = \boldsymbol{b} + \boldsymbol{s}_R \cdot X \in \mathbb{Z}_p^N[X], \tag{3.6}$$

$$t(X) = \langle l(X), r(X) \rangle = t_0 + t_1 \cdot X + t_2 \cdot X^2 \in \mathbb{Z}_p^N[X]. \tag{3.7}$$

By creating like that the polynomials, it allows for an evaluation of the polynomial at a given point without disclosing any information about the vectors $\boldsymbol{a}$ or $\boldsymbol{b}$. The prover needs to convince the verifier that the constant term of $t(X)$ equals ic. To do so, the prover commits to the remaining coefficients of the polynomial

$$\tau_1, \tau_2 \leftarrow_{\$} \mathbb{Z}_p, \tag{3.8}$$

$$T_i = g^{t_i}h^{\tau_i} \in G, i = \{1, 2\}. \tag{3.9}$$

[7]Note that this protocol can be made non-interactive with the Fiat-Shamir Heuristic [FS87].

$\mathcal{P} \to \mathcal{V}$**:** S, T_1, T_2

$\mathcal{V}$**:** The verifier then computes a challenge

$$c \leftarrow_{\$} \mathbb{Z}_p^*.$$

$\mathcal{V} \to \mathcal{P}$**:** c

$\mathcal{P}$**:** It computes the response using the challenge received

$$\boldsymbol{l} = l(c) = \boldsymbol{a} + \boldsymbol{s}_L \cdot c \in \mathbb{Z}_p^N, \tag{3.10}$$
$$\boldsymbol{r} = r(c) = \boldsymbol{b} + \boldsymbol{s}_R \cdot c \in \mathbb{Z}_p^N, \tag{3.11}$$
$$\hat{t} = \langle \boldsymbol{l}, \boldsymbol{r} \rangle \in \mathbb{Z}_p, \tag{3.12}$$
$$\tau_c = \tau_2 \cdot c^2 + \tau_1 \cdot c + \gamma, \tag{3.13}$$
$$\mu = \alpha + \rho \cdot c \in \mathbb{Z}_p. \tag{3.14}$$

$\mathcal{P} \to \mathcal{V}$**:** $\tau_c, \mu, \hat{t}, \boldsymbol{r}, \boldsymbol{l}$.

$\mathcal{V}$**:** The verifier needs to check that $\boldsymbol{r}, \boldsymbol{l}$ are correct and the inner product relation holds with respect to $\hat{t}$. To this end it performs the following checks:

$$g^{\hat{t}} h^{\tau_c} \stackrel{?}{=} V \cdot T_1^c \cdot T_2^{c^2}, \tag{3.15}$$
$$P = A \cdot S^c \in G, \tag{3.16}$$
$$P \stackrel{?}{=} h^{\mu} \cdot \boldsymbol{g}^{\boldsymbol{l}} \cdot \boldsymbol{h}^{\boldsymbol{r}}, \tag{3.17}$$
$$\hat{t} \stackrel{?}{=} \langle \boldsymbol{l}, \boldsymbol{r} \rangle. \tag{3.18}$$

If all checks validate, then this means that the statement is true with overwhelming high probability.

To make this proof logarithmic, we use the same trick as in the original paper. Instead of sending the vectors $\boldsymbol{r}, \boldsymbol{l}$ to prove the inner product relation, we leverage IP-ZKP over the blinded vectors recursively.

We present the following result:

Theorem 1 ([Que+21])**.** The inner product proof presented in Protocol 1 has perfect completeness, perfect special honest verifier zero-knowledge, and knowledge soundness.

Proof. This theorem is simply an instantiation of Theorem 4 (which proves the above statement for batch proofs) with $m = 1$. □

3.4 Scheme

Now that we have a sub-linear zero-knowledge inner product proof, and by leveraging the proofs introduced in Section 2.2, we can design our ad-hoc ZKP of SVM evaluation. Without loss of generality, we assume that the number of sensor inputs is n for every sensor.

Protocol 3.4.1 (zkSVM)**.** The protocol is divided in three phases. First, the setup phase, $\texttt{Setup}(\epsilon)$, where the server generates the model and their corresponding weights. Secondly, the proving phase, where the prover fetches the SVM related data, computes the difference vector, and proceeds to provably compute the average and standard deviation of these values. It then applies the corresponding linear combinations to the hidden features, and opens the result to send it to the verifier. Finally, the verification phase, where the verifier checks that all computations were correctly performed. Then, it checks whether the scores do correspond to a human or a bot.

Setup phase
The setup involves the server, where it generates the cryptographic material, and trains the SVM model. The details on how this model is generated falls out of the scope of zkSVM, the only requirement is that it follows the specifications described at the beginning of this section. Once these values are computed, they are published to allow provers to fetch them and generate the proofs.

> **Procedure 3.4.1.** (Setup) On input the security parameter, ϵ, the zkSVM server runs $\texttt{Setup}(\epsilon)$ to generate the SVM and cryptographic parameters. Precisely, it trains the SVM model and generates the normalisation mean, normalisation scale, and SVM weight and intercept, M_i, S_i, w_i and c respectively. It also considers the size of the input vectors, n, which defines how long a touch is considered and the measurement frequency. Next it selects the group, G, with generators g and h, and prime order p. It proceeds by computing two vectors, $\boldsymbol{g}, \boldsymbol{h}$, of generators that act as bases for the Pedersen Vector Commitments. Note that the corresponding discrete log of these bases must remain unknown.

Proving phase
The proof generation is divided in five protocols. First the prover computes the difference vectors and proves correctness. Next it computes the average of all vectors, followed by a computation of the standard deviation. Finally, it evaluates the normalising linear computations over the results, and sends the opening of the result to the verifier together with the proofs of correctness.

The prover's secret is $\mathbf{v} \in \mathbb{Z}_p^N, r \in \mathbb{Z}_p$ such that

$$S_H = \boldsymbol{g}^{\mathbf{v}} \cdot h^r.$$

Procedure 3.4.2. (Consecutive difference) In this step the prover's goal is to compute the difference of consecutive values in the input vector, while keeping it hidden. Mainly, we want a provable value of

$$S_H^d = \boldsymbol{g}^{\mathbf{v}_d} h^{r_d},$$

with $\mathbf{v}_d = [\mathbf{v}_1 - \mathbf{v}_2, \mathbf{v}_2 - \mathbf{v}_3, \ldots, \mathbf{v}_{n-1} - \mathbf{v}_n, 0]$, and $r_d \in \mathbb{Z}_p$, indistinguishable from random. The intuition here is first to get a commitment of the iterated values of the sensor vector, then leverage the homomorphic property to subtract this commitment with S_H and finally provably replace the value in position n by zero. To compute the iterated value, the prover first iterates the base generators, to get

$$\boldsymbol{g}_{iter} = [g_n, g_1, \ldots, g_{n-1}].$$

Note that this step can be performed by the verifier as the generators are public. It then commits the sensor vector with this base

$$S_H^{iter} = \boldsymbol{g}_{iter}^{\mathbf{v}} \cdot h^{r_{iter}},$$

with $r_{iter} \leftarrow_{\$} \mathbb{Z}_p$, and generates a proof of equality,

$$\Pi_{Eq} = \Pi_{Eq}.\mathtt{Gen}(\boldsymbol{g}, \boldsymbol{g}_{iter}, h, S_H, S_H^{iter}; \mathbf{v}, r, r_{iter}).$$

Note that

$$S_H^{iter} = g_1^{\mathbf{v}_2} \cdots g_{n-1}^{\mathbf{v}_n} \cdot g_n^{\mathbf{v}_1} \cdot h^{r_{iter}},$$

so now we can simply subtract the two commitments to get

$$\overline{S_H} = S_H / S_H^{iter} = g_1^{v_1 - v_2} g_2^{v_2 - v_3} \cdots g_{n-1}^{v_{n-1} - v_n} g_n^{v_n - v_1} \cdot h^{r - r_{iter}}.$$

Note that r and r_{iter} are random, and therefore, so is $r - r_{iter}$. Finally, the prover replaces the value in the exponent of g_n by a zero, to get the final commitment:

$$Diff = g_1^{v_1 - v_2} g_2^{v_2 - v_3} \cdots g_{n-1}^{v_{n-1} - v_n} g_n^0 \cdot h^{r_{diff}}, \tag{3.19}$$

and generates a proof of correctness,

$$\Pi_0 = \Pi_0.\mathtt{Gen}(\boldsymbol{g}, h, \overline{S_H}, Diff; \mathbf{v}_d, r - r_{iter}, r_{diff}). \tag{3.20}$$

It stores $\Delta = [Diff, S_H^{iter}, \Pi_{Eq}, \Pi_0]$.

Procedure 3.4.3. (Sum of vectors) The prover now computes $\tilde{\mu} = N \cdot \mu$, mainly, the sum of all values. To provably compute this, we leverage IP-ZKP between the initial commitment, S_H, and a Pedersen commitment with base $\boldsymbol{h}$ of the one vector, $\boldsymbol{h^1}$, to prove that a third commitment, $Avg = \texttt{Comm}(\tilde{\mu}, r_\mu)$, commits to the sum of the committed values in S_H,

$$\langle \mathbf{v}, \mathbf{1} \rangle = \tilde{\mu}.$$

The user proves correctness of the commitment,

$$\Pi^{\mu}_{IP} = \Pi_{IP}.\texttt{Gen}(\boldsymbol{g}, \boldsymbol{h}, g, h, S_H \cdot \boldsymbol{h^1}, Avg; \mathbf{v}, r).$$

It stores both values $\mathrm{M} = [Avg, \Pi^{\mu}_{IP}]$. It repeats the same steps as above with the commitment of the consecutive difference vector, resulting in a commitment of the average, Avg', and a proof of correctness, $\Pi^{\mu}_{IP}{}'$. It stores both values $\mathrm{M}' = [Avg', \Pi^{\mu}_{IP}{}']$.

Procedure 3.4.4. (Standard deviation) To calculate a factor of the standard deviation, σ, we first compute the variance, σ^2. Recall that

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (v_i - \mu)^2,$$

or written differently

$$\sigma^2 = \frac{1}{N} \langle \mathbf{v} - \boldsymbol{\mu}, \mathbf{v} - \boldsymbol{\mu} \rangle,$$

where $\boldsymbol{\mu}$ is a vector with μ in all its positions. For this we need the average, but only have a provable commitment of the sum, $\tilde{\mu}$. Hence, instead of computing the variance, we compute $N^3 \cdot \sigma^2$ by leveraging the inner product proof and the arithmetic properties of the commitment function. The intuition is the following: if we multiply each entry of $\mathbf{v}$ by N, we can get the following relation.

$$\begin{aligned} &\langle N \cdot \mathbf{v} - \tilde{\boldsymbol{\mu}}, N \cdot \mathbf{v} - \tilde{\boldsymbol{\mu}} \rangle = \langle N \cdot \mathbf{v} - N \cdot \boldsymbol{\mu}, N \cdot \mathbf{v} - N \cdot \boldsymbol{\mu} \rangle = \\ &\langle N \cdot (\mathbf{v} - \boldsymbol{\mu}), N \cdot (\mathbf{v} - \boldsymbol{\mu}) \rangle = N^2 \langle \mathbf{v} - \boldsymbol{\mu}, \mathbf{v} - \boldsymbol{\mu} \rangle = N^3 \cdot \sigma^2. \end{aligned} \quad (3.21)$$

However, we only have S_H, and a provable commitment of $\tilde{\mu}$ (not of $\tilde{\boldsymbol{\mu}}$). Moreover, we need a commitment of $\mathbf{v}$ and $\tilde{\boldsymbol{\mu}}$ under both bases ($\boldsymbol{g}$ and $\boldsymbol{h}$). To this end, the prover computes the following steps.

First, it computes the commitment of $\tilde{\boldsymbol{\mu}}$ with both bases. To this end, the prover first computes a product of all the bases,

$$g_\Pi = \prod_{i=1}^{n} g_i \quad \text{and} \quad h_\Pi = \prod_{i=1}^{n} h_i.$$

Note that this step is again reproducible by the verifier, and hence no proof is required. Next it commits the average using these products as a base, to get

$$G^{\tilde{\mu}} = g_\Pi^{\tilde{\mu}} \cdot h^{r_G} = g_1^{\tilde{\mu}} \cdots g_n^{\tilde{\mu}} \cdot h^{r_G},$$

and

$$H^{\tilde{\mu}} = h_\Pi^{\tilde{\mu}} \cdot h^{r_H} = h_1^{\tilde{\mu}} \cdots h_n^{\tilde{\mu}} \cdot h^{r_H},$$

with $r_G, r_H \leftarrow_\$ \mathbb{Z}_p$. It proves equality between the opening of $Avg, G^{\tilde{\mu}}$ and $H^{\tilde{\mu}}$ using Π_{Eq}, and stores the two proofs,

$$\Pi_{Eq}^G = \Pi_{Eq}.\texttt{Gen}(g_\Pi, g, h, Avg, G^{\tilde{\mu}}; \mu, r_\mu, r_G),$$

and

$$\Pi_{Eq}^H = \Pi_{Eq}.\texttt{Gen}(h_\Pi, g, h, Avg, H^{\tilde{\mu}}; \mu, r_\mu, r_H),$$

one for each base. Finally, the prover commits to $\mathbf{v}$ with randomness r_S with $\boldsymbol{h}$ as bases,

$$H_S = \boldsymbol{h}^{\mathbf{v}} h^{r_S},$$

and proves equality of opening with respect to S_H, getting

$$\Pi_{Eq}^S = \Pi_{Eq}.\texttt{Gen}(\boldsymbol{g}, \boldsymbol{h}, h, S_H, H_S; \mathbf{v}, r, r_S).$$

This allows the prover to leverage relation (3.21) to provably compute a commitment of a factor of the variance using the proof presented in Protocol 3.3.1 To this end, it computes

$$A_S = S_H^N / G^{\tilde{\mu}} \cdot H_S^N / H^{\tilde{\mu}} = \boldsymbol{g}^{N \cdot \mathbf{v} - \tilde{\boldsymbol{\mu}}} \cdot \boldsymbol{h}^{N \cdot \mathbf{v} - \tilde{\boldsymbol{\mu}}} \cdot h^{N \cdot r - r_G + N \cdot r_S - r_H}$$

and the commitment of the factor of the variance,

$$Var = \texttt{Comm}(N^3 \cdot \sigma^2, r_V).$$

It generates a proof of correctness

$$\Pi_{IP}^{\sigma^2} = \Pi_{IP}.\texttt{Gen}(A_S, Var, \boldsymbol{g}, \boldsymbol{h}, g, h; \mathbf{v}, \tilde{\mu}, r, r_G, r_H, r_S, r_V).$$

Finally, the prover needs to compute the square root of the variance. To this end, it commits to the floor of the square root of $N^3 \cdot \sigma^2$, $Std = \texttt{Commit}(\lfloor\sqrt{N^3 \cdot \sigma^2}\rfloor, r_{\surd})$, with randomness $r_{\surd}$. Then, the prover leverages the square root proof introduced in Section 2.2.3, and generates

$$\Pi^{\sigma}_{sqrt} = \Pi_{sqrt}.\texttt{Gen}(Var, Std, g, h; \sigma^2, \sigma, r_V, r_{\surd}).$$

This results in a provable commitment of a factor of the floor of the standard deviation, Std. The prover stores,

$$\Lambda = \left[G^{\tilde{\mu}}, H^{\tilde{\mu}}, H_S, \Pi^G_{Eq}, \Pi^H_{Eq}, \Pi^S_{Eq}, Var, \Pi^{\sigma^2}_{IP}, Std, \Pi^{\sigma}_{sqrt}\right].$$

It repeats the same steps above with the consecutive difference vector (and average), resulting in

$$\Lambda' = \left[G^{\tilde{\mu}\,\prime}, H^{\tilde{\mu}\,\prime}, H_S{}', \Pi^G_{Eq}{}', \Pi^H_{Eq}{}', \Pi^S_{Eq}{}', Var', \Pi^{\sigma^2}_{IP}{}', Std', \Pi^{\sigma}_{sqrt}{}'\right].$$

Procedure 3.4.5. (Computing SVM score) Provably computing SVM score, equation (3.1), reduced to provably computing $\sum_{i=1}^{N} f_i \left\lfloor \frac{w_i}{S_i} 10^d \right\rfloor$ where f_i are the features. However, note that we do not have the features themselves, but a factor of them. Hence, with this scheme, we need to compute instead $r = \sum_{i=1}^{N} f_i \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor$ where N_i equals N if f_i is an average, and $N^{3/2}$ if it is a standard deviation (note that we have different factors of each). Again, $\frac{w_i}{N_i \cdot S_i}$ is public. Let $\texttt{Comm}_i = \texttt{Commit}(f_i, r_i)$ be the commitment of feature f_i with blinding factor r_i. The prover computes the following:

$$\begin{aligned} Res = \prod_{i=1}^{N} \texttt{Comm}_i^{\left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor} = \\ \prod_{i=1}^{N} \texttt{Commit}\left(f_i \cdot \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor, r_i \cdot \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor\right) = \\ \texttt{Commit}\left(\sum_{i=1}^{N} f_i \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor, r_R\right), \end{aligned} \quad (3.22)$$

where $r_R = \sum_{i=1}^{N} r_i \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor$, is the blinding factor known to the prover. Once these operations have been performed, the prover stores the opening of the commitment, $Score = \sum_{i=1}^{N} f_i \left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor$, and the blinding factor r_R.

Procedure 3.4.6. (Sending values) The prover sends to the verifier the following tuple

$$[S_H, \Delta, \mathrm{M}, \mathrm{M}', \Lambda, \Lambda', Score, r_R]\,.$$

Verification phase
The verifier then performs all respective linear combinations commitments, and verifies the zero-knowledge proofs. If any proof fails or the evaluation of the model over *Score* fails, the verifier denies the request. Else, it accepts it. More precisely, the verifier follows the following procedures:

Procedure 3.4.7. (Verifying consecutive difference) The verifier begins by iterating the base of generators, to get

$$\boldsymbol{g}_{iter} = [g_n, g_1, \ldots, g_{n-1}],$$

and then verifies the proof of opening equality,

$$\Pi_{Eq}.\texttt{Verif}(\boldsymbol{g}, \boldsymbol{g}_{iter}, h, S_H, S_H^{iter}) \stackrel{?}{=} \top.$$

Next it computes the subtraction commitment to get

$$\overline{S_H} = S_H / S_H^{iter}.$$

Finally, it verifies that the proof

$$\Pi_0.\texttt{Verif}(\boldsymbol{g}, h, \overline{S_H}, Diff) \stackrel{?}{=} \top.$$

Procedure 3.4.8. (Verifying sum of vectors) The verifier checks the inner product proof

$$\Pi_{IP}^{\mu}.\texttt{Verify}(\boldsymbol{g}, \boldsymbol{h}, g, h, S_H \cdot \boldsymbol{h^1}, Avg) \stackrel{?}{=} \top.$$

It repeats this step for the consecutive difference commitment and proof.

Procedure 3.4.9. (Verifying standard deviation) The verifier first computes a product of all the bases,

$$g_\Pi = \prod_{i=1}^{n} g_i \quad \text{and} \quad h_\Pi = \prod_{i=1}^{n} h_i.$$

Next it verifies the proofs of equality of commitments using these bases

$$\Pi^{G}_{Eq}.\texttt{Verify}(g_{\Pi}, g, h, Avg, G^{\tilde{\mu}}) \stackrel{?}{=} \top,$$

and

$$\Pi^{H}_{Eq}.\texttt{Verify}(h_{\Pi}, g, h, Avg, H^{\tilde{\mu}}) \stackrel{?}{=} \top.$$

Next, the verifier checks that H_S commits to the input vector

$$\Pi^{S}_{Eq}.\texttt{Verify}(\boldsymbol{g}, \boldsymbol{h}, h, S_H, H_S) \stackrel{?}{=} \top.$$

Now the verifier needs to generate the commitments under which the inner product proof of the variance will verify against. To this end it computes

$$L = S_H / G^{\tilde{\mu}} \text{ and } R = H_S / H^{\tilde{\mu}} \tag{3.23}$$

and uses them to verify the inner product proof

$$\Pi^{\sigma^2}_{IP}.\texttt{Verify}(L \cdot R, Var, \boldsymbol{g}, \boldsymbol{h}, g, h) \stackrel{?}{=} \top.$$

Finally, the verifier checks the correctness of the factor of the standard deviation commitment

$$\Pi^{\sigma}_{sqrt}.\texttt{Verify}(Var, Std, g, h) \stackrel{?}{=} \top.$$

It repeats these steps for the consecutive difference commitments.

Procedure 3.4.10. (Computing SVM score) Finally, the verifier computes the same linear combinations as the prover,

$$Res' = \prod_{i=1}^{N} \texttt{Comm}_i^{\left\lfloor \frac{w_i}{N_i \cdot S_i} 10^d \right\rfloor}, \tag{3.24}$$

and checks the validity of the received opening,

$$o(Res', Score, r_R, g, h) \stackrel{?}{=} \top.$$

It uses this value to compute the final score as described in equation (3.1). If any of the checks fail or the score determines the user is a bot, it returns $\perp$, otherwise it returns $\top$.

In a nutshell, by extending the inner product proof presented in [Bün+18] to a zero-knowledge proof and leveraging the arithmetic properties of Pedersen commitments, we build zkSVM, a privacy-preserving SVM evaluation model.

3.5 Security analysis

In this section we formally define the properties we expect out of zkSVM, namely privacy and verifiability. We use game based proofs to show that zkSVM indeed provides these properties. We formally define them here, and include the proofs subsequently. To model the experiments of privacy and verifiability, we define the following five functions:

- `Setup`(ϵ): Which is defined exactly as in Procedure 3.4.1. We omit the notation of the cryptographic material, and consider it implicit. We represent the set of parameters of the SVM model (normalisation mean, normalisation scale, SVM weight and SVM intercept) by W.
- `GenProof`($\mathbf{v}$): Generates the zkSVM proof by running Procedures 3.4.2, 3.4.3, 3.4.4, and 3.4.5. Mainly, it runs all steps of the proof except for the submission step. We simplify the representation of the resulting tuple by $[S_H, \Theta, Score, r_R]$, where Θ consists of all intermediate proofs and commitments.
- `SubmitReq`($[S_H, \Theta, Score, r_R]$): Submits the output of `GenProof`($\mathbf{v}$) by sending it to the verifier (Procedure 3.4.6).
- `VerifReq`($[S_H, \Theta, Score, r_R]$): Runs all procedures defined in the Verification phase of zkSVM, mainly Procedures 3.4.7, 3.4.8, 3.4.9 and 3.4.10.
- `EvalSVM`($\mathbf{v}$): Generates the result, $Score$, corresponding to $\mathbf{v}$, as defined in the zkSVM proof, but excluding the cryptographic mechanisms.

Privacy

The goal of zkSVM is that no information is leaked from the input vector other than the result of the SVM model. To model this, in Figure 3.4 we define an experiment, $\texttt{Exp}^{\texttt{priv},b}_{\mathcal{A},\mathcal{D}}$, between an adversary, $\mathcal{A}$, and a challenger, $\mathcal{D}$. The latter chooses a bit $b \in \{0, 1\}$, uniformly at random, which is given as input to the experiment. The adversary controls the zkSVM verifier, and its goal is to distinguish the submissions of two different input vectors. The adversary is given access to an oracle, O`Submit`(), which takes as input two vectors of size n, runs `GenProof`() over them, and submits a result. Note that it is the adversary who chooses the vectors over which the zkSVM is executed and may modify the weights of the SVM model outputted by `Setup`(). Therefore, this experiment models the malicious choice of the SVM parameters, as well as any possible choice of input vector. Depending on the bit, b, the oracle submits the result of one vector or the other, by running `SubmitReq`().

$\texttt{Exp}^{\texttt{priv},b}_{\mathcal{A},\mathcal{D}}(\epsilon)$:
  W$\leftarrow$ $\texttt{Setup}(\epsilon)$
  $b' \leftarrow \mathcal{A}^{O\texttt{Submit}}(W)$
  Output b'

$O\texttt{Submit}(v_1, v_2)$:
  Let $[S^1_H, \Theta^1, Score^1, r^1_R] \leftarrow \texttt{GenProof}(v_1, W)$
  Let $[S^2_H, \Theta^2, Score^2, r^2_R] \leftarrow \texttt{GenProof}(v_2, W)$
  Let $\Theta^2 \leftarrow \texttt{SimResult}(S^2_H, Score^1, r^1_R)$
  return $\texttt{SubmitReq}([S^b_H, \Theta^b, Score^1, r^1_R])$

Figure 3.4: In the privacy experiment $\texttt{Exp}^{\texttt{priv},b}_{\mathcal{A},\mathcal{D}}$, the adversary $\mathcal{A}$ has access to the oracle $O\texttt{Submit}$ and controls zkSVM verifier.

However, to avoid a trivial win by the adversary, the submitted SVM score is always computed over the first vector. Hence, the experiment simulates the proof of the second vector by running $\texttt{SimResult}()$. The adversary may call this oracle as many times as it wishes. By the end of the experiment, the adversary outputs a bit, $b' \in \{0, 1\}$. The adversary wins if $b' = b$ with non-negligible probability with respect to the security parameter, ϵ.

Theorem 2. There exists a $\texttt{SimResult}$ algorithm, such that no PPT adversary can win the zkSVM privacy experiment, $\texttt{Exp}^{\texttt{priv},b}_{\mathcal{A},\mathcal{D}}$, with colluding verifier with probability non-negligibly better than $1/2$ with respect to ϵ.

Proof. To prove that zkSVM provides privacy we proceed by a series of games. We start with the adversary playing the privacy experiment with $b = 0$, and after a sequence of game step transitions, the adversary finishes playing the ballot privacy experiment with $b = 1$. We argue that each of these steps are indistinguishable, and therefore the results follows. The proof proceeds along the following sequence of games:

Game $\mathbf{G}_0$: Let game $\mathbf{G}_0$ be the $\texttt{Exp}^{\texttt{priv},0}_{\mathcal{A},\mathcal{D}}(\epsilon)$ game (see Figure 3.4).

Game $\mathbf{G}_1$: Game $\mathbf{G}_1$ is as in $\mathbf{G}_0$, but now $O\texttt{Submit}$ always computes a simulation of Θ regardless of the bit. Mainly, $O\texttt{Submit}$ proceeds as follows:

$O\texttt{Submit}(v_1, v_2)$:
  Let $[S^1_H, \Theta^1, Score^1, r^1_R] \leftarrow \texttt{GenProof}(v_1, W)$
  Let $[S^2_H, \Theta^2, Score^2, r^2_R] \leftarrow \texttt{GenProof}(v_2, W)$
  Let $\Theta^1 \leftarrow \texttt{SimResult}(S^1_H, Score^1, r^1_R)$
  Let $\Theta^2 \leftarrow \texttt{SimResult}(S^2_H, Score^1, r^1_R)$
  return $\texttt{SubmitReq}([S^1_H, \Theta^1, Score^1, r^1_R])$

The function `SimResult` proceeds by simulating all zero-knowledge proofs contained in Θ. Because all these proofs are zero-knowledge proofs, and hence have the special honest-verifier zero-knowledge property (see Section 2.2), there exists a simulation algorithm such that $\mathcal{A}$ cannot distinguish between a real and a simulated proof. Note that at this point of the experiment, the commitment S_H and all commitments in Θ correspond to those of v_1 —only the zero-knowledge proofs in Θ are simulated.

Game $\mathbf{G}_2$: Game $\mathbf{G}_2$ is as in $\mathbf{G}_1$, but now, instead of returning

$$\texttt{SubmitReq}([S_H^1, \Theta^1, Score^1, r_R^1]),$$

the oracle $O\texttt{Submit}$ returns

$$\texttt{SubmitReq}([S_H^2, \Theta^2, Score^1, r_R^1]).$$

In $\mathbf{G}_2$ the view of the adversary is identical to the one of $\texttt{Exp}^{\texttt{priv},1}_{\mathcal{A},\mathcal{D}}(\epsilon)$. Only thing that remains is to prove that $\mathbf{G}_1$ and $\mathbf{G}_2$ are indistinguishable

Given the Special Honest-Verifier Zero-Knowledge property of the proofs, we know that the simulated view is random. Given that both simulations are equally distributed, it is infeasible to distinguish between Θ^1 and Θ^2. Similarly, given the perfectly hiding property of Pedersen commitments, no adversary can distinguish between S_H^1 and S_H^2.

Clearly the resulting view is independent of b. And privacy follows. □

Verifiability

The other goal of zkSVM is that an adversary cannot convince a verifier that the result is not linked to the committed vector as defined by the protocol. To model this, in Figure 3.5 we define an experiment, $\texttt{Exp}^{\texttt{verif}}_{\mathcal{A},\mathcal{D}}$, between an adversary, $\mathcal{A}$, and a challenger, $\mathcal{D}$. Informally, verifiability ensures that a result that is not the outcome of the model evaluation over the committed vector cannot have a valid proof. Note that this property does not ensure that the sensor data has not been used before nor that it comes from the sensors of the device. It is out of the scope of this security analysis to ensure that the mobile device is not rooted and that the adversary cannot tamper the sensor data. During the experiment the adversary has access to the weights of the model, and generates an input vector, a result, and its corresponding proof material. The adversary wins the experiment if the result does not correspond the the SVM execution of the committed vector, and the verifier validates. We formally describe the experiment in Figure 3.5.

$\texttt{Exp}^{\texttt{verif}}_{\mathcal{A},\mathcal{D}}(\epsilon)$:
W$\leftarrow \texttt{Setup}(\epsilon)$
$[\mathbf{v}, S_H, \Theta, Score, r_R] \leftarrow \mathcal{A}(W)$
If $\texttt{EvalSVM}(\mathbf{v}) = Score$ return 0
If $\texttt{VerifReq}([S_H, \Theta, Score, r_R]) = \perp$
then return 0, else return 1.

Figure 3.5: In the verifiability experiment $\texttt{Exp}^{\texttt{verif}}_{b,\mathcal{A}}$, the adversary $\mathcal{A}$ needs to submit a result not corresponding to the commited vector.

Theorem 3. No PPT adversary can win the zkSVM verifiability experiment, $\texttt{Exp}^{\texttt{verif}}_{\mathcal{A},\mathcal{D}}$, with non-negligible probability with respect to ϵ.

Proof. At the end of the proof generation procedure, the prover (in our case the adversary) outputs a commitment, S_H, a tuple of intermediate cryptographic material, Θ, and a score, $Score$, together with the randomness associated to the commitment of the score, r_R. The result follows from the soundness property of ZKPs and the binding property of Pedersen commitments. Let us extend the cryptographic material associated with the vector. We have that $\Theta = [\Delta, \mathrm{M}, \mathrm{M}', \Lambda, \Lambda']$, with

$$\begin{aligned}
\Delta &= \left[Diff, S_H^{iter}, \Pi_{Eq}, \Pi_0\right], \\
\mathrm{M} &= \left[Avg, \Pi_{IP}^{\mu}\right], \\
\mathrm{M}' &= \left[Avg', \Pi_{IP}^{\mu}{}'\right], \\
\Lambda &= \left[G^{\tilde{\mu}}, H^{\tilde{\mu}}, H_S, \Pi_{Eq}^{G}, \Pi_{Eq}^{H}, \Pi_{Eq}^{S}, Var, \Pi_{IP}^{\sigma^2}, Std, \Pi_{sqrt}^{\sigma}\right], \\
\Lambda' &= \left[G^{\tilde{\mu}\,\prime}, H^{\tilde{\mu}\,\prime}, H_S{}', \Pi_{Eq}^{G}{}', \Pi_{Eq}^{H}{}', \Pi_{Eq}^{S}{}', Var', \Pi_{IP}^{\sigma^2}{}', Std', \Pi_{sqrt}^{\sigma}{}'\right].
\end{aligned}$$

The proof verifies all proofs. In particular

- Procedure 3.4.7 first generates the iterated generators, and checks that indeed S_H and S_H^{iter} commit to the same opening, by verifying Π_{Eq}. Then, it checks that $Diff$ commits to the difference of S_H and S_H^{iter}, in all entries but the last, which contains a zero, by verifying Π_0.

- Procedure 3.4.8 checks that Avg commits to the sum of the elements committed in S_H by verifying Π_{IP}^{μ}. It does the same for the difference vector.

- Procedure 3.4.9 first checks that $G^{\tilde{\mu}}$ and $H^{\tilde{\mu}}$ commit to the sum committed in Avg by verifying Π_{Eq}^{G} and Π_{Eq}^{H} respectively. Then, it checks that H_S commits to the same values as S_H by verifying Π_{Eq}^{S}. With

these verified commitments, the verifier can check that Var commits to $N^2\langle \mathbf{v} - \boldsymbol{\mu}, \mathbf{v} - \boldsymbol{\mu} \rangle$, in other words, that it commits to a factor of the variance. It does so by running the algebraic operations of equation (3.23), and verifying $\Pi^{\sigma^2}_{IP}$. Finally, to check that Std commits to the standard deviation of the vector committed in S_H, it simply needs to check that it contains the square root of Var. It does so by verifying Π^{σ}_{sqrt}. The same is performed for the difference vector.

Given that all these proofs are Zero Knowledge Proofs, which provide the knowledge soundness property, the commitments of the SVM features, Avg, Avg', Std, Std', do not contain the expected features of the vector committed in S_H with negligible probability. It only remains to prove that the result indeed corresponds to the SVM function executed with these features as inputs.

- Procedure 3.4.10 first leverages the homomorphic properties of the commitment scheme. In this way, the verifier obtains a commitment of the linear combination of the values committed in Avg, Avg', Std, Std'. This results in the commitment of the result, as described in equation (3.22). This operation is conducted solely by the verifier, avoiding any possible attacks by the adversary. Finally, the verifier checks that the submitted score, and the corresponding opening key, indeed correspond to the locally computed commitment. Only if this is true, the verifier validates.

Given that the commitment scheme used in zkSVM is computationally binding, a PPT adversary has no more than negligible probability of submitting an opening that does not correspond to the committed value.

Given that `EvalSVM` and the proof of zkSVM compute the exact same operations over the input vector, the result that the adversary has no more than negligible probability of wining the verifiability experiment follows.

□

3.6 System Implementation

To assess the feasibility and effectiveness of our approach we developed (i) an open source library of zkSVM, and (ii) a prototype Android SDK of zkSENSE.

3.6.1 Enclosing SVM Result in a ZKP

The zkSVM library: In order to prove the integrity of the classification results, we developed an open-sourced Rust library that implements the logic presented in Protocol 3.4.1, on enclosing SVM results in zero-knowledge proofs. To this end, we additionally implemented the Pedersen Commitment ZKPs as described in Section 2.2.3. For the proofs $\Pi_{Sq}, \Pi_{Eq}, \Pi_0$ and Π_{Op}, we based our implementation in the work presented in [Gro09]. We used the range proof presented in [Bün+18] and implemented in [VYA20]. Finally, for Π_{IP}, we implemented the zero-knowledge proof presented in Protocol 3.3.1 All the above proofs were implemented using the ristretto255 prime order group over Curve25519 by leveraging the curve25519-dalek [LV20] library. To integrate this library with our detection engine, we used the Android NDK development kit.

General-purpose zkSNARK: To compare the performance of zkSVM with the ZK-SNARK based solution, we implement the SVM execution using the ZoKrates general-purpose zkSNARK toolbox [ET18]. ZoKrates works as a high-level abstraction for the encoding of the computation to be proved into a zkSNARK. ZoKrates constructs the ZKP by using the Rust implementation of Bellman's [Gri16] Groth16 zkSNARK [Gro16]. This construction has optimal proof size and verification time. However, this comes by trading off prover's computational complexity and the requirement of a trusted setup.

To test our scheme in ZoKrates, we test the performance both with the SHA and the Pedersen hash ZoKrates standard library implementations[8] and conclude that the former is more efficient (only SHA and Pedersen where available at the time of testing). ZoKrates only supports static arrays, i.e. their size needs to be known at compilation time. Because of that, we defined the maximum array size to be 70 which was an upper bound for the size of most of the touch events as defined in Section 3.1.1. We padded with zeros the events shorter than 70 sensor values. Moreover, the Pedersen standard library implementation offers a 512-bit input version of the hash function. Given that we use a precision of 6 decimal digits (and hence, can represent each integer with 25 bits), we batch 18 sensor outputs per hash. Given that the gyroscope and accelerometer work in a three dimensional space, and the static size of the array is 70, we need to use signatures over 11 different hashes, and hence prove knowledge of pre-image of 11 Pedersen hashes for every sensor. Moreover, the constant size zkSNARK used in ZoKrates requires a trusted setup.

Comparison: zkSVM does not have any of these limitations: (i) it does

[8] `https://github.com/Zokrates/ZoKrates/tree/master/zokrates_stdlib/stdlib/hashes`

not have a trusted setup, (ii) it supports dynamic arrays and (iii) computing Pedersen hashes is not bounded to a maximum. This allows us to compute a single Pedersen hash per sensor.

3.6.2 Prototype of our Approach

We implement a prototype SDK of zkSENSE for Android, which consists of around a thousand lines of code. Our prototype collects the output of the Android's accelerometer and gyroscope during a touch event and, by applying a pre-trained model, it determines if the touch event was performed by a human or not.

For demonstration purposes, we created a demo app with a user interface that shows the output of the detection model and in [Ano19] we provide publicly a video that demonstrates its functionality. In this demo, we test multiple scenarios to showcase the accuracy of our system:

1. When the device is resting on a steady platform and:
   (a) A human is performing clicks.
   (b) Clicks are simulated.
2. When the device is docked on a swing motion device that produce artificial movement.
3. When the device is held in one hand and:
   (a) A human is performing clicks.
   (b) Clicks are simulated.

In each scenario, the app is using our SDK to attest the humanness of a touch-screen event on the mobile device.

After reading the output of the accelerometer and gyroscope sensors, the zkSENSE SDK applies, on the background, our pre-trained model and classifies the origin of the touch-screen event (i.e. performed by a human or not). For the generation of the ZKPs and the model's evaluation, we leverage the library we implemented and described previously, which we call from the mobile device using the sensor data. The pre-trained model, is generated on a server of ours. Apart from generating and distributing the trained model, the server also acts as the external auditor that verifies the validity of the transmitted attestation results.

3.7 Performance Evaluation

In this section, we set out to explore the performance of humanness attestation in zkSENSE. More specifically, we benchmark our Android prototype with respect to the *duration* of its main operations: (i) humanness classification, (ii) Pedersen commitment computation, and (iii) zero-knowledge proof construction. Next, we evaluate general resource utilization metrics: (a) CPU, (b) memory, and (c) battery consumption. Our tests cover the three key operations of a humanness attestation in zkSENSE, and a comparative baseline:

1. The *baseline*, where we run our demo application which uses zkSENSE service (see Section 3.6) and several artificial clicks are generated.

2. The *detection* operation, where sensors input is collected and humanness classification realized on zkSENSE.

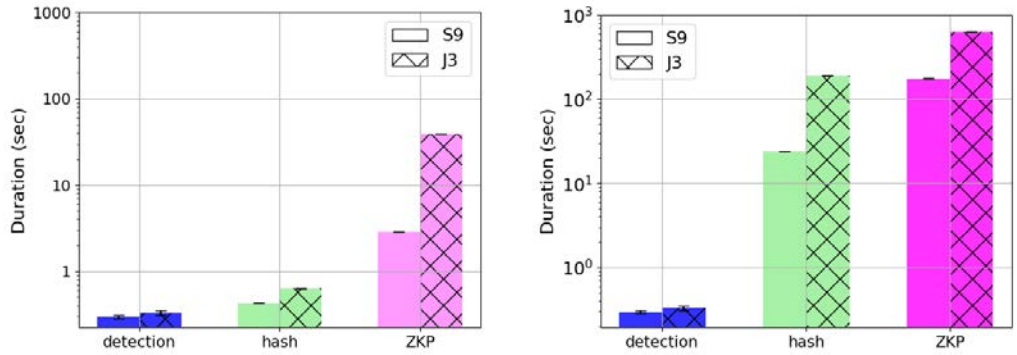
3. The *commitment* operation where the Pedersen commitment computation is taking place.

4. The *ZKP* operation where the proof of correct attestation is constructed.

We test and compare the different implementations described in Section 3.6: (i) the general-purpose zkSNARK[9] and (ii) our proposed zkSVM. We run each stage for an hour and we ensure the same number of artificial clicks by using as an interval the duration of the longest operation (ZKP) as empirically measured on each device under test (Figure 3.6).

Setup We leverage a testbed composed of two Android devices representative of a mid-end (Samsung Galaxy S9, model 2018) and a low-end (Samsung Galaxy J3, model 2016) device, to inspect what is the worst performance a user can get on a cheap (around 90 USD) device. The S9 mounts an octa-core processor (a Quad-Core Mongoose M3 at 2.7GHz and a Quad-Core ARM Cortex-A55 at 1.8Ghz), while the J3 is equipped with a quad-core ARM Cortex A53 at 1.2 Ghz. The S9 also has twice as much memory (4 GB when J3 has 2 GB) and a larger battery (3,000 mAh when the battery of J3 is 2,600 mAh). The low-end device (J3) is part of Batterylab [Bat19; Var+19], a distributed platform for battery measurements. It follows that fine grained battery measurements (via a Monsoon High Voltage Power Monitor [Mon19] directly connected to the device's battery) are available for this device. Automation of the above operations is realized via `adb` run over WiFi to avoid noise in the power measurements caused by USB powering.

[9]We ignore the time elapsed while computing the trusted setup, as this cannot be computed by the client.



(a) ZKP generation, using zkSVM, lasts about 2.9 seconds.

(b) ZKP generation, using the generic zkSNARK solution, lasts about 174.5 seconds

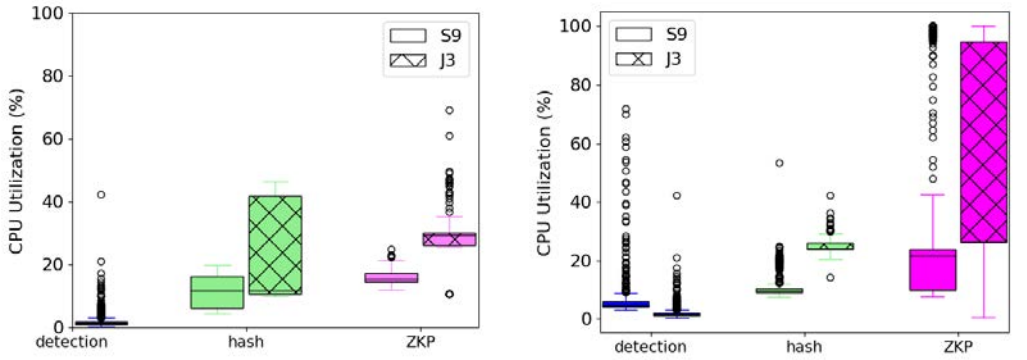
Figure 3.6: Execution time per operation, commented over results on commodity hardware (S9)

Execution time: Figure 3.6a shows the average duration (and standard deviation as error-bars) of each zkSENSE's operation, per device, when considering zkSVM. Regardless of the device, humanness classification and commitments are extremely fast, i.e. about 0.3 and 0.6 seconds even on the less powerful J3. The ZKP generation is instead more challenging, lasting about 2.9 and 39.2 seconds on the S9 and J3, respectively.

Figure 3.6b shows the same results but when considering the ZK-SNARK based solution. In this case, we measure commitment operations comprised between 24 and 190 seconds, and ZKP generation comprised between 175 and 600 seconds, depending on the device. This suggests one order of magnitude speedup of zkSVM versus the more generic zkSNARK solution.

For the verification time, the general purpose zkSNARK (808 nanoseconds) outperforms zkSVM (177 milliseconds). This is expected as the Groth16 approach used by ZoKrates has a big prover overhead and a trusted setup in exchange of minimal communication and verification time overhead. However, in zkSENSE's scenario, these verification times can be handled by the server, and instead, zkSVM makes the prover times reasonable (as shown, the entire attestation takes a bit less than 3 seconds) and removes the need for trusted setup.

CPU and memory utilization: Figures 3.7a and 3.7b show the CPU utilization per operation and device, using zkSVM and the ZK-SNARK based solution respectively. Since no significant difference was observed between baseline and detection operation, we improve the figure visibility by reporting only one boxplot representative of both operations. The figure shows,



(a) Median CPU consumption, using zkSVM, of about 15%.

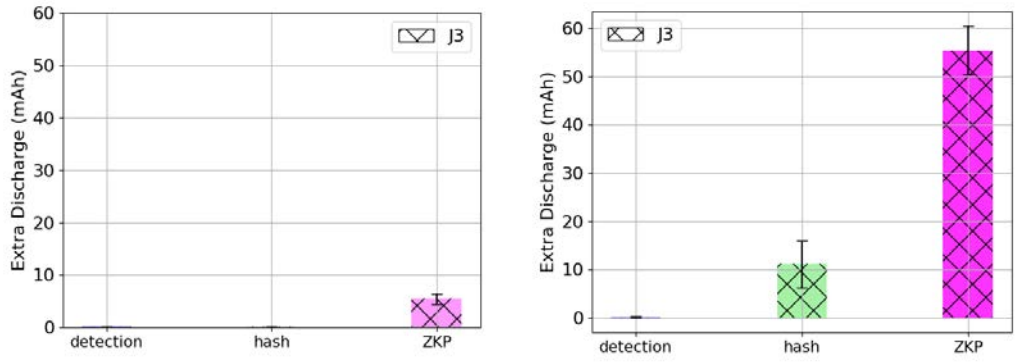
(b) CPU utilization, using zkSNARK, gets higher than 20% for 25% of the operation duration on a S9 device.

Figure 3.7: CPU utilization per operation, commented over results on commodity hardware (S9), over the most expensive operation, the ZKP generation.

overall, minimal CPU utilization associated with humanness classification[10] and commitment operations with zkSVM. Even on the less powerful J3, committing only consumes about 12% of CPU (median value across devices) with peaks up to 45% on the J3. The ZKP generation is the most expensive operation, showing a median CPU consumption of about 15% and 30%, on respectively the S9 and J3. Overall, the CPU analysis suggests minimal impact of zkSENSE's operation and feasibility even on entry-level devices like the J3. On the other hand, we can see that the ZK-SNARK based solution induces a considerable overhead on the user, with CPU peaks of about 30-40% for the bot detection and hashing operations. The ZKP generation is by far the most expensive operation. While easy to manage for the S9 device (e.g. CPU utilization higher than 20% for only 25% of the operation duration), it can be challenging for an entry level device like the J3 for which the plot shows high CPU utilization (90% of higher).

During our tests, we also collected memory usage of zkSENSE's as reported by `procstats`. Detailed results are omitted since zkSENSE's memory consumption is negligible, i.e. less than 20MB regardless of device and operation. In comparison, the zkSNARK solution requires up to 1GB of memory due to the data generated during the trusted setup. These data cannot be generated on the client's device and hence needs to be stored in the zkSENSE app. This is quite limiting in presence of low-end devices which might not have that amount of free memory, requiring swapping and thus a further

[10]The non-intuitive higher CPU usage at the S9 is due to the fact that this is a personal device with hard to filter background activities from other apps.



(a) ZKP computation, using zkSVM, consumes about 5 mAh or 0.2% of the device's battery.

(b) Hashing and ZKP computation, using the zkSNARK solution, consume an extra 10 and 50mAh, or an overall of about 2% on the device's battery.

Figure 3.8: Energy consumption per operation on a low-end hardware (J3). Only considering the non-negligible operations.

increasing in execution time,

Battery consumption: We quantify the extra battery discharge (in mAh) associated with zkSENSE's key operations. First, we compute the battery discharge of each operation (i.e. detection, commitment, ZKP creation) from the fine grain current and voltage measurements reported by the power meter. Next, we derive the additional battery discharge caused by each zkSENSE's operation by subtracting the baseline discharge from each specific operation. In Figure 3.8a and Figure 3.8b, we plot the results[11] of zkSVM and zkSNARK based solution respectively. As expected from the previous results, the battery overhead imposed by zkSENSE's detection and committing operation is negligible. Even ZKP computation only consumes about 5 mAh or 0.2% of the J3's battery (2,600 mAh). Even assuming five zkSENSE's humanness verification per day, the later result suggests minimal impact on battery life even for entry-level devices. Note that this is a worst case operation, given the entry level device. The lower duration and CPU utilization, coupled with bigger batteries, make zkSENSE less noticeable in term of battery consumption on more powerful devices. On the other hand, zkSNARK based solution drains more battery by an order of magnitude, where each validation cost about 2% of the device battery.

Data consumption: Finally, we compute what is the data consumption required by the user to send a proof. The proof consists of the commitments of the difference vector, the average and standard deviation, for each

[11]Given the design of S9 device, we were unable to wire its battery with the power meter. Therefore we could not measure the energy consumption on this device.

of the input vectors the user submits. We use data from two sensors, namely the gyroscope and accelerometer, and for each sensor, we use three axis data. Moreover, we split each period into two segments as explained in Section 3.1.1. This results in a total of 12 input vectors. For every input vector, the proof consists of 14 KB. In our library, we implemented the trivial construction, where we build 12 of such proofs in parallel, resulting in 167 KB. However, there are ways to reduce this overhead, for which we provide the estimates.

3.7.1 zkSENSE Vs. reCAPTCHA

As a next step, we compare the performance of zkSENSE with the state-of-the-art privacy-preserving humanness attestation mechanism (i.e. visual CAPTCHA). To do so, we developed an Android app which embeds reCAPTCHA for Android [Goo21]. The app is minimal[12] to ensure its performance evaluation covers only the CAPTCHA aspect rather than any extra components. For the same reason, we opted for Android reCAPTCHA rather than setting up a webpage with a CAPTCHA to solve. This alternative approach would require an Android browser for testing and the performance evaluation would be tainted by the extra cost of running a full browser. We did not evaluate Privacy Pass [Dav+18] for two reasons: 1) it currently requires a full browser along with an add-on, 2) it lacks support on mobile devices. Note that we do not expect critical performance difference between Privacy Pass and Android reCAPTCHA since they use a very similar strategy. Their difference instead lies in how invasive they are, both in how frequently they require user input and from a sensor data collection standpoint.

Using the above application we setup the following experiment. We enable remote access to the S9 device via the browser using Android screen mirroring [JMO21b] coupled with noVNC [JMO21a]. Then we asked 10 volunteers to visit the device from their browsers and solve CAPTCHAs as needed by the app. The app was coded such that users can continuously request for a new CAPTCHA to solve. Note that Android reCAPTCHA, as reCAPTCHA v3, leverages client side behavior to minimize friction, i.e. whether to ask or not a user to solve a visual CAPTCHA like "click on all images containing a boat". It follows that often users do not need to solve any visual CAPTCHA. We label *automatic* all the samples we collected where our volunteers did not have to solve a CAPTCHA. We instead label *manual* all the samples where

[12]Source code: `https://github.com/svarvel/CaptchaTest`

	zkSENSE	**reCAPTCHA Automatic/Manual**
Execution time	3	1.4/8.9 sec
CPU utilization	15%	3/15%
Memory utilization	20 MB	20 MB
Replay protection	No	Yes
Consumed data	167 KB	16 KB

Table 3.3: Performance of zkSENSE vs. Android reCAPTCHA.

human interaction was needed. To increase the chance of showing an actual CAPTCHA, we created 15 CAPTCHAs that the app rotates on.

Over one week, our volunteers have requested about 500 CAPTCHAs with a 70/30 split: 350 automatic and 150 manual. Table 3.3 directly compares zkSENSE with Android reCAPTCHA with respect to: execution time, CPU and memory utilization, and data consumption. The median was reported for each metric, further differentiating between automatic and manual for Android reCAPTCHA. The table shows that zkSENSE adds about 1.6 seconds to the time required by Android reCAPTCHA when no user interaction is needed. zkSENSE instead saves a whole 6 seconds to the (median) user by never requiring any interaction. Even considering the fastest user in our experiment (5.4 seconds), zkSENSE is about 2x faster —and 10x faster than the slowest user (28.5 seconds). This is possible because zkSENSE removes the need of user interaction, at the cost of a higher risk for replay attacks. With respect to CPU and memory utilization, Table 3.3 shows that the two mechanisms are quite similar and both very lightweight. Data-wise, reCAPTCHA outperforms zkSENSE (16 versus 160KB), which communication overhead is still minimal and bearable even by devices with very little connectivity.

Table 3.3 currently reports on "overall time" which includes both the computation time and the time required to report to the server. While for Google we have no control on the server endpoint – located within 10ms in our experiments – in our experiments the server runs in the same LAN with 1-2 ms delay (negligible). We currently use HTTP (POST) + TLS1.3 to return the proof. For TCP, we use an unmodified kernel running Cubic with an initial window of 10 packets (1500B MTU). Given our proof has a size of 160KB, the content delivery requires a worst case of: 1 RTT (for TCP) + 1 RTT (for TLS, in case of unknown server) + 3 RTT for TCP to transfer the data – assuming slow start (doubling of cwnd), aka 15K (10 MTU sized packets) + 30K + 60K + 60K (1/2 of the last cwnd available). This sums up to about 3/4 RTTs, which we could further reduce assuming a larger initial

cwnd, or using QUIC – thus bringing the duration down to a maximum of 2RTTs. Assuming a CDN runs such a service, as Google does, this would thus cost between 20ms (with optimization) up to 100ms. Assuming a very bad client connection, e.g., on mobile with RTT of 150ms, then this would cost an extra 300 to 750ms.

3.7.2 Summary

Our experiments show that the general purpose zkSNARK is not a viable solution for mobile-based ZKP computation (600 sec and 2% battery drain on a low-end device). By designing our own model (i.e. zkSVM), we reduce ZKP's execution time by 10x, achieving a duration of a bit less than 3 sec and 0.2% battery drain. This execution time is comparable with today's visual CAPTCHA solving time, 9.8 sec on average [All13]), thus making zkSENSE a serious competitor to state-of-the-art mechanisms for humanness attestation.

3.8 Further improvements to zkSVM

The opening and equality proofs we used (introduced in Section 2.2.3) could be improved, by implementing them using the techniques presented in [Boo+16]. This would reduce the complexity from linear to logarithmic, resulting in 3x improvement, with a size of the full proof of 56 KB. Similarly, we could use the batching techniques for the range proofs, as presented in [Bün+18]. This would further reduce the size of the proof to 45 KB.

However we can further reduce the data consumption by batching the proofs presented in Section 3.3.1. In this section we present how we can achieve this and prove the corresponding properties of the scheme presented. In particular, we present the batching technique used for range proofs in [Bün+18] for our IP-ZKP. With these improvements, we recompute the data consumption of zkSENSE and show these techniques further improve the construction.

Protocol 3.8.1. Batch IP-ZKP Given the similarities of IP-ZKP with the bulletproofs system, we follow the same path for the batch proofs. In particular, the prover proves the following argument

$$SPK\{([\alpha, \gamma, \boldsymbol{a}, \boldsymbol{b}, ic]_{i=1}^{m}): \\ A_i = h^{\alpha_i} \boldsymbol{g}_i^{\boldsymbol{a}_i} \boldsymbol{h}_i^{\boldsymbol{b}_i} \ \wedge \ V_i = g^{ic_i} h^{\gamma_i} \ \wedge \ \langle \boldsymbol{a}_i, \boldsymbol{b}_i \rangle = ic_i \text{ for } i \in [1, m]\}$$

with $\boldsymbol{g}_i, \boldsymbol{h}_i$ distinct for all $i \in [1, m]$.

The proof presented in the previous section achieves logarithmic complexity with respect to the number of elements in each vector. The trivial way to solve this "batch" proof would with a multiplicative factor m —by performing m of the proofs presented in Section 3.3.1. However, if we follow the work presented in [Bün+18], we can reduce the overhead of batching to an additive factor of m. In particular, if we have that the size of the vectors is N, we achieve a complexity of $2 \cdot \log(N \cdot m) + 4$. The intuition of this proof is the same as the one presented in [Bün+18], where instead of proving m times the logarithmic proof, we generate a single proof for an extended vector. This extended vector is the concatenation of all witness vectors, in particular,

$$\begin{aligned} \tilde{\boldsymbol{a}} = &\quad [\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_m], \\ \tilde{\boldsymbol{b}} = &\quad [\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_m]. \end{aligned}$$

However, simply changing the witness vectors is not sufficient to prove that the argument holds. In particular, proving that $\langle \tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}} \rangle = \sum_{i=1}^{m} ic_i$ is not sufficient to prove that $\langle \boldsymbol{a}_i, \boldsymbol{b}_i \rangle = ic_i$ for $i \in [1, m]$. Hence, we need to perform slight modifications to the proof. In particular, we require the verifier to send a challenge before the prover commits to the polynomial, so that the prover can use different powers of this challenge to separate the different vectors in the same inner product equation. In particular, the prover and verifier interact as follows.

$\mathcal{P}$**:** It computes blinding vectors, $\boldsymbol{s}_L, \boldsymbol{s}_R$ for each vector of the inner product and commits to it:

$$\boldsymbol{s}_L, \boldsymbol{s}_R \leftarrow_{\$} \mathbb{Z}_p^{N \cdot m}, \tag{3.25}$$

$$\rho \leftarrow_{\$} \mathbb{Z}_p, \tag{3.26}$$

$$S = h^{\rho} \boldsymbol{g}^{\boldsymbol{s}_L} \boldsymbol{h}^{\boldsymbol{s}_R} \in G. \tag{3.27}$$

$\mathcal{P} \rightarrow \mathcal{V}$**:** S

$\mathcal{V}$**:** $y \leftarrow_{\$} \mathbb{Z}_p^*$

$\mathcal{V} \rightarrow \mathcal{P}$**:** y

$\mathcal{P}$**:** Now the prover defines two linear vector polynomials, $l(X), r(X) \in$

$\mathbb{Z}_p^{N\cdot m}[X]$, and a quadratic polynomial as follows:

$$l(X) = \tilde{\boldsymbol{a}} + \boldsymbol{s}_L \cdot X \in \mathbb{Z}_p^{N\cdot m}[X], \tag{3.28}$$

$$r(X) = \sum_{i=0}^{m-1} \tilde{\boldsymbol{b}}_{[(i-1)\cdot N:i\cdot N]} \cdot y^i + \boldsymbol{s}_R \cdot X \in \mathbb{Z}_p^{N\cdot m}[X], \tag{3.29}$$

$$t(X) = \langle l(X), r(X)\rangle = t_0 + t_1 \cdot X + t_2 \cdot X^2 \in \mathbb{Z}_p^{N\cdot m}[X]. \tag{3.30}$$

By creating like that the polynomials, it allows for an evaluation of the polynomial at a given point without disclosing any information about the vectors $\tilde{\boldsymbol{a}}$ or $\tilde{\boldsymbol{b}}$. The prover needs to convince the verifier that the constant term of $t(X)$ equals $\sum_{i=1}^{m} y^i i c_i$. To do so, the prover commits to the remaining coefficients of the polynomial

$$\tau_1, \tau_2 \leftarrow_{\$} \mathbb{Z}_p, \tag{3.31}$$

$$T_i = g^{t_i} h^{\tau_i} \in G, i = \{1, 2\}. \tag{3.32}$$

$\mathcal{P} \rightarrow \mathcal{V}$: S, T_1, T_2

$\mathcal{V}$: $c \leftarrow_{\$} \mathbb{Z}_p^*$

$\mathcal{V} \rightarrow \mathcal{P}$: c

$\mathcal{P}$: It computes the response using the challenge received

$$\boldsymbol{l} = l(c) = \tilde{\boldsymbol{a}} + \boldsymbol{s}_L \cdot c \in \mathbb{Z}_p^{N\cdot m}, \tag{3.33}$$

$$\boldsymbol{r} = r(c) = \sum_{i=0}^{m-1} \tilde{\boldsymbol{b}}_{[(i-1)\cdot N:i\cdot N]} \cdot y^i + \boldsymbol{s}_R \cdot c \in \mathbb{Z}_p^{N\cdot m}, \tag{3.34}$$

$$\hat{t} = \langle \boldsymbol{l}, \boldsymbol{r}\rangle \in \mathbb{Z}_p, \tag{3.35}$$

$$\tau_c = \tau_2 \cdot c^2 + \tau_1 \cdot c + \sum_{i=0}^{m-1} y^i \gamma_i, \tag{3.36}$$

$$\mu = \alpha + \rho \cdot c \in \mathbb{Z}_p. \tag{3.37}$$

$\mathcal{P} \rightarrow \mathcal{V}$: $\tau_c, \mu, \hat{t}, \boldsymbol{r}, \boldsymbol{l}$.

$\mathcal{V}$: The verifier needs to check that $\boldsymbol{r}, \boldsymbol{l}$ are correct and the inner product relation holds with respect to $\hat{t}$. However, it needs to take into account the different powers of the challenge that have been included in the commitment. To this end it computes $\boldsymbol{h}' = \prod_{i=0}^{m-1} \boldsymbol{h}_i^{y^{-i}}$. This will cancel

out the challenge in the verification function, and allow the verifier to check the above statement. To this end it performs the following checks:

$$g^{\hat{t}} h^{\tau_c} \stackrel{?}{=} \prod_{i=0}^{m-1} V^{y^i} \cdot T_1^c \cdot T_2^{c^2}, \tag{3.38}$$

$$P = A \cdot S^c \cdot \boldsymbol{h}' \in G, \tag{3.39}$$

$$P \stackrel{?}{=} h^{\mu} \cdot \boldsymbol{g}^{\boldsymbol{l}} \cdot \boldsymbol{h}'^{\boldsymbol{r}}, \tag{3.40}$$

$$\hat{t} \stackrel{?}{=} \langle \boldsymbol{l}, \boldsymbol{r} \rangle. \tag{3.41}$$

If all checks validate, then this means that the statement is true with very high probability.

To make this proof logarithmic, we use the same trick as in the original paper. Instead of sending the vectors $\boldsymbol{r}, \boldsymbol{l}$ to prove the inner product relation, we leverage IP-ZKP over the blinded vectors recursively. Note that at this point, the logarithmic factor is of $O(\log(m \cdot n))$ instead of the trivial $O(m \cdot \log(n))$.

Theorem 4. The batch inner product proof presented in Protocol 3.3.1 has perfect completeness, perfect special honest verifier zero-knowledge, and knowledge soundness.

Proof. Completeness follows from the fact that, if

$$\langle a, b \rangle = c,$$

then

$$\langle a, N \cdot b \rangle = N \cdot c.$$

It is then straightforward to see that by generating $\boldsymbol{h}'$, and performing the checks by the verifier, an honest run of the protocol will always validate.

To prove special honest verifier zero-knowledge we construct a simulator that generates valid statements which are indistinguishable from random. In particular, the simulator acts as follows:

$$\begin{aligned}
y, c &\leftarrow_{\$} \mathbb{Z}_p^*, \\
\boldsymbol{l}, \boldsymbol{r} &\leftarrow_{\$} \mathbb{Z}_p^{N \cdot m}, \\
\hat{t}, \tau_c, \mu &\leftarrow_{\$} \mathbb{Z}_p, \\
T_2 &\leftarrow_{\$} G, \\
T_1 &= \left(g^{\hat{t}} h^{\tau_c} \cdot \left(\prod_{i=0}^{m-1} V^{y^i} \right)^{-1} \cdot T_2^{-c^2} \right)^{1/c}, \\
S &= \left(h^{\mu} \cdot \boldsymbol{g}^{\boldsymbol{l}} \cdot \boldsymbol{h}^{\boldsymbol{r}} \cdot A^{-1} \right)^{1/c}.
\end{aligned}$$

In the simulated transcript,

$$(S, T_1, T_2; c; \boldsymbol{l}, \boldsymbol{r}, \hat{t}, \tau_c, \mu),$$

all elements except for S and T_1 are random, and the latter two are computationally indistinguishable from random due to the DDH assumption (Section 2.1).

Also, note that S and T_1 are generated following the verification equations, hence this simulated conversation is valid.

We now construct an extractor to prove knowledge soundness. The extractor runs the prover with three different values of the challenge c and m different values of the challenge y, ending with the following valid proof transcript:

$$\begin{aligned}
&(S, T_1, T_2; y_i, c'; \boldsymbol{l}', \boldsymbol{r}', \hat{t}', \tau_c', \mu')_{i=1}^m,\\
&(S, T_1, T_2; y_i, c''; \boldsymbol{l}'', \boldsymbol{r}'', \hat{t}'', \tau_c'', \mu'')_{i=1}^m,\\
&(S, T_1, T_2; y_i, c'''; \boldsymbol{l}''', \boldsymbol{r}''', \hat{t}''', \tau_c''', \mu''')_{i=1}^m.
\end{aligned}$$

Additionally, the extractor invokes the extractor of the inner product argument. This extractor is proved to exist in the original paper of Bünz *et al.* [Bün+18]. For each proof transcript, the extractor first runs the inner product extractor, to get a witness $\boldsymbol{l}, \boldsymbol{r}$ to the inner product argument such that $P = h^{\mu}\boldsymbol{g}^{\boldsymbol{l}}\boldsymbol{h}^{\boldsymbol{r}} \wedge \langle \boldsymbol{l}, \boldsymbol{r} \rangle = \hat{t}$. With this witness, and using two valid transcripts, one can compute linear combinations of (3.17), we can compute $\alpha, \rho, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{s}_L, \boldsymbol{s}_R$ such that $A = h^{\alpha}\boldsymbol{g}^{\boldsymbol{a}}\boldsymbol{h}^{\boldsymbol{b}}$ and $S = h^{\rho}\boldsymbol{g}^{\boldsymbol{s}_L}\boldsymbol{h}^{\boldsymbol{s}_R}$. Such an extraction of these values proceeds as follows:

$$A \cdot S^{c'} = h^{\mu'}\boldsymbol{g}^{\boldsymbol{l}'}\boldsymbol{h}^{\boldsymbol{r}'}, \qquad A \cdot S^{c''} = h^{\mu''}\boldsymbol{g}^{\boldsymbol{l}''}\boldsymbol{h}^{\boldsymbol{r}''}.$$

Combining both relations we have

$$S = \left(h^{\mu'-\mu''}\boldsymbol{g}^{\boldsymbol{l}'-\boldsymbol{l}''}\boldsymbol{h}^{\boldsymbol{r}'-\boldsymbol{r}''}\right)^{\frac{1}{c'-c''}}.$$

The extraction of A follows.

Using these representations of A and S, as well as $\boldsymbol{l}^i$ and $\boldsymbol{r}^i$ with $i \in \{', '', '''\}$, we have that

$$\begin{aligned}
\boldsymbol{l}^i &= \boldsymbol{a} + \boldsymbol{s}_L c^i,\\
\boldsymbol{r}^i &= \boldsymbol{b} + \boldsymbol{s}_R c^i.
\end{aligned}$$

If these do not hold for all challenges, then the prover has found two distinct representations of the same group element, yielding a non-trivial discrete logarithm relation.

Now, we can extract the values T_i with $i \in \{1, 2\}$ from equation (3.15), by fixing a challenge y_0, and computing the linear operations as follows:

$$g^{\hat{t}'} h^{\tau_c'} = \prod_{i=0}^{m-1} V^{y_0^i} \cdot T_1^{c'} \cdot T_2^{c'^2},$$
$$g^{\hat{t}''} h^{\tau_c''} = \prod_{i=0}^{m-1} V^{y_0^i} \cdot T_1^{c''} \cdot T_2^{c''^2},$$
$$g^{\hat{t}'''} h^{\tau_c'''} = \prod_{i=0}^{m-1} V^{y_0^i} \cdot T_1^{c'''} \cdot T_2^{c'''^2},$$

which we can combine to get the following representation of T_2:

$$T_2 = \left(\left(g^L h^R \right)^{\frac{1}{c'-c'''}} \right)^{\frac{c'+c'''}{c'+c''}}$$

with $L = \frac{\hat{t}'-\hat{t}''}{c'-c''} - \frac{\hat{t}'-\hat{t}'''}{c'-c'''}$ and $R = \frac{\tau_c'-\tau_c''}{c'-c''} - \frac{\tau_c'-\tau_c'''}{c'-c'''}$. Extractions of T_1. Additionally, from that same equation, we can compute ic and γ such that

$$g^{ic} h^{\gamma} = \prod_{i=0}^{m-1} V^{y^i}.$$

Given that the extractor is running the prover with m different challenges y, we can then compute $(ic, \gamma)_{i=1}^m$ with $g^{ic_i} h_i^{\gamma} = V_i$. If for any transcript, we have that

$$\hat{t}^i \neq \sum_{j=0}^{m-1} y_k^j \cdot ic_i + t_1 c^i + t_2 c^{i2}$$

with $i \in \{', '', '''\}$ and $k \in [1, m]$, then the extractor has again found a non trivial discrete logarithm relation.

Finally, let $P(X) = \langle l(X), r(X) \rangle$. Due to the validity of the transcripts, we have that $P(X)$ equals $t(X) = \sum_{j=0}^{m-1} y^j ic_i + t_1 X + t_2 X^2$ at least in the different challenges c', c'', c'''. In other words, the polynomial $P(X) - t(X)$ has at least three roots, and is of degree 2, hence it must be the zero polynomial. Therefore, we have that $t(X) = P(X)$. This implies that the zero coefficient of $t(X)$, namely ic, equals $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ and we have a valid witness for the statement. □

This batching further reduces the communication complexity. In particular, for every sensor vector we compute two IP-ZKPs. With the naïve way explained in Section 3.4, we compute the two IP-ZKPs for each coordinate of each vector and their corresponding 'difference vector', resulting in 12 proofs (resulting in a multiplicative factor of 12). If, in contrast, we use the scheme presented above, the factor would turn additive, and considerably reduce the proof size. In particular, the number of points sent through the wire would be reduced to a total of 16KB, which is a 64% improvement over the evaluation presented in Section 3.7. We leave as future work the implementation of this scheme.

CHAPTER 4

Internet Voting

This chapter presents the work performed throughout the thesis in what regards internet voting. In particular, we present the work published in [QA+20] and [LQAT20]. These papers are an extension of what was initially published in [QA+17], each with different trade-offs. First we present NetVote [QA+20], which provides an internet voting scheme with linear filtering (with respect to the number of votes), but provides a slightly weaker notion of coercion resistance. Secondly, we present VoteAgain [LQAT20], which presents a scheme with poly-logarithmic filtering complexity but provides the stronger notion of coercion resistance. In this chapter we also include minor modifications on the formal definition, protocol description, and proofs that improve the security and assumptions of the protocols. We begin, in Section 4.1, with a comparison of the two existing solutions used to mitigate coercion resistance. In Section 4.2 we introduce some cryptographic concepts, particular to our constructions, that were not introduced in Chapter 2, as well as the parties involved in the protocols. Section 4.3 presents an overview of our constructions, and Section 4.4 the formal definitions of the security properties. Next, in Section 4.5 and Section 4.6 we present our two constructions, NetVote and VoteAgain respectively. Finally, we conclude in Section 4.7 with some final remarks.

Table 4.1: Comparison of assumptions in pre-election phase and election phase required to mitigate coercion attacks in fake credentials and revoting-based systems.

Assumptions	Fake Credentials	Revoting
Pre-election phase		
No coercion	✓	N/A
Inalienable authentication	✓	N/A
Election phase		
Lie convincingly	✓	✗
Coercer absent at some point during election	✓	✓
Absence of coercer after coercion	✗	✓
Device holding voting secrets or need to remember special pwds	✓	✗
Inalienable authentication	✗	✓

4.1 Fake credentials vs. re-voting

Coercion in remote elections is an important threat that requires strong assumptions regarding the adversary and the user interaction with the election. As presented in Section 2.3.2, current solutions mitigate coercion either by the use of fake credentials, or by allowing voters to re-vote. In this section we give an intuition of the motivation of choosing the latter for both our works. We do not analyse specific constructions, but rather study the intrinsic limitations of each approach.

We now set to compare the differences between the two settings, as shown in Table 4.1. For fake credentials setting, the assumptions are the following:

1. User needs inalienable means of authentication.
2. User needs to lie convincingly while being coerced.
3. User needs to store cryptographic material securely and privately.
4. Coercer needs to be absent during the registration and at some point during the election.

In the case of the re-voting setting, the assumptions are as follows:

1. User needs inalienable means of authentication.

2. Coercer needs to be absent at the end of the election.

Current proposals have failed to present a solution to remove the assumption on inalienable means of authentication, and it seems that it is an intrinsic problem to remote voting. A voter, either to register or to cast a vote, will need to authenticate itself to prove that it is an eligible voter. If these authentication means can be removed by an adversary, then coercion is inevitable.

In our opinion, fake credential schemes seem to have stronger assumptions. Lying convincingly to an adversary while voting can be a challenge to some. However, this is not the only limitation. It is clear from how much money it has been lost in the cryptocurrency world (because of losing keys) that storing cryptographic credentials securely and privately is not obvious [Cry18]. Moreover, having to store the cryptographic material in a device opens attack vectors for a coercer to impede re-voting, simply by removing the device where these keys are stored. Last, but not least, a study by Neto *et al.* [Net+18] concluded that *more* than 90% of the participants did not understand the concept of casting fake votes, and were uncomfortable with the fact of not being able to distinguish between real or fake votes at the time of casting. This result puts the usability of fake credential systems for the common public under question.

Finally, we argue on the difference of the strength of the assumptions regarding the absence of the adversary. Both approaches require that the coercer is absent during a fixed period. In the re-voting setting the coercer must not allow to cast a vote to the coerced voter at the end of the election. As long as the voter is left alone enough time to cast a vote (say 5 min) before the end of the election, the voter will be able to escape coercion. In the other case, the coercer needs to be absent during registration and some time during the election. In general, a registration process can happen across several hours or days (In Spain it is 24 days for citizens living abroad [Mar19]). This enlarges the time the adversary has to perform the attack —it is no longer limited to five minutes. Clearly, both solutions allow an adversary to successfully perform a targeted attack. However, this intuitively shows that producing a large scale coercion attack in a small time frame (the coercer needs to coerce all voters at the same time, i.e. before the end of the election) is harder than performing one during the registration phase. This motivates our choice of a re-voting scenario rather than fake credentials.

4.2 Parties and Cryptographic background

This section introduces the parties involved and the specific notation and cryptographic background not covered in Chapter 2.

4.2.1 Parties

The parties involved throughout the protocol are:

- The electorate, formed by n_t voters, of which we only consider the subset that takes part in the election, say $n_{\mathcal{V}}$ voters, with $n_{\mathcal{V}} \leq n_t$ identified as $\mathcal{V} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_{n_t}\}$.
- The n_c candidates, $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{n_c}\}$.
- The k trustees $T = \{T_1, T_2, \ldots, T_k\}$, each holding a share, $sk_{v,i}$, of the threshold voting key (see Subsection 2.2.1), pk_v. Similarly, the trustees handle, in a decentralised fashion the voting credential generation. This is, whenever voters want to cast a vote they request from trustees the voting token. However, to simplify the description of the protocol, we explicitly treat the Certificate Authority, CA, as a different entity.
- Next, we require the participation of an append-only Public Bulleting Board, PBB, which validates and posts the votes. In all our models, we implicitly assume that the PBB is append-only.
- Finally, the Tallying Server, TS, which performs the filtering and the counting of votes. For the sake of minimizing the coercion resistance assumptions, TS also holds a share of the voting key, pk_v, such that its participation in the decryption procedure is required.

4.2.2 Cryptographic Background

The internet voting protocols presented in this thesis share the same key generation procedure, and same cryptographic scheme, namely ElGamal (see Section 2.2). The voting key is distributively generated among a set of trustees (see Subsection 2.2.1), such that a subset of these need to collaborate in order to decrypt the ciphertexts. To this end, the trustees jointly run $\texttt{DistKeyGen}(\epsilon, t, k)$ to generate the trustees share of the key, pk_T, and the distinct shares of the tellers, $sk_{v,i}$. As mentioned above, to improve the security of the scheme, we require the participation of the TS in the decryption procedure. Hence, when a voter encrypts its choice, it encrypts it under

pk_T and then under pk_{TS}. For sake of simplicity we denote this action implicitly by a single encryption under the master voting key, pk_v. To encrypt a vote for candidate $\mathcal{C}_i$, a voter calls $(V, \Pi_V) = \texttt{VoteEnc}(pk_v, \mathcal{C}_i)$ to obtain an encrypted vote, V, and zero-knowledge proof, Π_V, that V encrypts a choice for a valid candidate. We denote the encryption of the zero candidate (i.e. no candidate) with explicit randomiser $r \leftarrow_\$ \mathbb{Z}_p$ by $\texttt{VoteZEnc}(pk_v; r)$. This is equivalent to $\texttt{VoteEnc}(pk_v, 0)$ with explicit randomiser r. The algorithm $\Pi_V.\texttt{Verify}(pk_v, V, \Pi_V)$ outputs $\top$ if the encrypted vote V is correct, and $\bot$ otherwise. We use deterministic encryption (with randomness zero) as a cheap verifiable 'encoding' for the dummy ballots. This allows the TS to include dummy ballots in the verifiable shuffle, introduced below, in a cheap verifiable way.

We use verifiable reencryption shuffles [BG12; Gro10] to support coercion resistance in a privacy-preserving and verifiable way. These enable an entity to verifiably shuffle a list of homomorphic ciphertexts in such a way that it is infeasible for a computationally bounded adversary to match input and output ciphertexts. These are defined by a function $\texttt{Shuffle}(A) = (\Pi_S, A')$ which on input a list of ciphertexts, A, outputs a shuffled list of ciphertexts, A', and a proof of shuffle, Π_S.

In the construction of NetVote, Section 4.5, the TS, in order to prove correctness of filtering, needs to prove that an encrypted counter, $\texttt{Enc}(pk, \texttt{count}_1)$ is greater than another encrypted counter, $\texttt{Enc}(pk, \texttt{count}_2)$, without disclosing any information about the counters. For this it uses the homomorphic property of the encryption scheme to subtract both encryptions:

$$\texttt{count}' = \texttt{Enc}(pk, \texttt{count}_1)/\texttt{Enc}(pk, \texttt{count}_2) = \texttt{Enc}(pk, \texttt{count}_1 - \texttt{count}_2).$$

If $\texttt{count}_1$ is greater than $\texttt{count}_2$, we expect the subtraction to be greater than zero, and negative otherwise. However, as we are working over finite fields, even if $\texttt{count}_2$ is greater, the subtraction will be positive. Given that the counters are numbers of at most 32 bits, then we know that if $\texttt{count}_1$ is greater, then $\texttt{count}'$ will also be a number of at most 32 bits. In the opposite scenario, $\texttt{count}'$ will be a much bigger number (given that the order of the finite field is a number of 256 bits). It suffices then to prove that $\texttt{count}'$ is a number of at most 32 bits. For this, we use the range proof presented by Bünz *et al.* [Bün+18]. To denote the proof that a number is greater than another, we use Π_{GT}.

In NetVote, we use anonymous credentials during the registration phase and vote cast. The only requirement of these credentials is that they certify certain attributes which are used to group voters by electoral colleges

and filter votes cast by the same voter. Several constructions exist in the literature, [Bra00; CL02; Par+09]. We instantiate them by the use of three algorithms: the request, $\texttt{ReqCred}(\texttt{auth})$, where the user authenticates to the credential authority and requests an anonymous credential; the generation, $\texttt{Cert}(\{\texttt{attr}\}_{i=1}^{n}) = \texttt{CredGen}(\{\texttt{attr}\}_{i=1}^{n})$, where upon receipt of a certificate request, the certificate authority generates one with the attributes, attr, assigned to the user; and the verification of the certification, $\texttt{CertVerify}(pk_{CA}, \texttt{Cert}(\{\texttt{attr}\}_{i=1}^{n}))$, where with input a certificate and the public key of the certificate authority, verifies the correctness of the certificate. It outputs $\top$ if the verification succeeds, and $\perp$ otherwise. While any type of attributes can be added in these certificates, throughout the chapter we only consider a unique anonymous identifier per voter, and leave additional attributes optional to particular electoral runs.

Finally, we use an append-only PBB where votes, proofs (re-encryption, shuffle, decryption) and all intermediate steps until the tally step are posted. A specification of a possible construction is explained in [HL09]. Another interesting approach for a PBB is the use of blockchain technologies, as presented in [MSH17].

4.3 Overview

As it was mentioned before, this chapter presents two internet voting schemes, whose goals differ in the asymptotic complexity and security properties. However, the rationale used for both is similar, and principally both seek the usability on what concerns users. In this section we give an overview of the basic design ideas, and present both schemes in more detail in Sections 4.5 and 4.6.

The protocols can be divided in three phases: pre-election, election and tallying phases. During the *pre-election* phase all server keys are generated. CA randomly generates n_t voter identifiers, VId_i for $1 \leq i \leq n_t$, and other user data. In the *election-phase* the voters first obtain a voting certificate, which allows them to cast a vote while proving they are eligible voters. This certificate is ephemeral, and is generated each time that a voter wants to cast a vote. The intuition behind using ephemeral certificates is that by having one-time usage keys, we avoid the user the need of storing and transporting cryptographic material safely. Moreover, enforcing a single key to users might require them to use a single device throughout the election, limiting the ability to escape coercion. Each certificate generated for the same voter contains the same voter identifier in order to uniquely identify each vote cast by the same voter at the time of filtering. Using the encrypted identifier as

an attribute (instead of the decrypted identifier), and randomising it at each request (see Section 2.2.1), ensures that an adversary is not able to determine whether a particular voter revoted or not.

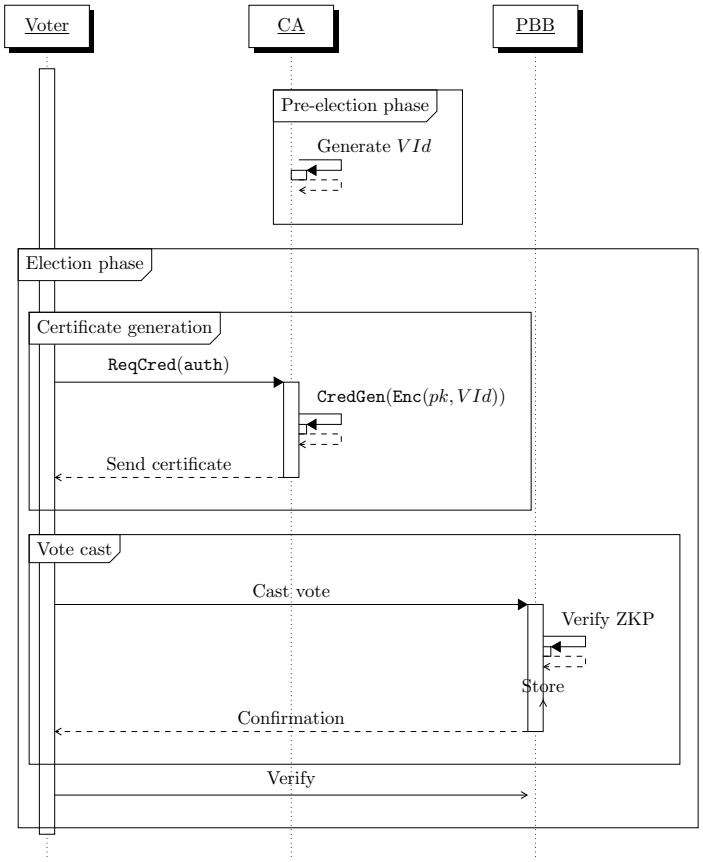


Figure 4.1: Diagram presenting a summary of the pre-election and election phases involving the voters, certificate authority and public bulletin board.

When the voter wants to cast a vote, she signs the encrypted ballot with the ephemeral certificate to prove, on the one hand, that she is an eligible voter, and on the other hand, to link the vote to the VId_i. This allows TS to filter votes, and select only one cast vote per user. She then sends her vote to PBB, which verifies the correctness of the vote and publishes it. The voter verifies that her vote was recorded as cast. These two steps are presented in Figure 4.1.

During the *tallying* phase, the TS generates 'dummy' certificates for each of the voters and casts 'dummy' votes in a verifiable manner. This avoids a coercer performing a 1009 attack (Section 2.3.2), by hiding the real size of groupings of votes cast by the same voter. Next, the votes are filtered by TS in a verifiable and private manner. Finally, the tellers, T, proceed

with a complete tally and decryption of the votes. This phase is presented in Figure 4.2.

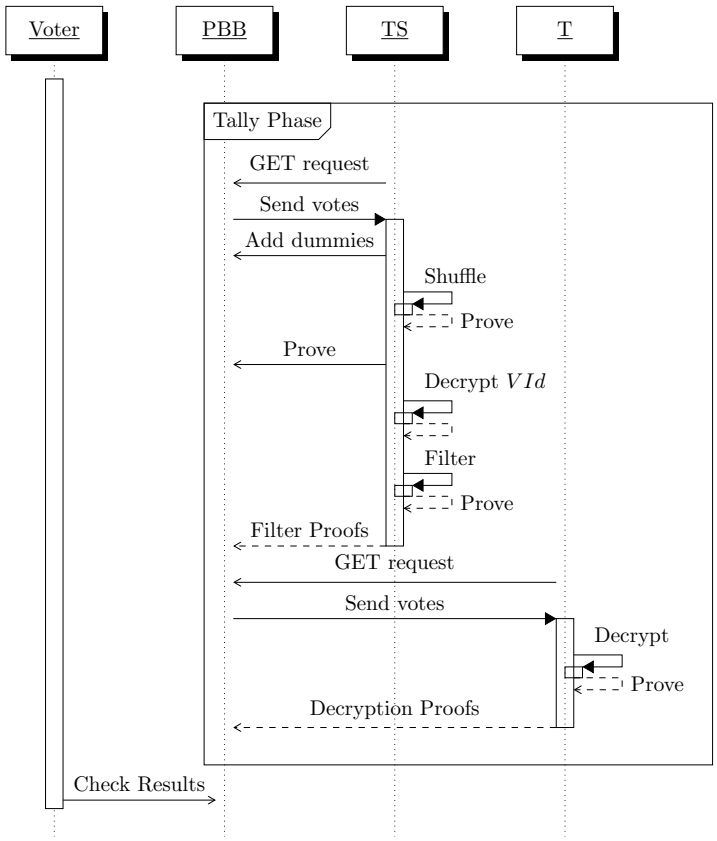


Figure 4.2: Diagram showing a summary of the tallying phase, involving the voters, the public bulletin board, the tallying server and the tellers.

4.4 Security Properties

In order to analyse the security properties, we define them using a general, protocol independent, syntax. The goal of this syntax is to be able to prove the security properties of other schemes with minimal possible changes to the game definitions. In this section we begin introducing the used syntax and we proceed with the formal definitions of ballot privacy, practical everlasting privacy, verifiability, coercion resistance and strict coercion resistance.

A voting scheme, $\mathcal{V}$, consists of nine protocols: $\mathtt{Setup}$, $\mathsf{Register}$, $\mathsf{CastVote}$, $\mathsf{VoteVerify}$, Valid, Filter, $\mathsf{VerifyFilter}$, Tally and Verify:

- $\texttt{Setup}(\mathcal{E}, \mathcal{C})$. In the pre-election phase, the scheme runs `Setup` to prepare the voting scheme for voting. This protocol takes as input the electoral roll $\mathcal{E}$, the list of all eligible voters, and the list of candidates $\mathcal{C}$.

- $\mathsf{Register}(i)$. Before casting a vote, voter $\mathcal{V}_i$ runs $\mathsf{Register}(i)$ to obtain a token/certificate τ that allows her to cast a vote.

- $\mathsf{CastVote}(\tau, \mathcal{C}_i)$. After registering, voters use their token τ and selected candidate $\mathcal{C}_i$ to cast their vote using the $\mathsf{CastVote}(\tau, \mathcal{C}_i)$ protocol. The user produces a ballot β containing her choice, and interacts with the voting scheme. If the user's ballot β is accepted by the voting scheme, the ballot β is added to the public bulletin board.

- $\mathsf{VoteVerify}(\beta, PBB)$. After casting a vote, voters can verify that the ballot was recorded as cast, i.e. they verify that the vote was successfully stored in PBB.

- $\mathsf{Valid}(\beta, PBB)$. Once the scheme receives a vote, it verifies that it is valid.

- $\mathsf{Filter}(PBB, L, sk_{TS})$. After voting and before the tallying phase, the voting scheme runs Filter. This protocol takes as input the PBB, the number of all cast votes and the secret key of the TS. It returns the filtered votes, $\mathcal{S}$, together with a proof of correctness, θ.

- $\mathsf{VerifyFilter}(PBB, \mathcal{S}, \theta)$. Any third party can use as input the PBB (to take the votes before filtering), $\mathcal{S}$ and θ to verify the correctness of the shuffle.

- $\mathsf{Tally}(\mathcal{S})$. After filtering, the voting scheme runs Tally. This protocol takes as input the set $\mathcal{S}$ of ballots on the bulletin board. It computes the tally of the votes and outputs an election result, z, and a proof of correctness Π_z of the election results.

- $\mathsf{Verify}(\mathcal{S}, z, \Pi_z)$. Finally, any third party can run Verify. This protocol takes the set $\mathcal{S}$ of ballots, the result, z and the proof of correct tally, Π_z, and checks its correctness.

The following game between an adversary, $\mathcal{A}$, and a challenger, $\mathcal{D}$, based on Bernhard et al. [Ber+15], formalizes ballot privacy. $\mathcal{A}$ wins the game if it manages to differentiate between a real and fake world. To model these two worlds, the game, $\texttt{Exp}^{\texttt{bpriv},b}_{\mathcal{A},\mathcal{D}}$, tracks two bulletin boards, PBB_0 and PBB_1 for each world respectively. At least t trustees are assumed to be honest. During

the game, the adversary can make calls to the $O\texttt{LRvote}(i, \mathcal{C}_0, \mathcal{C}_1)$ oracle, where $\mathcal{A}$ selects two possible candidates, $\mathcal{C}_0, \mathcal{C}_1$ for voter $\mathcal{V}_i$. The challenger produces voting tokens τ and generates one ballot for each candidate, β_0, β_1. It then places β_0 and β_1 in PBB_0 and PBB_1 respectively. It can call this oracle at any point of the game. Note that this oracle implicitly assumes that the adversary cannot submit different tokens for the same voter. This is the case under our assumption because at least t trustees are honest, and hence the adversary cannot forge certificates. Similarly, the adversary can call $O\texttt{cast}(\beta)$ oracle, where $\mathcal{A}$ has the ability to cast a vote for *any* voter. The same ballot, β, is generated for both bulletin boards. It can call this oracle at any point of the game. To view the state of the bulletin board, we give the adversary access to $O\texttt{board}()$, which allows the adversary to see the information posted until that moment in the bulletin board. It can call this oracle at any point of the game.

Finally, the adversary is given access to the $O\texttt{tally}(\mathcal{S}, \theta)$ oracle which allows $\mathcal{A}$ to request the result of the election given as input the filtered list of votes, $\mathcal{S}$, and the proof of correctness, θ. Recall that the adversary controls the tallying server, and hence can perform the filtering. The oracle first verifies the correctness of the proof, and proceeds with the tally computation. To avoid information leakage of the tally result, when $b = 1$ the result is counted on PBB_0, and the results and proofs are simulated. It can call this oracle once.

We denote the calls to the oracles by $\mathcal{A}^O$. At the end of the game, the adversary needs to output b', guessing which of the two worlds (real or fake) it is seeing. If it guesses correctly with non negligible probability, the adversary has won the game. We formally describe the experiment, $\texttt{Exp}^{\texttt{bpriv},b}_{\mathcal{A},\mathcal{D}}$, in Figure 4.3.

Definition 1. Consider a voting scheme $\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{CastVote}, \mathsf{VoteVerify}, \mathsf{Valid}, \mathsf{Filter}, \mathsf{VerifyFilter}, \mathsf{Tally}, \mathsf{Verify})$. We say the scheme has *ballot privacy* if there exists an algorithm $\texttt{SimTally}$ such that for all probabilistic polynomial time adversaries $\mathcal{A}$

$$\left|\Pr\left[\texttt{Exp}^{\texttt{bpriv},0}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C}) = 1\right] - \Pr\left[\texttt{Exp}^{\texttt{bpriv},1}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C}) = 1\right]\right|$$

is negligible with respect to ϵ.

4.4.1 Ballot privacy

Strong Consistency

The ballot privacy definition ensures that ballots and the proof of correct tally, Π, do not leak anything about how voters voted. However, maliciously

$\mathtt{Exp}^{\mathtt{bpriv},b}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C})$:
  $(pk, sk_{CA}, sk_{TS}, sk_v) \leftarrow \mathsf{Setup}(1^\epsilon, \mathcal{E}, \mathcal{C})$
  $b \leftarrow \mathcal{A}^O(pk, sk_{CA}, sk_{TS})$
  Output b'

$O\mathtt{LRvote}(\tau, \mathcal{C}_0, \mathcal{C}_1)$:
  Let $\beta_0 = \mathsf{CastVote}(\tau, \mathcal{C}_0)$ and $\beta_1 = \mathsf{CastVote}(\tau, \mathcal{C}_1)$
  If $\mathsf{Valid}(PBB_b, \beta_b) = \perp$ return $\perp$
  Else $PBB_0 \leftarrow PBB_0 \parallel \beta_0$ and $PBB_1 \leftarrow PBB_1 \parallel \beta_1$

$O\mathtt{cast}(\beta)$:
  If $\mathsf{Valid}(PBB_b, \beta) = \perp$ return $\perp$
  Else $PBB_0 \leftarrow PBB_0 \parallel \beta$ and $PBB_1 \leftarrow PBB_1 \parallel \beta$

$O\mathtt{board}()$:
  return PBB_b

$O\mathtt{tally}(\mathcal{S}, \theta)$
  If $\mathsf{VerifyFilter}(PBB_b, \mathcal{S}, \theta) = \perp$ return $\perp$
  $PBB_b \leftarrow PBB_b \parallel \mathcal{S}$
  $PBB_{1-b} \leftarrow PBB_{1-b} \parallel \mathsf{Filter}(PBB_{1-b}, |PBB_{1-b}|, sk_{TS})$
  $(z, \Pi_0) \leftarrow \mathsf{Tally}(PBB_0, sk_v)$
  $\Pi_1 = \mathtt{SimTally}(PBB_1, z)$
  return (z, Π_b)

Figure 4.3: In the ballot privacy experiment $\mathtt{Exp}^{\mathtt{bpriv},b}_{\mathcal{A},\mathcal{D}}$, the adversary $\mathcal{A}$ has access to the oracles $O = \{O\mathtt{LRvote}, O\mathtt{cast}, O\mathtt{board}, O\mathtt{tally}\}$. It is assumed that at least t trustees are honest. This ensures that the adversary is not capable of decrypting votes or misbehaving during voting certificate generation. It can call $O\mathtt{tally}$ only once.

crafted voting schemes might leak information about honest votes in the result z itself. To ensure that this is not possible, Bernhard et al. [Ber+15] introduced the notion of strong consistency. Intuitively, this notion ensures that the result z is equal to the result function applied directly to the valid ballots (skipping the filter and tally phase). We follow the exposition of Bernhard et al., but make some changes to account for the fact that our scheme selects ballots with the highest corresponding ballot number `count`, rather than simply the last per voter. This allows us to use this ballot privacy definition for both VoteAgain and NetVote, where the former considers the vote generated with the latest token as the 'last vote', while the latter counts the last vote submitted as the 'last vote'.

Our voting scheme depends on a formal result function ρ: $((\mathbb{Z}_p \times \mathbb{N}) \times \mathcal{C})^* \to \mathbb{R}$, where $\mathbb{Z}_p^*$ is the space of voters identifiers, $\mathbb{N}$ is the space of counters, and $\mathbb{R}$ is the result space. Our result function selects, for every $VId \in \mathbb{Z}_p$, the ballot $((VId, \mathtt{count}), c)$, where `count` is the maximal counter for this voter.

Then it counts the number of votes per candidate c in the selected ballots and returns the result.

To model that the result z output by Tally is consistent with the result function ρ, we require the existence of an extraction algorithm `Extract` that takes as input the TS's key sk_{TS}, the trustee key sk_v and a ballot, and outputs a tuple $((VId, \texttt{count}), c) \in ((\mathbb{Z}_p \times \mathbb{N}) \times \mathcal{C})$ with the corresponding voter identifier VId, ballot number `count` and candidate c in this ballot. If it fails to extract these values, it outputs $\perp$.

Moreover, we require the method Valid that validates ballots independent of the bulletin board. The function Valid takes as input the election public key pk and a ballot, and outputs $\top$ if the ballot is valid, and $\perp$ otherwise.

Definition 2 (Adapted from Bernhard et al. [Ber+15])**.** A voting scheme $\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{CastVote}, \mathsf{VoteVerify}, \mathsf{Valid}, \mathsf{Filter}, \mathsf{VerifyFilter}, \mathsf{Tally}, \mathsf{Verify})$ for an electoral roll $\mathcal{E}$ and candidate list $\mathcal{C}$ has *strong consistency* with respect to a result function ρ if there exists algorithms `Extract` and Valid as above, such that the following three conditions hold:

1. For any $(pk, sk_{CA}, sk_{TS}, sk_v)$ output by `Setup`, for all voters $i \in \mathcal{E}$ with voter identifier VId, for all $\tau \leftarrow \mathsf{Register}(i)$ where `count` is the corresponding ballot number, and for any ballot $\beta \leftarrow v(\tau, c)$ with $c \in \mathcal{C}$, we have that $\texttt{Extract}(sk_v, sk_{CA}, \beta) = ((VId, \texttt{count}), c)$.

2. For any $(PBB, \beta) \leftarrow \mathcal{A}()$ we have that $\mathsf{Valid}(PBB, \beta) = \top$ implies $\mathsf{Valid}(\beta) = \top$.

3. For all probabilistic polynomial time adversary $\mathcal{A}$ we have that

$$\Pr\left[\texttt{Exp}^{\texttt{s-cons}}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C}) = 1\right]$$

   is a negligible function in ϵ (see Figure 4.4 for the game).

The first condition ensures that `Extract` can extract $((VId, \texttt{count}), \mathcal{C})$ correctly for honestly created ballots. The second condition ensures that ballots that are accepted by Valid with respect to the board PBB must also be accepted by Valid. Finally, the third condition ensures that the adversary cannot produce bulletin boards where the result z does not correspond to the formal result function ρ executed on the individual ballots. (The adversary loses if Filter or Tally aborts because of an invalid bulletin board.)

Strong correctness

Finally, a malicious protocol designer might modify which ballots are accepted based on earlier ballots. To address this attack, Bernhard et

$\mathtt{Exp}^{\mathtt{s\text{-}cons}}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C})$:
$(pk, sk_{CA}, sk_{TS}, sk_v) \leftarrow \mathsf{Setup}(1^\epsilon, \mathcal{E}, \mathcal{C})$
$PBB = [\beta_1, \ldots, \beta_L] \leftarrow \mathcal{A}(pk, sk_{CA})$
If $\exists \beta_i$ s.t. $\mathsf{Valid}(\beta_i) = \perp$ then return 0
Let $\mathcal{S}, \theta \leftarrow \mathsf{Filter}(PBB, L', sk_{TS})$
Let $(z, \Pi) \leftarrow \mathsf{Tally}(PBB \parallel \mathcal{S} \parallel \theta, sk_v)$
If $z = \perp$ return 0
If $z \neq \mathsf{Tally}(\mathtt{Extract}(sk_v, sk_{CA}, \beta_1), \ldots, \mathtt{Extract}(sk_v, sk_{CA}, \beta_L))$
return 1, else return 0.

Figure 4.4: In the strong-consistency experiment $\mathtt{Exp}^{\mathtt{s\text{-}cons}}_{\mathcal{A},\mathcal{D}}$, adversary $\mathcal{A}$ must output a board PBB with ballots that are not tallied correctly given `Extract`.

$\mathtt{Exp}^{\mathtt{s\text{-}corr}}_{\mathcal{A},\mathcal{D}}(\epsilon)$:
$(pk, sk_{CA}, sk_{TS}, sk_v) \leftarrow \mathsf{Setup}(1^\epsilon)$
$(i, \tau, c, PBB) \leftarrow \mathcal{A}(pk, sk_{CA})$
Let $\beta = v(\tau, c)$
If v aborts because τ is invalid, return $\top$
Else return $\mathsf{Valid}(PBB, \beta)$

Figure 4.5: In the strong-correctness experiment $\mathtt{Exp}^{\mathtt{s\text{-}corr}}_{\mathcal{A},\mathcal{D}}$, the adversary outputs a board PBB such that an honest ballot by a voter of its choice is not valid.

al. [Ber+15] introduce the notion of strong correctness. Informally, a scheme has strong correctness if honestly generated ballots are accepted regardless of the content of the bulletin board.

Definition 3 (Adapted from Bernhard et al. [Ber+15])**.** Consider a voting scheme $\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{CastVote}, \mathsf{VoteVerify}, \mathsf{Valid}, \mathsf{Filter}, \mathsf{VerifyFilter}, \mathsf{Tally}, \mathsf{Verify})$ for an electoral roll $\mathcal{E}$ and candidate list $\mathcal{C}$. We say the scheme has *strong correctness* if

$$\Pr\left[\mathtt{Exp}^{\mathtt{s\text{-}corr}}_{\mathcal{A},\mathcal{D}}(\epsilon) = \perp\right]$$

is a negligible function in ϵ (see Figure 4.5 for the game).

4.4.2 Practical Everlasting Privacy

In this section, we present the first game-based definition of practical everlasting privacy. We use the definition as introduced by Arapinis *et al.* [Ara+13]. A more recent game-based definition of everlasting privacy (note the absence

$\texttt{Exp}^{\texttt{everbpriv},b}_{\mathcal{A},\mathcal{D}}(\epsilon,\mathcal{E},\mathcal{C})$:
  $b \leftarrow \mathcal{A}^{O}(pk, sk_{CA}, sk_{TS}, sk_v)$
  Output b'

$O\texttt{RunElection}(\epsilon,\mathcal{E},\mathcal{C})$:
  $(pk, sk_{CA}, sk_{TS}, sk_v) \leftarrow \mathsf{Setup}(1^\epsilon, \mathcal{E}, \mathcal{C})$
  Let $\tau_0 = \mathsf{Register}(\mathcal{V}_0)$ and $\tau_1 = \mathsf{Register}(\mathcal{V}_1)$
  Let $\beta_0^b = \mathsf{CastVote}(\tau_0, \mathcal{C}_b)$ and $\beta_1^b = \mathsf{CastVote}(\tau_1, \mathcal{C}_{1-b})$.
  $PBB_0 \leftarrow PBB_0 \parallel \beta_0^0 \parallel \beta_1^0$ and $PBB_1 \leftarrow PBB_1 \parallel \beta_1^1 \parallel \beta_0^1$
  $(z_0, \Pi_0) \leftarrow \mathsf{Tally}(PBB_0, sk_v)$
  $(z_1, \Pi_1) \leftarrow \mathsf{Tally}(PBB_1, sk_v)$
  return (z_b, Π_b)

Figure 4.6: In the practical everlasting privacy game, $\texttt{Exp}^{\texttt{everbpriv},b}_{\mathcal{A},\mathcal{D}}()$, the adversary $\mathcal{A}$ has access to the oracle $O = \{O\texttt{RunElection}\}$. In this scenario, the setup needs to happen inside the oracle, so that the voter identifiers are 'reset'.

of practical) was presented by Grontas *et al.* [GPZ19], which allows the future adversary to control all electoral entities during the election. Our schemes do not satisfy this stronger model of everlasting privacy, as we assume that the information generated by the Certificate Authority during the election is unreachable to the future adversary. In the definition of Arapinis *et al.*, it is assumed that the adversary can only get information that was posted in the PBB during the election. This is, all information that was exchanged during the election is not accessible to the adversary (such as timing attacks or tokens requests). We propose a game based definition, $\texttt{Exp}^{\texttt{everbpriv},b}_{\mathcal{A},\mathcal{D}}$, similar to $\texttt{Exp}^{\texttt{bpriv},b}_{\mathcal{A},\mathcal{D}}$. Again, $\mathcal{A}$ wins the game if it manages to differentiate between a real and fake world. To model these two worlds, the game tracks two bulletin boards, PBB_0 and PBB_1 for each world respectively. To model such a scenario, we allow the adversary to call on runs of the voting protocol, with electoral roll $\mathcal{E} = \{\mathcal{V}_0, \mathcal{V}_1\}$, and candidate list $\mathcal{C} = \{\mathcal{C}_0, \mathcal{C}_1\}$. Hence, the adversary can make calls to the $O\texttt{RunElection}(\mathcal{V}_0, \mathcal{V}_1, \mathcal{C}_0, \mathcal{C}_1)$ oracle, where the adversary chooses two voters and two candidates and requests the challenger to run the election. The challenger runs the election, by first running the `Setup` protocol, generating keys for all parties, and distinct random identifiers to each of the voters. It proceeds with the $\mathsf{Register}(i)$ protocol for each voter, generating voting credentials for each of the voters, then proceeds by casting votes for both voters in both worlds, and, finally, it runs the tally protocol.

Then, $\mathcal{A}$ needs to guess which world it is interacting with with its guess b'. We formally describe the game, $\texttt{Exp}^{\texttt{everbpriv},b}_{\mathcal{A},\mathcal{D}}$, in Figure 4.6. Note that the result is independent of the game bit b, as it will always be one vote for

$\mathcal{C}_0$ and one vote for $\mathcal{C}_1$.

Definition 4. Consider a voting scheme $\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{CastVote}, \mathsf{VoteVerify}, \mathsf{Valid}, \mathsf{Filter}, \mathsf{VerifyFilter}, \mathsf{Tally}, \mathsf{Verify})$. We say the scheme has *practical everlasting privacy* if for a computationally unbounded adversary $\mathcal{A}$

$$|\Pr\left[\mathtt{Exp}^{\mathtt{everbpriv},0}_{\mathcal{A},\mathcal{D}}(\epsilon,\mathcal{E},\mathcal{C})=1\right]-\Pr\left[\mathtt{Exp}^{\mathtt{everbpriv},1}_{\mathcal{A},\mathcal{D}}(\epsilon,\mathcal{E},\mathcal{C})=1\right]|$$

is negligible with respect to ϵ.

4.4.3 Verifiability

In their analysis, Achenbach et al. [Ach+15] adapt the correctness definition of Juels et al. [JCJ05] to the revoting setting. However, Achenbach et al.'s model does not take into account that voters may not check that their ballots are cast correctly, nor that newer ballots should supersede older ballots *even* if voters have been coerced or corrupted. To address these cases, we adapt the qualitative game-based verifiability definition of Cortier et al. [Cor+14] —which accounts for a malicious bulletin board and voters not checking their ballots —to our setting by adding the Register function and explicitly modeling revoting. As in Cortier et al. [Cor+14], our game does not model voter's intent, and assumes that the voting hardware, i.e., the device and software running CastVote, is honest. We refer to Cortier et al. [Cor+16] for a formal process-based computational model that does model verifiability with voter intent. We note that the correctness definition by Juels et al. [JCJ05] was renamed to 'verifiability' by Cortier et al. [Cor+14], and therefore any model satisfying the latter also satisfies the former.

In a nutshell, a voting scheme is verifiable [Cor+14] if for n_{C} corrupt voters, the result of the election always includes: (1) all votes by honest voters that verified whether their ballots were cast correctly, (2) at most n_{C} corrupted votes, and (3) a subset of the votes by honest voters that did not check if their ballots were cast correctly. These conditions ensure that while a malicious bulletin board can drop ballots of voters that do not check, it can insert at most n_{C} new votes.

Extending the current verifiability definition. We extend the definition presented by Cortier et al. [Cor+14] for the revoting setting to explicitly consider the number of votes cast by a voter, see Figure 4.7. The CA is honest, and hence we assume that at least t trustees are honest. The system implicitly tracks the number of tokens $\#\mathtt{tokens}(i)$ that have been obtained by voter $\mathcal{V}_i$. The game tracks when each voter is corrupted in a (initially empty) list of corruption events C, and tracks the honest votes in $\mathfrak{H}$. The

0 $\mathtt{Exp}^{\mathtt{verif},b}_{\mathcal{A},\mathcal{D}}(\epsilon,\mathcal{E},\mathcal{C})$:
1 $(pk, sk_{CA}, sk_{TS}, sk_v) \leftarrow \mathtt{Setup}(1^\epsilon,\mathcal{E},\mathcal{C})$
2 Set $\mathfrak{H} \leftarrow \emptyset$ and $\mathfrak{C} \leftarrow \emptyset$
3 $(PBB, \mathcal{S}, \theta, z, \Pi) \leftarrow \mathcal{A}^O(pk, sk_{TS}, sk_v)$
4 If $\mathsf{VerifyFilter}(PBB, \mathcal{S}, \theta) = \perp$ or $\mathsf{Verify}(PBB \parallel \mathcal{S} \parallel \theta, z, \Pi) = \perp$ return 0
5 Let $\mathtt{Verified} = \{(i_1, \mathtt{count}_1), \ldots, (i_{n_\nu}, \mathtt{count}_{n_\nu})\}$
correspond to checked ballots.
6 Let $\mathsf{Corrupted} = \{i \mid (i, \mathtt{count}) \in \mathsf{C} \wedge \forall (i, \mathtt{count}') \in \mathtt{Verified} :$
$\mathtt{count}' < \mathtt{count}\}$
7 Let $\mathsf{Checked} = \{i \mid (i, \_) \in \mathtt{Verified}\} \setminus \mathsf{Corrupted}$
8 Let $\mathsf{Unchecked} = \{i \mid (i, \_, \_) \in \mathfrak{H} \wedge (i, \_) \notin \mathsf{C}\} \setminus \mathsf{Checked}$
9 Let $\mathsf{AllowedVotes}[i] = \{c \mid (i, \mathtt{count}, c) \in \mathfrak{H}$ s.t. $\forall (i, \mathtt{count}') \in \mathtt{Verified} :$
$\mathtt{count} \geq \mathtt{count}'\}$
10 If $\exists\, c^V_1, \ldots, c^V_{n_V}$ s.t. $c_j \in \mathsf{AllowedVotes}[i^V_j]$ where $\mathsf{Checked} = \{i^V_1, \ldots, i^V_{n_V}\}$
11 $\exists\, (i^U_1, c^U_1), \ldots, (i^U_{n_U}, c^U_{n_U})$ s.t. $i^U_j \in \mathsf{Unchecked}, c^U_j \in \mathsf{AllowedVotes}[i^U_j]$
with i^U_j different
12 $\exists\, c^B_1, \ldots, c^B_{n_B} \in \mathcal{C}$ s.t. $0 \leq n_B \leq |\mathsf{Corrupted}|$
13 s.t. $z = \tilde{\tau}(\{c^V_i\}_{i=1}^{n_V}) \star_R \tilde{\tau}(\{c^U_i\}_{i=1}^{n_U}) \star_R \tilde{\tau}(\{c^B_i\}_{i=1}^{n_B})$
14 Then return 0, otherwise return 1

$O\mathtt{cast}(i, c)$:
Let $\tau = \mathsf{Register}(i)$
Add $(i, \#\mathtt{tokens}(i), c)$ to $\mathfrak{H}$
Return $\mathsf{CastVote}(\tau, c)$

$O\mathtt{register}(i)$:
Let $\tau = \mathsf{Register}(i)$
Add $(i, \#\mathtt{tokens}(i))$ to $\mathfrak{C}$
return τ

Figure 4.7: In the verifiability game experiment $\mathtt{Exp}^{\mathtt{verif},b}_{\mathcal{A},\mathcal{D}}$, the adversary $\mathcal{A}$ has access to the oracles $O = \{O\mathtt{cast}, O\mathtt{register}\}$.

adversary can call two oracles: $O\mathtt{cast}(i, c)$ to request that honest voter i outputs a ballot for candidate c, and $O\mathtt{register}(i)$ to get a voting token for user i. Note that the oracle $O\mathtt{cast}()$ in the verifiability game is different to that of the ballot privacy game. This models *both corruption and coercion* of voter i. After a call to $O\mathtt{register}(i)$, voter i is considered corrupted until it casts an honest ballot using $O\mathtt{cast}(i, c)$. Eventually, the adversary outputs a bulletin board PBB, the selected votes $\mathcal{S}$ and proof θ, the election outcome $z \in \mathbb{R}$, and a tally proof Π_z (line 3). The adversary loses if θ or Π_z do not verify (line 4). If it verifies, the adversary wins if the result does not satisfy the three intuitive conditions above.

The game computes the following groups of voters:

- Corrupted (line 6): voters considered corrupted, i.e., voters that were once corrupted (by calling $O\texttt{cast}$) and thereafter never cast a checked honest vote.
- Checked (line 7): voters that verified a ballot and were not corrupted thereafter.
- Unchecked (line 8): voters that were never corrupted, but did not check their ballots either.

The game computes allowed candidates for honest voters:

- AllowedVotes$[i]$ (line 9): A list of candidates that voter i honestly voted for in or after the last checked ballot. If voter i never checked a ballot, this list includes all candidates this voter ever voted for.

The adversary wins if the result z verifies but violates any of the following conditions (lines 10–13): (1) For each honest voter that verified a ballot and was not thereafter corrupted (i.e., voters in Checked) the result should include either the candidate in that ballot, or a candidate in a later ballot. This corresponds to the candidates $\{c_i^V\}_{i=1}^{n_V}$ in the game. (2) Of the honest voters that did not check their ballots but were never corrupted (i.e., voters in Unchecked), at most one candidate that the honest voter voted for (in any ballot) can be included. This corresponds to the candidates $\{c_i^U\}_{i=1}^{n_U}$ in the game. Note that n_U can be smaller than $|\mathsf{Unchecked}|$ or in fact zero. (3) At most n_C corrupted (or bad) votes were counted (i.e., the candidates $\{c_i^B\}_{i=1}^{n_B}$).

In the game, the sum of these choices is modeled by the tallying function $\tilde{\tau}\colon \mathcal{C}^* \to \mathbb{R}$ that maps the voter's choices in $\mathcal{C}$ to an election result in $\mathbb{R}$. This function should support partial tallying, i.e., for any two lists S_1 and S_2 we have that $\tilde{\tau}(S_1 \cup S_2) = \tilde{\tau}(S_1) \star_R \tilde{\tau}(S_2)$ for a commutative binary operator $\star_R\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. Note that a tally function that outputs the number of votes per candidate naturally admits partial tallying.

Definition 5. Consider a voting scheme $\mathcal{V} = (\texttt{Setup}, \mathsf{Register}, \mathsf{CastVote}, \mathsf{VoteVerify}, \mathsf{Valid}, \mathsf{Filter}, \mathsf{VerifyFilter}, \mathsf{Tally}, \mathsf{Verify})$ for an electoral roll $\mathcal{E}$ and candidate list $\mathcal{C}$. We say the scheme is *verifiable* if for all probabilistic polynomial time adversary $\mathcal{A}$

$$\left|\Pr\left[\texttt{Exp}^{\texttt{verif},0}_{\mathcal{A},\mathcal{D}}(\epsilon,\mathcal{E},\mathcal{C}) = 1\right] - \Pr\left[\texttt{Exp}^{\texttt{verif},1}_{\mathcal{A},\mathcal{D}}(\epsilon,\mathcal{E},\mathcal{C}) = 1\right]\right|$$

is a negligible function in ϵ.

4.4.4 Coercion resistance

Coercion resistance ensures that a coercer should not be able to determine whether a coerced user submitted to coercion, other than the information it can learn by seeing the result of the election (e.g., if there are zero votes for the candidate selected by the coercer, the latter knows the coerced user escaped coercion).

Existing coercion resistant models are insufficient. Juels, Catalano and Jakobsson (JCJ) model coercion resistance by comparing a real-world game with an ideal game [JCJ05]. In JCJ, voters evade coercion by providing the coercer with a fake credential. The real-world models normal execution. The adversary plays the role of the coercer and chooses a set of corrupted voters and identifies the coerced voter. Then, the honest voters cast their ballots (or abstain). If the coerced voter does not submit she also casts her true ballot. Thereafter, the adversary is given the credentials of all corrupt users, a credential for the coerced voter (which is fake if that voter resists), and the current bulletin board. The adversary can now cast more ballots. Upon seeing the result and the tally proof the adversary decides if the coerced voter submitted. In the ideal game, the adversary is not shown the content of the bulletin board, and she is given the true credential of the coerced voter and can therefore cast real ballots for the coerced voter. However, a modified tally function does not count ballots for the coerced voter cast by the adversary if the coerced voter resists. Once the election phase is over, the adversary is shown only the tally result, not the tally proof.

The JCJ model does not work for the revoting setting where the coerced voter casts another ballot *after* casting the ballot under coercion. Achenbach et al. [Ach+15] propose a variant in which the coerced voter acts after the adversary has cast his votes, revoting if she resists or doing nothing if she submits. Thereafter, the adversary is shown the new bulletin board and the resulting tally and proof. In the ideal model, the adversary is only provided the length of the bulletin board.

The model proposed by Achenbach et al. [Ach+15] does not capture coercion resistance. Following the real/ideal paradigm, in the ideal game it should hold with overwhelming probability that the adversary cannot distinguish between a submitting and a resisting coerced voter. Then, the proof would show that the adversary cannot learn more in the real world than it could in the ideal world. However, in the ideal game proposed by Achenbach et al., the coercion resistance property does not hold. The adversary can *always* distinguish between these two cases by simply observing the length of the bulletin board (which increases by one ballot if the coerced voter revotes). Therefore, proofs in this model say nothing about whether the real scheme

$\mathtt{Exp}^{\mathtt{cr},b}_{\mathcal{A},\mathcal{D}}(\epsilon,\mathcal{E},\mathcal{C})$:
  $(pk, sk_{CA}, sk_{TS}, sk_v) \leftarrow \mathtt{Setup}(1^\epsilon, \mathcal{E}, \mathcal{C})$
  Create CA_0 and CA_1 with keys pk^0_{CA}, pk^1_{CA}
  $b' \leftarrow \mathcal{A}^O(pk, pk^b_{CA}, pk^{1-b}_{CA})$
  Output b'

$O\mathtt{voteDR}(i_0, \mathcal{C}_0, i_1, \mathcal{C}_1)$:
  Let $\tau_0 \leftarrow CA_0.\mathtt{ReqCred}(i_0)$ and $\tau_1 \leftarrow CA_1.\mathtt{ReqCred}(i_1)$
  Let $\beta_0 = v(\tau_0, \mathcal{C}_0)$ and $\beta_1 = v(\tau_1, \mathcal{C}_1)$
  If $\mathsf{Valid}(PBB_b, \beta_b) = \perp$ return $\perp$
  Else $PBB_0 \leftarrow PBB_0 \parallel \beta_0$ and $PBB_1 \leftarrow PBB_1 \parallel \beta_1$

$O\mathtt{register}(i)$:
  Let $\tau_0 \leftarrow CA_0.\mathtt{ReqCred}(i)$ and $\tau_1 \leftarrow CA_1.\mathtt{ReqCred}(i)$
  return $\tau = \tau_b, \tau' = \tau_{1-b}$

$O\mathtt{cast}(\beta, \beta')$:
  Let $\beta_b \leftarrow \beta$ and $\beta_{1-b} \leftarrow \beta'$
  If $\mathsf{Valid}(PBB_0, \beta_0) = \perp$ or $\mathsf{Valid}(PBB_1, \beta_1) = \perp$ return $\perp$
  Else $PBB_0 \leftarrow PBB_0 \parallel \beta_0$ and $PBB_1 \leftarrow PBB_1 \parallel \beta_1$

$O\mathtt{board}()$:
  return PBB_b

$O\mathtt{tally}()$
  Let L be the number of tokens obtained from CA_0.
  Let $\mathcal{S}_0, \theta_0 = \mathsf{Filter}(PBB_0, L, sk_{TS})$
  Let $(z, \Pi_{z,0}) \leftarrow \mathsf{Tally}(PBB_0 \parallel \mathcal{S}_0 \parallel \theta_0, sk_v)$
  Let $(\mathcal{S}_1, \theta_1) \leftarrow \mathtt{SimFilter}(PBB_1, L', z)$
  Let $PBB_0 \leftarrow PBB_0 \parallel \mathcal{S}_0 \parallel \theta_0$ and $PBB_1 \leftarrow PBB_1 \parallel \mathcal{S}_1 \parallel \theta_1$
  $\Pi_{z,1} = \mathtt{SimTally}(PBB_1, z)$
  return $(z, \Pi_{z,b})$

Figure 4.8: In the coercion resistance experiment $\mathtt{Exp}^{\mathtt{cr},b}_{\mathcal{A},\mathcal{D}}$, adversary $\mathcal{A}$ has access to oracles $O = \{O\mathtt{register}, O\mathtt{cast}, O\mathtt{board}, O\mathtt{tally}\}$. It can call $O\mathtt{tally}$ only once, thereafter it can see the result θ_b by using $O\mathtt{board}()$.

offers coercion resistance. The Achenbach et al. [Ach+15] scheme seems to be coercion resistant, but coercion resistance does not follow from the proof in their model.

Finally, the model by Achenbach et al. does not capture the leakage resulting from the state kept by the voter, or as in our protocol, by the polling authority. Our protocol deliberately hides the ballot counter from the voter, so that if the coercer coerces the voter again, it cannot determine whether the coerced voter re-voted based on this counter. Achenbach et al.'s model does not capture this property, as the coercer is not allowed to coerce a voter

more than once.

A new coercion resistance definition. We propose a new game-based coercion resistance definition inspired by Bernhard et al.'s ballot privacy definition. The game tracks two bulletin boards, PBB_0 and PBB_1, of which only one is accessible to the adversary (depending on the bit b). We ensure that regardless of the bit b, the same number of ballots are added to the bulletin board. The goal of the adversary is to determine b (see Figure 4.8). Recall that we assume that TS, and the trustees (and therefore the CA) are honest with respect to coercion resistance. We model honesty of the PBB (respectively, the use of an anonymous communication channel) by not revealing which voter posted to the bulletin board.

To model submits versus resists, we provide the adversary with an $O\texttt{voteDR}(i_0, \mathcal{C}_0, i_1, \mathcal{C}_1)$ oracle to let voter i_0, a "coerced" voter, cast a vote for candidate $\mathcal{C}_0$ in PBB_0, and voter i_1, any other voter, cast a vote for candidate $\mathcal{C}_1$ in PBB_1. The adversary is allowed to make this call multiple times. Regardless of the value of b, every call to $O\texttt{voteDR}$ results in a single ballot being added to each bulletin board. This prevents the trivial win in the Achenbach et al. model. Since the polling authority keeps state, we work with two CA: CA_0 and CA_1.

We model a coercion attack as follows. The adversary can cast votes using any user by calling $O\texttt{register}(i)$ to obtain a voting token τ for voter i on the board that it can see, and a token τ' for the other board. It can then run $\beta = v(\tau, \mathcal{C}_i)$ and $\beta' = v(\tau', \mathcal{C}_i)$ itself to create ballots for candidate $\mathcal{C}_i$, on both boards and cast them using $O\texttt{cast}(\beta, \beta')$. Note that per our assumptions, the adversary does not get access to the voter's means of authentication. Moreover, we require that the adversary always casts valid ballots to both boards (but the encoded candidate need not be the same).

Finally, the adversary can make one call to $O\texttt{tally}()$ which performs the filtering step and returns the result z (always computed on PBB_0) and the tally proof. To correct for leakage stemming from the tally result, as in the ballot privacy game, we simulate the filter and tally proofs if the adversary sees PBB_1.

This game models all the coercion attacks applicable to a re-voting protocol:

- *The 1009 attack.* The adversary casts a ballot as coerced voter i_0 using $\tau, \tau' = O\texttt{register}(i_0)$, $\beta = v(\tau, c)$, $\beta' = v(\tau', c)$ and then $O\texttt{cast}(\beta, \beta')$ 1009 times. (Both boards now contain 1009 ballots by voter i_0.) Then it calls $O\texttt{voteDR}(i_0, c, i_1, c)$. If $b = 0$ the coerced voter revotes for candidate c on PBB_0, otherwise it does not, and the alternative voter casts a ballot for candidate c on PBB_1 visible to the adversary. Note

that if the result of Filter reveals the size of a group of ballots, the adversary can win this game (`SimFilter` does not model this leakage as it only gets L' and z as input).

- *Returning coercer.* Let voter i_0 be the coerced voter. First the coercer runs $\tau, \tau' = \texttt{ReqCred}(i_0)$, $\beta = v(\tau, c)$ and $\beta' = v(\tau', c)$, and $O\texttt{cast}(\beta, \beta')$ to cast one vote as the coerced user on both boards and to observe the token τ corresponding to the board PBB_b it can see. Then it runs $O\texttt{voteDR}(i_0, c_0, i_1, c_1)$, causing i_0 to cast a vote on the bulletin board PBB_b if $b = 0$, and i_1 to casts a vote on PBB_b if $b = 1$. Thereafter, it can examine the state by running $\tau, \tau' = \texttt{ReqCred}(i_0)$ again. If the new token τ leaks whether voter i_0 voted again (on board PBB_b), then the adversary wins the coercion resistance game.

- *Italian attacks* [DC07]. Let voter i_0 be the coerced voter. Let VP be a long list of candidates unlikely to be selected by any other voter. The adversary then casts ballots for each of the candidates in VP in the name of the user. More precisely, The adversary runs $\tau, \tau' = O\texttt{register}(i_0)$, $\beta = v(\tau, c_j)$, $\beta' = v(\tau', c_j)$ for $c_j \in VP$. Then it calls $O\texttt{voteDR}(i_0, c, i_1, c)$. If $b = 0$ the coerced voter revotes for candidate c on PBB_0, otherwise it does not, and the alternative voter casts a ballot for candidate c on PBB_1 visible to the adversary. Note that if the adversary has access to all decrypted votes, it wins the game.

Definition 6. Consider a voting scheme $\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{CastVote}, \mathsf{VoteVerify}, \mathsf{Valid}, \mathsf{Filter}, \mathsf{VerifyFilter}, \mathsf{Tally}, \mathsf{Verify})$ for an electoral roll $\mathcal{E}$ and candidate list $\mathcal{C}$. We say the scheme has *coercion resistance* if there exist algorithms `SimFilter` and `SimTally` such that for all probabilistic polynomial time adversaries $\mathcal{A}$

$$\left|\Pr\left[\texttt{Exp}^{\texttt{cr},0}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C}) = 1\right] - \Pr\left[\texttt{Exp}^{\texttt{cr},1}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C}) = 1\right]\right|$$

is a negligible function in ϵ.

4.4.5 Strict coercion resistance

The definition of coercion resistance presented above, ensures that even a user that wants to submit to coercion (or sell her vote) is capable of doing so with negligible probability. However, in real world elections, the property of coercion might not be of the same importance as the one of vote-selling. In the case of NetVote, the protocol presented in Section 4.5, the vote buying can happen with low probability, but in order to achieve linear filtering we dispense the negligibility of it happening.

$\mathtt{Exp}^{\mathtt{scr},b}_{\mathcal{A},\mathcal{D}}(\epsilon,\mathcal{E},\mathcal{C})$:
  $(pk, sk_{CA}, sk_{TS}, sk_v) \leftarrow \mathtt{Setup}(1^\epsilon, \mathcal{E}, \mathcal{C})$
  $b' \leftarrow \mathcal{A}^O(pk, pk_{CA})$
  Output b'

$O\mathtt{voteDR}(i_0, \mathcal{C}_0, i_1, \mathcal{C}_1)$:
  Let $\tau_0 \leftarrow \mathsf{Register}(i_0)$ and $\tau'_1 \leftarrow \mathsf{RegisterDummy}(i_1)$
    $\tau_1 \leftarrow \mathsf{Register}(i_1)$ and $\tau'_0 \leftarrow \mathsf{RegisterDummy}(i_0)$
  Let $\beta_0 = \mathsf{CastVote}(\tau_0, \mathcal{C}_0)$ and $\beta'_1 = \mathsf{CastVoteDummy}(\tau'_1, 0)$
    $\beta_1 = \mathsf{CastVote}(\tau_1, \mathcal{C}_1)$ and $\beta'_0 = \mathsf{CastVoteDummy}(\tau'_0, 0)$
  If $\mathsf{Valid}(PBB_b, \beta_b) = \perp$ or
    $\mathsf{Valid}(PBB_b, \beta'_{1-b}) = \perp$ return $\perp$
  Else $PBB_0 \leftarrow PBB_0 \parallel \beta_0 \parallel \beta'_1$ and
    $PBB_1 \leftarrow PBB_1 \parallel \beta_1 \parallel \beta'_0$

$O\mathtt{register}(i)$:
  Let $\tau \leftarrow \mathsf{Register}(i)$
  return τ

$O\mathtt{cast}(\tau, \mathcal{C})$:
  If $\mathsf{Valid}(PBB_0, \beta) = \perp$ or
    $\mathsf{Valid}(PBB_1, \beta) = \perp$ return $\perp$
  Else $PBB_0 \leftarrow PBB_0 \parallel \beta$ and
    $PBB_1 \leftarrow PBB_1 \parallel \beta$

$O\mathtt{board}()$:
  return PBB_b

$O\mathtt{tally}()$
  Let L be the number of tokens obtained from CA_0.
  Let $\mathcal{S}_0, \theta_0 = \mathsf{Filter}(PBB_0, L, sk_{TS})$
  Let $(z, \Pi_{z,0}) \leftarrow \mathsf{Tally}(PBB_0 \parallel \mathcal{S}_0 \parallel \theta_0, sk_v)$
  Let $(\mathcal{S}_1, \theta_1) \leftarrow \mathtt{SimFilter}(PBB_1, L', z)$
  Let $PBB_0 \leftarrow PBB_0 \parallel \mathcal{S}_0 \parallel \theta_0$ and $PBB_1 \leftarrow PBB_1 \parallel \mathcal{S}_1 \parallel \theta_1$
  $\Pi_1 = \mathtt{SimTally}(PBB_1, z)$
  return (z, Π_b)

Figure 4.9: In the strict coercion resistance experiment, $\mathtt{Exp}^{\mathtt{scr},b}_{\mathcal{A},\mathcal{D}}(\epsilon,\mathcal{E},\mathcal{C})$, adversary $\mathcal{A}$ has access to oracles $O = \{O\mathtt{voteDR}, O\mathtt{cast}, O\mathtt{board}, , O\mathtt{register}, O\mathtt{tally}\}$. It can call $O\mathtt{tally}$ only once.

Current literature covers vote-selling as part of coercion resistance. To model such a difference we need to differentiate between coercion-resistance and vote-selling. To this end we need to modify the definition presented above.

We propose a new game-based coercion-resistance definition,

$\mathtt{Exp}^{\mathtt{scr},b}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C})$, that we define under the name *strict coercion-resistance*, inspired by the coercion resistance definition presented in the previous section. The game tracks two bulletin boards, PBB_0, PBB_1, of which the adversary has access to only one. The goal of this new notion is not that the adversary should not be able to distinguish between a voter resisting or submitting to coercion. Instead, we only aim to model that if the voter escapes coercion, the coercer cannot determine whether it decided to resist coercion and cast another vote, or if on the contrary, the voter decided to submit to coercion. The goal of the adversary is to determine which run of the experiment it is seeing (see Figure 4.8). To model this, we provide the adversary with the $O\mathtt{voteDR}(i_0, \mathcal{C}_0, i_1, \mathcal{C}_1)$ oracle, to make voter i_0 cast a vote for candidate $\mathcal{C}_0$, and voter i_1 cast a dummy vote for a dummy candidate, 0, in PBB_0. And, to make voter i_1 cast a vote for candidate $\mathcal{C}_1$, and voter i_0 cast a dummy vote for a dummy candidate, 0, in PBB_1. We use, $\mathsf{RegisterDummy}(i)$ to denote the dummy registration of voter i. Moreover, we denote by $\mathsf{CastVoteDummy}(\tau, \mathcal{C})$ the dummy vote cast for candidate $\mathcal{C}$ using token τ. The adversary is allowed to make this call multiple times.

Note that regardless of the call, two votes are added to both bulletin boards. This oracle models the situation we want to protect our voters from. If they wish to escape coercion, the adversary will not be able to distinguish that situation with one where a dummy vote is added. Note that in our schemes, the dummy votes are added once the election is closed. However, in the game we model the dummy vote addition in parallel to the voting stage to avoid the trivial win of the adversary. To this end, we allow the challenger to use two additional protocols to register and cast dummy votes. These protocols are $\mathsf{RegisterDummy}(i)$ and $\mathsf{CastVoteDummy}(\tau, 0)$, which create a token for and cast a dummy vote for user i respectively.

The rest of the oracles are modelled as for the coercion resistance experiment, with the difference that this game does not keep state of the voting tokens. Mainly $O\mathtt{register}(i)$, allows the adversary to register and obtain a token, τ, for voter $\mathcal{V}_i$; $O\mathtt{cast}(\tau, \mathcal{C})$, where using a token τ, the adversary can call this oracle to cast a vote for candidate $\mathcal{C}$ in PBB_0 and PBB_1; $O\mathtt{board}()$ allows the adversary to see the bulletin board; finally, $O\mathtt{tally}()$ allows the adversary to compute the tally of the election. It can call this oracle only once.

Again, as in the coercion resistance game, the result is always computed from the same bulletin board, to avoid leakage of information given by the result. To this end, the game simulates the tally and proofs of correctness in case the game is using PBB_1. We stress that this game models the 1009 attack only when the voter decides to escape coercion. The vote-selling where the voter decides to submit to the adversary and prove that it did not

resubmit is not hereby modeled. At the end of the game, the adversary needs to output a guess, b'. If it guesses correctly with non-negligible probability, it wins the game. We formally define $\texttt{Exp}^{\texttt{scr},b}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C})$ in Figure 4.9.

Definition 7. Consider a voting scheme $\mathcal{V} = (\texttt{Setup}, \mathsf{Register}, \mathsf{CastVote}, \mathsf{VoteVerify}, \mathsf{Valid}, \mathsf{Filter}, \mathsf{VerifyFilter}, \mathsf{Tally}, \mathsf{Verify})$. We say the scheme has strict coercion resistance if there exists algorithm `SimTally` such that for all probabilistic polynomial time adversaries $\mathcal{A}$

$$\left|\Pr\left[\texttt{Exp}^{\texttt{scr},0}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C}) = 1\right] - \Pr\left[\texttt{Exp}^{\texttt{scr},1}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C}) = 1\right]\right|$$

is negligible with respect to ϵ.

4.5 NetVote

This section presents the first protocol presented in this thesis in what concerns internet voting, namely NetVote. In this section we present a detailed explanation of the phases of the protocol, namely the pre-election, election and tally phases in Section 4.5.1. Then, Section 4.5.2 presents our dummy vote strategy. In Section 4.5.3 we prove that our scheme satisfies ballot privacy, practical everlasting privacy, verifiability and strict coercion resistance as presented in Section 4.4.

4.5.1 Scheme

Pre election

During the pre-election phase, the different parties generate their key pairs and publish the public key in PBB. Similarly, PBB publishes the list of candidates. Then, VS generates the random identifiers and commits to them by publishing their encryption. More formally:

> **Procedure 4.5.1** (`Setup`)**.** During the setup procedure, the different entities run $\texttt{Setup}(1^\epsilon, \mathcal{C}, \mathcal{E}, t, k)$, with security parameter ϵ, candidate list $\mathcal{C}$, electoral roll $\mathcal{E}$, threshold t and number of trustees k. They pick a group G with primer order p and generator g. Then they proceed as follows:
>
> 1. PBB publishes the list of candidates, $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{n_c}\}$, and initializes a counter $\texttt{count} = 0$.
> 2. CA and TS generate their key-pairs (pk_{CA}, sk_{CA}) and (pk_{TS}, sk_{TS}) respectively, by calling $\texttt{KeyGen}(1^\epsilon)$. They proceed by publishing their public keys in PBB.

3. The trustees and TS distributively run $\texttt{DistKeyGen}(1^\epsilon, t, k)$, where the voting key, pk_v, is generated, and each trustee and TS owns a share of the private key $sk_{v,i}$. They proceed by publishing the voting key, pk_v in PBB. Recall, as introduced in Section 4.2, that the collaboration of TS, (as well as t of k) is required to decrypt.

4. CA takes as input the total number of voters, $n_t = |\mathcal{E}|$, and generates random and distinct voting identifiers, $VId_i \leftarrow_\$ \mathbb{Z}_p$ for $1 \leq i \leq n_t$ with $VId_i \neq VId_j$ for $i \neq j$. It keeps the relation between VId and voter private. Next, it encrypts all identifiers:

$$w_i := \texttt{Enc}(pk_{TS}, VId_i) \text{ for } 1 \leq i \leq n_t.$$

Next, it signs each encrypted identifier, w_i:

$$\sigma_i = \texttt{Sign}(sk_{CA}, w_i).$$

Finally, it commits to these values by publishing the pair (w_i, σ_i) in PBB.

Election Phase

This phase comprises all steps that are taken while the election process is open. Note that a voter needs to follow a certification phase for each vote cast, which allows to avoid coercion without a high increase in complexity whilst simplifying the task for voters to cast votes multiple times from different devices. With the received certificate, `Cert`, the user signs their ballot β and sends it to the PBB. The later verifies the validity of the ballot, and the signature before posting it. More formally:

> **Procedure 4.5.2** ($\texttt{ReqCred}(id, \texttt{Auth})$)**.** The voter authenticates to the certification authority using her inalienable means of authentication, `Auth`, and requests an anonymous credential. As a response, she receives a one-time use anonymous certificate with a re-encryption, w_i', of the respective w_i as an attribute. Together with the certificate, $\texttt{Cert}(w_i')$, CA includes a proof of correct re-encryption of w_i.
>
> 1. The voter authenticates to CA using `Auth`.
>
> 2. CA selects the corresponding encrypted voter identifier, w_i, and randomizes the ciphertext by levering the homomorphic property:
>
> $$(\Pi_R, w_i') = \texttt{RandCtxt}(w_i, r_i),$$

for $r_i \leftarrow_\$ \mathbb{Z}_p$, to generate a randomization of the encrypted identifier, w_i', and a proof of correct randomization, Π_R.

3. CA generates an anonymous certificate, $\texttt{Cert}(w_i') = \texttt{CredGen}(\{w_i'\})$, with w_i' as an attribute, and sends it to the user together with w_i and the proof of re-encryption Π_R,

$$(\texttt{Cert}(w_i'), w_i, \Pi_R)\,.$$

4. The user verifies that the attribute of the certificate is an encryption of w_i by verifying Π_R. If it is not the first time it casts a vote, it may also verify that it has received the same w_i during both vote cast phases.

Procedure 4.5.3 ($\mathsf{Vote}(\texttt{Cert}, c)$)**.** The procedure of casting a vote proceeds as a 'regular vote-cast protocol'. Voter uses her voting credential, selects a candidate, encrypts it and proves correctness. It includes her encrypted identifier for a further filtering phase. It sends the tuple to the PBB. More precisely:

1. Voter encrypts the chosen candidate, $c \in \mathcal{C}$ and generates a proof of correctness by calling

$$(V, \Pi_V) = \texttt{VoteEnc}(pk_v, c)$$

and signs it using her ephemeral certificate, $\texttt{Cert}$. Let sk_C be the private key related to $\texttt{Cert}$, then the user computes

$$s = \texttt{Sign}(sk_C, V),$$

Finally, she sends to PBB her ballot,

$$\beta = (\texttt{Cert}(w_i'), V, \Pi_V, s)\,.$$

2. PBB verifies that the vote is valid running $\mathsf{Valid}(\beta)$ (see below). If everything verifies correctly, it publishes it and sends an acknowledge to the voter.

3. PBB augments the counter $\texttt{count} = \texttt{count} + 1$ and publishes the vote in the board

$$(\texttt{count}, \beta)\,.$$

4. Upon receiving the acknowledge, the voter can verify that her vote was recorded as cast. Moreover, any third party is able to check that all votes recorded in PBB come from a certified voter, the w'_i is related to the certificate and votes have a correct format.

Procedure 4.5.4 ($\mathsf{Valid}(\beta)$)**.** The PBB verifies that a ballot, $\beta = (\mathtt{Cert}(w'_i), V, \Pi_V, s)$ is valid before publishing it in the PBB. It verifies that the certificate was issued by the CA, the signature is a valid signature by $\mathtt{Cert}$, and the proof ensures that the encrypted vote corresponds to a valid candidate. More technically:

$$\begin{aligned}&\mathtt{CertVerify}(pk_{CA}, \mathtt{Cert}(w'_i)) = \top \wedge\\ &\mathtt{SignVerify}(pk_C, s) = \top \wedge\\ &\Pi_V.\mathtt{Verify}() = \top.\end{aligned}$$

Note that voters may repeat Procedure 4.5.2 and 4.5.3 as many times as they wish while the election is open, from different devices without needing to store or move the voting certificates.

Tallying phase

At this point, the election process is closed, and all votes from the voters that took place in the election (possibly more than one vote from some voters) are stored. Before proceeding to the tallying, the system needs to perform the filtering, i.e. keeping the vote with the highest counter for each voter. To this end, we make use of a proof determining whether $a > b$, with a, b being the counters of the objects in the PBB, without giving any other information of a, b. However, during the filtering procedure, we disclose the number of times each voter voted (even though these voters are anonymous). Therefore, to avoid the 1009 attack, we need to add dummy votes before the filtering procedure. We formally define these procedures and their verification counterparts.

Procedure 4.5.5 (Filter)**.** Our interest is to do the public filtering in a manner that a voter (or coercer) cannot know which of its votes has been accepted. However, it must be provable that only the vote with the highest by each individual voter passed the filtering. In order to do this, we use Bulletproofs [Bün+18] to prove that an encrypted counter is greater than another. The filter proof is formed by several distinct proofs, and distinct states of the votes (randomisations, shuffles, filters). Hence, we represent it as an appendable

list, θ, where all these proofs and states are added. At the end of the procedure, this parallel board is included in PBB.

1. The tally server, TS, extracts the ordered list of ballots, $[\beta_1, \ldots, \beta_L]$ from PBB. It then verifies each of the ballots by running $\mathsf{Valid}(\beta_i)$ for $i \in [1, L]$. If this fails, it adds $\perp$ to θ, appends it to PBB, and aborts.

2. TS follows by stripping the information from the ballot, keeping only the necessary information to proceed the filtering phase, mainly the counter, the encrypted identifier and the encrypted vote, $\beta' = (\texttt{count}, w'_i, V)$ and adds them to θ.

3. Next, the TS ads a random number of dummy voters per voter. In total, it ads n_D dummies. We describe how the TS chooses the number of voters to add per voter in Subsection 4.5.2. We want the counter of the dummy votes to be always less than the counters of real votes, and hence, TS always adds a dummy with a zero counter. Moreover, every added dummy uses the encrypted zero vector with zero randomness, $\texttt{VoteZEnc}(pk_v, 0)$. Finally, we don't want an adversary to know how many dummy votes were added to its coerced voter, so TS randomises the encrypted VId, $w''_i = \texttt{RandCtxt}(w'_i, r_i)$ for $r_i \leftarrow_\$ \mathbb{Z}_p$. Each added dummy vote has the following form,

$$\beta = (0, w''_i, \texttt{VoteZEnc}(pk_v, 0)).$$

   TS appends the dummy ballots to θ.

4. Next, TS encrypts each of the counters with randomness zero, $\texttt{EncCounter} = \texttt{Enc}(pk_{TS}, i)$ for every counter i (including the counters from dummy votes) resulting in each entry of PBB as follows:

$$(\texttt{EncCounter}_i, V_i, w'_i),$$

   which it ads to θ.

5. Now, TS proceeds by performing a verifiable shuffle to all the entries in the bulletin board. Let $\mathcal{B} = [\beta'_1, \ldots, \beta'_{L+n_D}]$ be the stripped ballots with the encrypted counter. It uses a provable shuffle to obtain a shuffled list of ballots, $\mathcal{B}_S = [\beta''_1, \ldots, \beta''_{L+n_D}]$ and a proof of correctness Π_S. It appends B' and Π_S to θ.

6. Next, it proceeds to group encrypted votes cast by the same voters. To achieve this it begins decrypting each identifier, $VId_i = \texttt{Dec}(sk_{TS}, w_i')$ and generating a proof of correct decryption, Π^d,

$$(\texttt{EncCounter}_i, V_i, VId_i, \Pi^d).$$

   Next, it groups ballots by VId_i. Each of these groups has size S_i^G. TS appends the ballots with decrypted VId in groups to θ.

7. The TS filters the votes by taking the encrypted vote with the higher counter. To do this it locally decrypts every counter and selects the one with the highest counter. It proceeds by publishing the filtered votes, together with $S_i^G - 1$ proofs that the counter is greater than all the other counters related to votes cast by the same voter, where S_i^G is the number of votes cast by voter $\mathcal{V}_i$. Note that the Bulletproof system that we use for this proof provides batch proofs, allowing TS to batch all these proofs at the cost of an additive factor (instead of a multiplicative factor) in the communication and computational complexity. It appends to θ

$$(\texttt{EncCounter}_i, V_i, VId_i, (\Pi_{GT,j})_{j \in S_i}),$$

   where S_i are the groups formed by votes cast with the same identifier VId_i.

8. Finally, the TS publishes the final list of votes $\mathcal{S} = [v_1, \ldots, v_{n_\mathcal{V}}]$ and the full proof θ to PBB.

Procedure 4.5.6 (Tally)**.** Finally, the TS calculates the full encrypted result by performing an addition of all ciphertexts accepted after the filtering

$$\texttt{Enc}(pk_v, v_{\texttt{Final}}) = \prod_{i=1}^{n_\mathcal{V}} \texttt{Enc}(pk_v, v_i) = \texttt{Enc}\left(pk_v, \sum_{i=1}^{n_\mathcal{V}} v_i\right).$$

Then, the result goes through the group of trustees T holding the different shares of the private key, together with the TS, which also holds a required share of the decryption key. They decrypt and generate the proof of correct decryption, Π_z. Moreover, any auditor or third party can calculate the sum of all the ciphertexts, and then verify that the result is a proper decryption of such a product.

Verification

Any external auditor can use the PBB to verify that all steps in the tally and filtering phases were performed correctly. We define the following verification procedures:

Procedure 4.5.7 (VerifyFilter)**.** Any party can verify that the filtering processes was performed correctly by running VerifyFilter(). This algorithm examines the content of the bulletin board and performs the following checks:

1. First, check if all ballots are correct and that no duplicate votes or public keys are included in the ballots as per step 1 of Procedure 4.5.5. If the checks fail, the bulletin board should contain $\theta = \perp$; VerifyFilter returns $\perp$ if that is not the case. Otherwise, it continues.

2. Retrieve the selected votes $\mathcal{S}$ and the proof θ from the bulletin board and continue as follows:

   (a) Verify that stripped real ballots are correctly formed. Consider ballots $[\beta_1, \ldots, \beta_L]$, where $\beta_i = (\texttt{count}_i, \texttt{Cert}(w_i'), V, \Pi_V, \sigma)$ and check that the stripped ballot $\beta_i' = (\texttt{count}_i, w_i', V)$ has been added to θ.

   (b) Verify that the dummy ballots on the bulletin board are correctly formed. For ballots $\beta_{L+1}', \ldots, \beta_{L+n_D}'$ where $\beta_i' = (\texttt{count}_i, w_i', V)$, check that $V_i = \texttt{VoteZEnc}(pk_v, 0)$ and $\texttt{count} = 0$.

   (c) Let $\texttt{EncCounter}_i$ be the encryption of counter $\texttt{count}_i$. For $\beta_1', \ldots, \beta_L'$, verify that $\texttt{EncCounter}_i = \texttt{Enc}(pk_{TS}, i)$ with randomness zero. Then, for $\beta_{L+1}', \ldots, \beta_{L+n_D}'$, verify that $\texttt{EncCounter}_i = \texttt{Enc}(pk_{TS}, 0)$, with randomness zero.

   (d) Let $\mathcal{B} = [\beta_1', \ldots, \beta_{L+n_D}']$ be all stripped ballots, and $\mathcal{B}_S = [\beta_1'', \ldots, \beta_{L+n_D}'']$ the shuffled and randomized ballots. Verify the shuffle proof Π_S to check that $\mathcal{B}_S$ is a correct shuffle of $\mathcal{B}$.

   (e) Next, let $[(VId_1, \Pi_1^d), \ldots, (VId_{L+n_D}, \Pi_{L+n_D}^d)]$ from the bulletin board, and verify the decryption proofs Π_i^d for each of the shuffled ballots β_i''.

   (f) Let VId_i' be the plaintexts verified in the previous step. Group the ballots by voter identifier into ballot groups G_j

of size S_j^G. For each group G_j, find the selected ballot, VId'_k, and verify the $S_i^G - 1$ proofs that the encrypted counter is greater than all other encrypted counters in the group.

If any of the checks fail, it returns $\perp$, and $\top$ otherwise.

Procedure 4.5.8 (Verify)**.** Any party can verify the result z and proof Π_z against the public bulletin board. To do so, they proceed as follows:

1. Verify that the TS operated honestly by running the VerifyFilter() algorithm. If VerifyFilter returns $\perp$, then return $\top$ if $(z, \Pi_z) = (\perp, \perp)$, otherwise return $\perp$.

2. Given the selected votes $\mathcal{S}$, return the result of $\texttt{VerifyTally}(pk_v, \mathcal{S}, z, \Pi_z)$.

4.5.2 Including dummy votes

The inclusion of dummy votes allows us to mitigate the 1009 attack in an elegant and simple manner. Undoubtedly, complexity increases, as we include more votes in the shuffle and the filtering stage. However, the number of dummy votes does not need to be big, and therefore complexity will only be increased by a small constant. In this section, we describe how the TS includes dummy votes once the election is closed and how this affects filtering complexity.

In order to hide the number of votes that each voter casts, we include a random number of votes per eligible voter. To see how it is not trivial to determine the number of dummies to add per voter, consider the following:

- It is straightforward to see why a fixed number of dummies for all voters would not serve the purpose here. So say that a random number of dummies for voter $\mathcal{V}_i$, $n_{i,dum}$, is added. If the set were we take $n_{i,dum}$ from is $\mathbb{Z}$, then the overhead would become prohibitive, as there is no upperbound. However, if $n_{i,dum} \leftarrow_{\$} \mathbb{Z}_u$, where u is the upperbound, then with probability $1/u$, $n_{i,dum} = u$ which would blow the wistle if voter $\mathcal{V}_i$ decided to re-vote to escape coercion (as now a total of $u + 1$ votes would have been added, where not all could have been dummies).

  Then, it seems necessary to have no upperbound, but instead of choosing $n_{i,dum}$ uniformly at random from $\mathbb{Z}_u$, we could use a distribution where $n_{i,dum} = n$ with probability $1/2^{n+1}$. With this distribution there

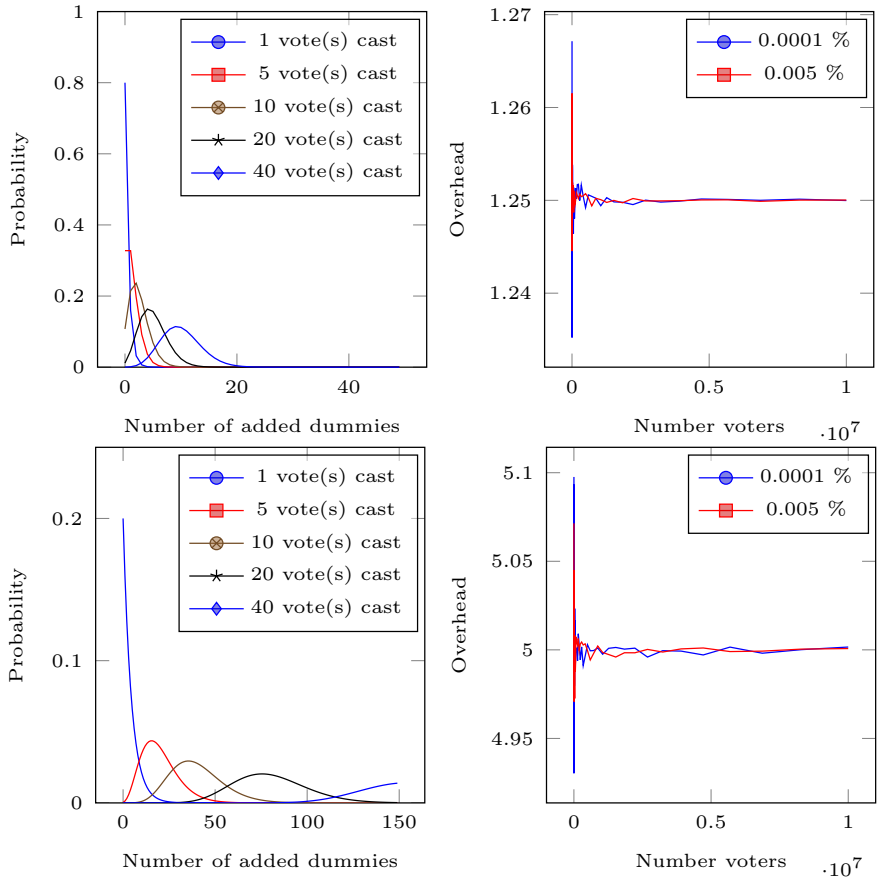


Figure 4.10: Negative binomial distribution (left) with probability of success 0.8 (top) and 0.2 (bottom). This distribution is used to select the number of dummies to add per voter, depending on how many votes a voter casts. Overhead caused by the dummy addition (right), having used the corresponding distributions (top and bottom), where the overhead is counted as $((\#votes + \#dummies)/\#votes)$.

is no upperbound but it is very unlikely to add a big number of dummies for voter $\mathcal{V}_i$. The drawback of this mechanism is that there is a lower bound. Moreover, with probability 1/2, zero dummy votes will be added. This is not a problem for a voter who wants to escape coercion, as re-voting would not reveal anything to the coercer. However, a voter that wants to submit would be able to prove so with probability 1/2, and therefore be able to sell its vote with high probability.

- However, while we want to hide the groups of votes with unusual group sizes (e.g. 1009), we do not need to add an overhead to small groups

(which are not prone to receive the 1009 attack). To this end, our solution adds random votes to voters depending on the votes they have cast, following the negative binomial distribution, defined as:

$$f(\mu;\rho,p) := \Pr[X=\mu] = \binom{\mu+\rho-1}{\mu}(1-p)^{\rho}p^{\mu}.$$

where ρ is the number of votes cast by the voter in question, and μ is the number of dummy votes to add for that user.

The choice of this probability distribution is made clear in Figure 4.10, left. This distribution results in adding, with high probability, a low number of dummy votes for voters that cast a small number of votes, and with very low probability adding zero dummy votes for voters who cast an unusual number of votes. This behaviour is exactly what we want to achieve for hiding voting patterns.

We present in Figure 4.10, left, the different distributions of the number of dummy votes added depending on how many votes a voter cast. In both top and bottom (left) we see the probability distribution of the number of dummies to add given a number of cast votes. In top, the probability of success is 0.8, while on the bottom it is of 0.2.

In order to understand how big the overhead is in the filtering phase, we present the overall overhead assuming that a subset of users re-voted. To define how the population of voters votes, we use the Poisson distribution to determine the number of votes cast by each voter. This distribution is defined by

$$P(k;\lambda) := \Pr[X=k] = \frac{\lambda^k e^{-\lambda}}{k!}.$$

Again, this distribution fits our goal with $\lambda = 1$, as we expect most users to vote once or twice, and consider the activity of casting many votes highly unlikely. Note that, with $\lambda = 1$, the probability of casting a high number of votes is low. However, we need to consider a subset of voters that casts lots of votes, either as an attack to the system or simply by users who are suffering coercion. To this end, we consider that a given percentage, say l, cast a very high number of votes, and therefore the number of re-votes are chosen at random from the Poisson distribution with $\lambda = 200$. In this case, the probability of casting a lot of votes is high. In Figure 4.10 right, we plot the overhead caused by dummies, $\frac{\#\text{ votes} + \#\text{ dummies}}{\#\text{ votes}}$, with a varying l between $0.1\% - 5\%$ of such users. Top and bottom represent the overhead using probability of success of 0.8 and 0.2 respectively. We can see that the overhead defined by ((# votes +# dummies)/# votes) remains constant in 1

and 5 respectively. This shows that the mechanism used to hide the patterns of whether voters revoted results in a minimal effect on the performance of the filtering phase.

4.5.3 Security Proofs

In this section we prove that NetVote satisfies the security properties introduced in Section 4.4.

Proof of Ballot Privacy

Theorem 5. NetVote provides ballot privacy, as defined in Definition 1, under the Decisional Diffie-Hellman (DDH) assumption in the random oracle model.

Proof. We provide a similar proof than the one used to prove that Helios [Adi08] has ballot privacy presented in [Ber+15] by using a sequence of games. We initialise the argument with $\texttt{Exp}^{\texttt{bpriv},0}_{\mathcal{A},\mathcal{D}}$ and go step by step until reaching a game equivalent to $\texttt{Exp}^{\texttt{bpriv},1}_{\mathcal{A},\mathcal{D}}$. By showing that each of the transitions between the steps are indistinguishable, we conclude that the two experiments are indistinguishable and hence NetVote satisfies ballot privacy.

Game G_0: Let G_0 be $\texttt{Exp}^{\texttt{bpriv},0}_{\mathcal{A},\mathcal{D}}$ as defined in Figure 4.3 where the adversary has access to the bulletin board PBB_0.

Game G_1: G_1 is defined exactly as G_0 with the exception that the tally proof is simulated. This is, the result is still computed from the votes in PBB_0, but the proof of tally is simulated. The proofs to be simulated are the shuffle proof in Step 5, the proofs of correct decryption in Step 6, and the proofs of greater or equal relation in Step 7, of Procedure 4.5.5. Given that all these proofs are zero-knowledge proofs, they require (in order to have the zero-knowledge property) the existence of a simulator algorithm that generates simulations of the proofs which are indistinguishable from real proofs. We use the random oracle to describe as such our `SimTally` algorithm (as introduced in Definition 1).

Let L be the number of votes published in the bulletin board. Note that, by how the oracles are defined, this number is equivalent in PBB_0 and PBB_1. We denote β_i the ballots posted in PBB_i for $i \in \{0, 1\}$. Next, we proceed with a series of L games where we change (one by one) each entry of posted votes. Let $G^0_1 = G_1$. For $i \in \{1, \ldots, L\}$, we do the following:

Game G_1^i: The difference between G_1^i and G_1^{i-1} is only one. If ballot β_0^i of PBB_0 differs from ballot β_1^i of PBB_1, it exchanges β_0^i by β_1^i.

Game G_2: We define G_2 as G_1^L. Note that this view is equivalent to the view of $\mathtt{Exp}_{\mathcal{A},\mathcal{D}}^{\mathtt{bpriv},1}$. Hence, all that remains to prove is that this set of transitions of games G_1^i are indistinguishable among each other.

In order to prove that the transitions made between games G_1^i are indistinguishable, we use the non-malleability under chosen plaintext attack (NM-CPA) property of the ElGamal cryptosystem [BPW12]. This property of ElGamal ensures that an adversary that chooses two plaintexts, and has a challenger encrypt them, is capable of distinguishing which ciphertext encrypts which plaintext with negligible probability. Recall that the ballots are formed by $(\mathtt{Cert}(w_i'), V, \Pi_V, \sigma)$, where the identifier and vote are encrypted. However, the process of changing each of the ballots will occur after TS has stripped them, as defined in Step 2 of Procedure 4.5.5, mainly, $(\mathtt{count}, w_i', V)$.

Hence, when changing two ballots between games G_1^i and G_1^{i+1}, we only need to change the encrypted vote V. We piggyback on this property of ElGamal encryption, and deduce that if an adversary is capable of distinguishing between two consecutive games G_1^i with non-negligible probability, then it is capable of breaking NP-CPA security of ElGamal.

This completes the proof that an adversary can distinguish between $\mathtt{Exp}_{\mathcal{A},\mathcal{D}}^{\mathtt{bpriv},0}$ and $\mathtt{Exp}_{\mathcal{A},\mathcal{D}}^{\mathtt{bpriv},1}$ with negligible probability. □

Proof of Everlasting privacy

Theorem 6. NetVote provides practical everlasting privacy, as defined in Definition 4.

Proof. As in the proof of Theorem 5, we begin defining a game corresponding to $\mathtt{Exp}_{\mathcal{A},\mathcal{D}}^{\mathtt{everbpriv},0}$, and proceed with indistinguishable changes until a game corresponding to $\mathtt{Exp}_{\mathcal{A},\mathcal{D}}^{\mathtt{everbpriv},1}$, therefore completing the proof.

Game G_0: This is defined as a run of $\mathtt{Exp}_{\mathcal{A},\mathcal{D}}^{\mathtt{everbpriv},0}$ as defined in Figure 4.6, where the adversary has access to the bulletin board PBB_0.

Game G_1: This game is defined exactly as G_0 with the sole exception that now, we change the register phase, and instead run: $\tau_1 = \mathsf{Register}(\mathcal{V}_0)$ and $\tau_0 = \mathsf{Register}(\mathcal{V}_1)$

Note that G_1 already corresponds to $\mathtt{Exp}_{\mathcal{A},\mathcal{D}}^{\mathtt{everbpriv},1}$. In order to prove that these two events are statistically (and not computationally) indistinguishable, we recall how the Register phase is defined in NetVote. A voter $\mathcal{V}_i$ uses

its authentication credentials (e.g. usr/pwd) to request a voting certificate, `Cert`() to CA. This certificate contains an encrypted voting identifier w_i unique to the voter. However, these identifiers are chosen at random during the `Setup` protocol, and the link between voter and VId is private to CA (i.e. it is not published in the PBB). Hence, while the adversary is capable of decrypting the identifier in each of the credentials, it is not capable of determining whether a given VId corresponds to $\mathcal{V}_0$ or $\mathcal{V}_1$, as they are chosen at random at each oracle call.

This completes the proof that NetVote provides practical everlasting privacy. □

Proof of Verifiability

Theorem 7. NetVote is verifiable, as defined in Definition 5, under the DDH assumption in the random oracle model.

Proof. We provide a similar proof by showing that dummy votes are not counted in the final tally and that at least one cast vote has been counted (one later than the last verification). At the end of the election, the adversary outputs z, Π. Because Π validates, we know that the result z is the addition of the filtered votes from the PBB. Now it remains to show that the filtered votes do indeed satisfy the conditions imposed by the game.

Let B be the stripped ballots in PBB once the election closes. We know that all these votes originate from a valid ballot cast by a voter, and hence are accompanied by a proof that they contain an encryption of a valid candidate.

Let $n_{\mathcal{V}}$ be the number of different voters that cast at least a ballot. We argue that the number of ballots included in the tally equals $n_{\mathcal{V}}$. Because of the correctness of the shuffle, Π_S, we know that these same ballots will be present after the shuffle, and hence votes of voters $1, \ldots, n_{\mathcal{V}}$ are present in B'. The filtering procedure groups votes cast by the same voter, and takes only the vote with the highest counter (where the counter is unique per PBB entry starting at one). Because of the validity of the decryption proof, Π^d, we know that the filtering is applied to votes cast by the same voter (recall that we assume the honesty of CA for verifiability). Moreover, given the correctness of the *greater than* proofs, $\Pi_{GT,i}$ for i ranging all indexes of votes cast by the same voter, we know that only the ballot with higher index was counted for each voter. Finally, all dummy votes are cast with non encrypted zero counter, and therefore it is impossible that a dummy vote supersedes a real vote cast by a voter.

Hence, we know that votes counted in the final tally are the votes with the highest counter recorded in the PBB by each voter that took part in the

election. These voters are part of one of the three groups of our definition. What remains is to prove that the conditions are met for each of these three groups.

First, we show that the last checked vote or one after is counted for each voter in `Checked`. Consider voter $\mathcal{V}_i \in$ `Checked`, and let $\texttt{count}_v$ be the counter of the last ballot it checked was recorded as cast. We know that ballot with counter $\texttt{count}_v$ was added to PBB, therefore in the grouping after the shuffle, we now that the group for voter $\mathcal{V}_i$ will exist, with at least ballot with counter $\texttt{count}_v$. Given that the selected ballot of each group is accompanied with a proof that the counter is greatest among the group, the selected ballot *must* be the one with counter $\texttt{count}_v$ or one cast afterwards. Given the proof of decryption of the homomorphic tally among all selected votes, we know that either ballot $\texttt{count}_v$ or a later ballot by voter $\mathcal{V}_i$ is counted in the final tally.

Now consider a voter $\mathcal{V}_i \in$ `Unchecked`. Then, by the same argument as above, we know that the tally either drops all ballots, or counts one of the ballots cast by the voter. In other words, the tally cannot add a vote not cast by voter $\mathcal{V}_i$.

Finally, all remaining voters correspond to group `Corrupted`. Notice that by the arguments above, the tally procedure cannot include any votes by voters who did not cast a vote. Moreover, only one vote per grouped votes is selected. Hence, it follows that the size of this group is a most the number of corrupted voters, $n_{\mathfrak{C}}$. □

Proof of Strict coercion resistance

Theorem 8. NetVote has strict coercion resistance, as defined in Definition 7, under the DDH assumption in the random oracle model.

Proof. As in the proof of Theorem 5, we construct our `SimTally` algorithm by leveraging the zero-knowledge property of the zero-knowledge proofs used in the tally. Namely, `SimTally` simulates the proofs of shuffle, decryption, greater than, and finally, the proof of decryption of the added votes (Procedures 4.5.5 and 4.5.6).

We present a proof following a series of games, replacing all the ballots that depend on the bit b. If all steps used throughout the replacement of these ballots are indistinguishable, strict coercion resistance follows.

Game G_1: This game is defined as $\texttt{Exp}^{\texttt{scr},b}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C})$ of Figure 4.8. Note that we differ from the proof of ballot privacy in that we do not start with a fixed value of b.

Game G_2: Later, we are going to replace all votes by random votes. To this end, in this game, we compute the result taking the decrypted ballots of PBB_0. Consider the stripped ballots of step 4 of Procedure 4.5.5, $(\texttt{EncCounter}, V, w')$. It computes the tally of these ballots, $\mathsf{Tally}((\texttt{EncCounter}, V, w')_{i=1}^{n_l})$, where n_l is the total number of votes cast. It does so by first decrypting $\texttt{EncCounter}, V, w'$ using sk_{TS}, sk_v and sk_{TS} respectively. Now, it proceeds by computing the final result by taking the last vote cast per VId_i. Note that, as the result is always taken from PBB_0 and the game does not publish these decrypted values, game G_2 is indistinguishable from game G_1.

Game G_3: Same as the previous game, but now all zero-knowledge proofs, regardless of b, are replaced by simulations. This is, the proof of shuffle, Π_S, of all votes (including the dummies), the proof of correct decryption, Π^d, of each identifier, VId_i, the greater than proofs used to filter all but the last vote, $\Pi_{GT,i}$, and finally, the proof of decryption of the final result, Π_z.

Due to the zero-knowledge property, we can use the random oracle to simulate this step, making the difference between game G_2 and G_3 indistinguishable.

Our goal now is to exchange all identifying ciphertexts (mainly w', `EncCounter`, and the corresponding values after the shuffle) by random ciphertexts. Note that the proofs are simulated, and the result (filter and tally) is calculated from the decrypted votes of game G_2, so the decryption, shuffle, and tally are now independent of the actual values of the encrypted votes.

Game G_4: We define G_4 the same as G_3 with the exception that we exchange all ciphertexts w' in the certificates by random ciphertexts. Note that due to the changes of G_2, the result is correctly calculated from the votes initially in PBB_0 and hence this does not affect the filtering stage. A hybrid argument reduces the indistinguishability of games G_4 and G_3 to the CPA security of ElGamal encryption. Note that this reduction is possible as we no longer need to decrypt the ciphertexts in the tallying.

Game G_5: We follow by replacing all votes cast by $O\texttt{voteDR}()$ by zero votes. Given that we assume TS is honest, and hence the adversary cannot decrypt the ciphertexts, the indistiguishability of this step is reduced to the NM-CPA security of ElGamal.

Game G_6: To ensure that the filter does not leak information to the coercer, we also replace all ciphertexts generated thereafter. We replace the

encryption of the counters, `EncCounter`, by random ciphertexts. We do the same with all ciphertexts that exit the shuffle. Namely, we replace the shuffled encrypted counters, encrypted votes and encrypted $VIds$. This exchange is possible as we do not need to decrypt the ciphertexts (as of game G_2). Moreover, the indistinguishability of this steps follows from the simulation of the zero-knowledge proofs (of game G_3) and the NM-CPA security of the encryption scheme.

The resulting game is clearly independent of b. Due to indistiguishable changes that we made to achieve this last game, we conclude that game G_1, and therefore $\mathtt{Exp}^{\mathtt{scr},b}_{\mathcal{A},\mathcal{D}}(\epsilon, \mathcal{E}, \mathcal{C})$, are independent of the game bit b, and hence strict coercion resistance follows. □

4.6 VoteAgain

This section presents our second protocol introduced in this thesis in what concerns internet voting, namely VoteAgain. Contrarily to the previous section, this protocol has stronger security properties with respect to coercion resistance in exchange of a quasi linear filtering phase. We begin the section pointing out the main differences between the two approaches in Section 4.6.1. We present a detailed explanation of the phases of the protocol, the pre-election, election and tally phases in Section 4.6.3, and an explanation of the deterministic dummy strategy in Section 4.6.4. In Section 4.6.5 we proof that our scheme satisfies ballot privacy, verifiability and coercion resistance as presented in Section 4.4. Finally, in Section 4.6.6, to prove that this increase in the computation complexity does not affect the usage of the scheme in large scale elections, we include an evaluation section.

4.6.1 Overview

The two key observations for possible improvement involve the anonymous credentials, the counter reference and the dummy strategy. The first is straightforward - we need to trust the CA both for coercion resistance and for everlasting privacy, therefore, it is not necessary to use anonymous credentials. By using simple tokens signed by the CA, we achieve the desired properties. Secondly, the counter reference makes an important difference. Have the scheme set up as presented in Section 4.5, the filtering procedure takes the token of the last vote cast, regardless of when the certificate was generated. This allows an adversary of collecting several tokens of many users, and use the last 5 minutes of the election to cast as many votes as possible. By changing the reference over which the TS performs the filtering,

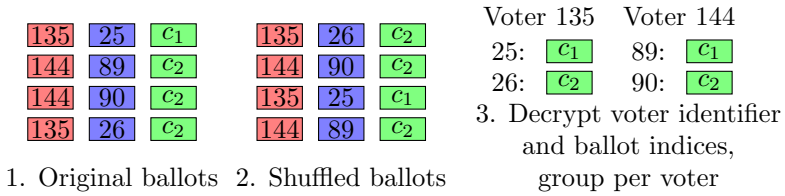


Figure 4.11: Basic filtering process by tally server without using dummies. Ballots consist of an encrypted voter identifier (▭), an encrypted ballot index (▭), and an encrypted vote (▭).

we can easily mitigate this attack. In VoteAgain, we include the counter in the voting token, not in the PBB. This ensures that the selected vote is the one cast with the last generated certificate, rather than the actual last vote.

We dedicate the rest of this section to focus on the third point as it requires a more thorough explanation, namely the new dummy strategy. Changing the dummy strategy allows us to enforce the security properties at the cost of minimally deteriorating performance. For simplicity, we omit the zero-knowledge proofs that parties use to show that they performed operations correctly.

4.6.2 A different dummy strategy

One of the first differences is that, in this new protocol counters are assigned to the certificate generation, rather than to the vote submission. This ensures that the last generated certificate is the one tallied in the final result. This prevents an attack from an adversary that collects plenty of certificate of coerced users, and submits the vote just before the end of the election. The dummy addition by the TS happens during the filtering procedure in the tallying phase. The encrypted voter identifiers and ballot indices enable the tally server (TS) to efficiently select the last ballot for each voter. The TS uses the simplest mechanism possible: It shuffles the ballots, and then decrypts the voter identifiers and ballot indices. The ballots can then publicly be grouped per voter, and the last ballot can be identified by inspection. Finally, the trustees tally the ballot with the highest counter of each voter. See Figure 4.11.

As presented previously, the random dummy generation procedure achieved good performance, but affected the security. In VoteAgain, TS inserts a deterministic number of *dummy ballots* and *dummy voters* before shuffling the ballots to hide such patterns while maintaining the simple public filtering procedure.

We illustrate VoteAgain's dummy mechanism in Figure 4.12, in a scenario with two voters (A and B) where, the coercer forces voter A to cast 2 ballots. At the end of the election phase the coercer observes 4 ballots and must determine whether A revoted (situation 2) or not (situation 1). Without dummies, distinguishing these situations is trivial: if A revoted there is a group of 3 ballots and one of 1 ballot, and there are two groups of 2 ballots otherwise. We add dummy ballots and voters to make both situations look identical. The idea is to find a *cover* of ballots that could result from both situations. For instance, adding to either situation two dummy voters that cast four dummy ballots total yields groups of 1, 2, 2, and 3 ballots. This observation makes both situations indistinguishable for the coercer (Figure 4.12, right).

To ensure that the cover is *independent* from the voters' real actions, its appearance must depend *only* on the information available to the coercer: (1) the number of ballots L posted by users to the bulletin board; and (2) the number of voters n_t that cast a ballot. The goal of the dummy generation strategy is to allocate dummy ballots such that the adversary observes the *same cover* regardless of the actual distributions of the L ballots over n_t voters.

Consider the case of two voters, i.e., $n_t = 2$, and 9 ballots, i.e., $L = 9$. As the filtering stage only reveals the sizes of the groupings and not their relation to voters the adversary's possible observations are $(1, 8), (2, 7), (3, 6)$, and $(4, 5)$. To cover all these scenarios one needs 8 voters (6 of which are dummy) casting $1, 2, 3, 4, 5, 6, 7$, and 8 ballots, for a total of $36 - 9 = 27$ dummy ballots.

We add dummy ballots to real voters as well to reduce the number of group sizes that are possible. For example, in the previous scenario one can pad the cases $(1, 8), (2, 7), (3, 6), (4, 5)$ to $(1, 8), (2, 8), (4, 8), (4, 8)$. This can be covered with a cover containing voters with $1, 2, 4, 8$ ballots each. Building this cover requires only 2 dummy voters and $15 - 9 = 6$ dummy ballots. We stress that *the number of added dummy ballots is independent of how the real ballots are actually distributed among the two voters.*

We refer to Section 4.6.4 for a generic, efficient algorithm for computing a cover.

Filtering with dummies. Before shuffling the ballots, the TS adds dummy ballots to achieve the desired grouping. To ensure that the TS cannot modify the election outcome, the TS assigns different tags to real and dummy ballots. different encrypted tag.

To determine how to add dummies, the TS inspects the decrypted voter identifiers and ballot indices; determines a cover; and then computes how

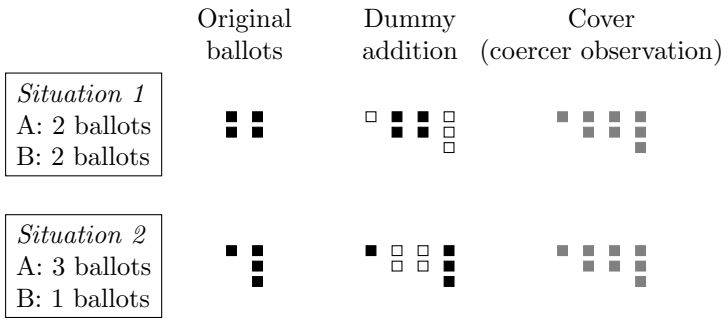


Figure 4.12: The original ballots' groups (■) create distinguishable situations. Adding 2 dummy voters casting a total of 4 dummy ballots (□), the situations become indistinguishable.

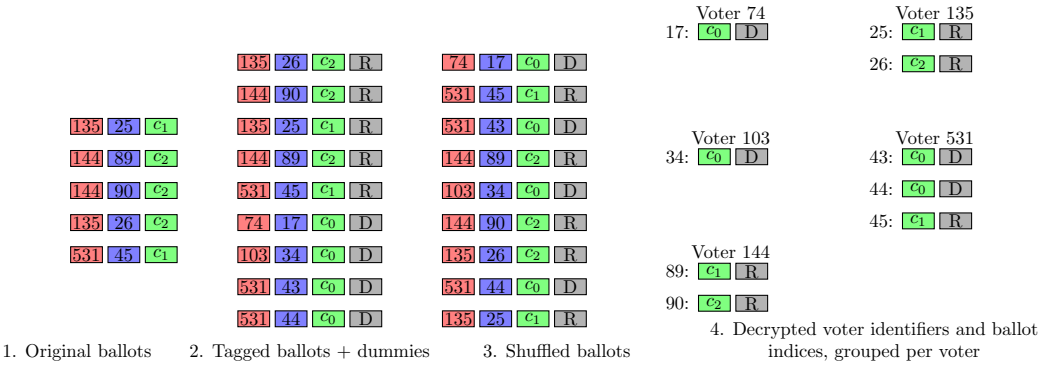


Figure 4.13: Filtering process by tally server including dummies. Labels as in Figure 4.11. To enable correctness proofs, the TS tags real ballots and dummy ballots with an encrypted marker (▭).

many dummies to add to existing voters, and how many dummies to add to dummy voters. Consider the example in Figure 4.13. Given 3 voters and 5 ballots, a cover with groups of size 1,1,2,2, and 3 suffices. The TS therefore adds 4 dummy ballots in step 2: 2 dummies to existing voter 531, and two dummy voters, 74 and 103, each with one dummy vote.

After adding the dummy ballots, the TS shuffles all ballots. Next, the TS decrypts the voter identifiers and ballot indices; groups ballots per voter, and selects the last ballot per voter. The tags enable the TS to prove that it did not omit real ballots cast by real voters, and it did not count dummy votes cast by dummy voters. In particular, the TS proves in zero-knowledge that the selected votes are either tagged as a real vote and therefore must correspond to the last ballot of a real voter; or the selected vote corresponds to a dummy voter (i.e., all the ballots in the group are tagged as dummies). Finally, the TS privately discards the selected votes corresponding to dummy

voters. We refer the reader to Procedure 4.6.5 for the full details.

4.6.3 Scheme

Pre-election

In the pre-election phase, the PBB publishes the candidates, and the TS and the trustees prepare their cryptographic material. The CA assigns a unique, random voter identifier VId_i to each eligible voter. The correspondence between voters and their identifiers is private to the CA. The CA also generates a random token index $\texttt{count}_i$ for each of the voters to enable the selection of the last ballot per voter. More formally:

> **Procedure 4.6.1** (`Setup`)**.** To setup an election system with security parameter ϵ, electoral roll $\mathcal{E}$, candidate list $\mathcal{C}$, threshold t, and k trustees, the different entities run the $\texttt{Setup}(1^\epsilon, \mathcal{E}, \mathcal{C}, t, k)$ procedure. First, they pick a group G with generator g and prime order p. They then proceed with the following steps:
>
> 1. The PBB initializes the bulletin board, and adds the list of candidates $\mathcal{C}$ to the bulletin board.
>
> 2. The CA stores the electoral roll $\mathcal{E}$. Let $n_t = |\mathcal{E}|$ be the number of eligible voters on the electoral roll. The CA generates a random and unique voter identifier $VId_i \in G$ and ballot index $\texttt{count}_i \in \{2^{\epsilon-2}, \ldots, 2^{\epsilon-1} - 1\}$ for each voter $\mathcal{V}_i$ on the electoral roll and stores them internally. Finally, the CA generates a public-private key-pair $(pk_{CA}, sk_{CA}) = \texttt{KeyGen}(1^\epsilon)$ to sign tokens. It publishes pk_{CA}.
>
> 3. The TS generates a public-private ElGamal key-pair $(pk_{TS}, sk_{TS}) = \texttt{KeyGen}(1^\epsilon)$. It publishes pk_{TS}.
>
> 4. The trustees run $\texttt{VoteKeyGen}(1^\epsilon, t, k)$ to generate a public encryption key pk_v and decryption keys $sk_{v,i}$ that the trustees keep private.

Election phase

In the election phase, voters first authenticate to the CA to obtain an ephemeral voting token τ. They use this token to sign their ballot β, and post the ballot on the bulletin board. The bulletin board verifies that the ballot is valid. We formalize this phase in three procedures:

Procedure 4.6.2 ($\texttt{GetToken}(id, \texttt{Auth})$)**.** On input her identity id and her inalienable means of authentication $\texttt{Auth}$:

1. The voter authenticates to the CA using $\texttt{Auth}$.

2. The CA looks up the corresponding voter identifier VId_i and ballot index $\texttt{count}_i$. Then, the CA encrypts the voter identifier $w = \texttt{Enc}(pk_{TS}, VId_i)$ and ballot number $\texttt{EncCounter} = \texttt{Enc}(pk_{TS}, \texttt{count}_i)$ (it first encodes $\texttt{count}_i$ as an element of G), and increments the ballot index $\texttt{count}_i := \texttt{count}_i + 1$. The CA hides the index $\texttt{count}_i$ from the user to prevent coercers – who can see what users can see under coercion – from being able to detect whether the user revoted.

3. The CA creates an ephemeral signing key $(pk, sk) = \texttt{KeyGen}(1^\epsilon)$ and signs this key together with the encrypted voter identifier and ballot number:

$$\sigma = \texttt{Sign}(sk_{CA}, pk \parallel w \parallel \texttt{EncCounter})$$

   and returns the token $\tau = (pk, sk, w, \texttt{EncCounter}, \sigma)$ to the user.

4. The user verifies the token $\tau = (pk, sk, w, \texttt{EncCounter}, \sigma)$ by checking that $\texttt{SignVerify}(pk_{CA}, \sigma, pk \parallel w \parallel \texttt{EncCounter}) = \top$.

Procedure 4.6.3 ($\mathsf{Vote}(\tau, \mathcal{C})$)**.** To cast a vote, the voter takes as private input the ephemeral voting token $\tau = (pk, sk, w, \texttt{EncCounter}, \sigma)$ and a candidate $c \in \mathcal{C}$, and then proceeds as follows:

1. Encrypts her candidate c as $(V, \Pi_V) = \texttt{VoteEnc}(pk_v, c)$ to obtain ciphertext V and zero-knowledge proof of correct encryption Π_V.

2. Creates the ballot

$$\beta = (V, \Pi_V, pk, w, \texttt{EncCounter}, \sigma, s)$$

   where $s = \texttt{Sign}(sk, V \parallel \Pi_V \parallel pk \parallel w \parallel \texttt{EncCounter} \parallel \sigma)$. The voter posts the ballot β to the public bulletin board.

3. The public bulletin board runs $\mathsf{Valid}(\beta)$, see below, to check that the ballot is valid, before appending it.

4. Finally, the voter verifies that the ballot β has been appended to the bulletin board.

Procedure 4.6.4 (Valid(β))**.** The bulletin board verifies that the ballot $\beta = (V, \Pi_V, pk, w, \texttt{EncCounter}, \sigma, s)$ is valid with respect to the current state of the bulletin board as follows:

1. The PBB checks the correctness of the encrypted vote; of the user's signature using the ephemeral key pk; and the CA's signature on this ephemeral key pk, the encrypted voter identifier w, and the encrypted ballot number `EncCounter`:

$$\begin{aligned}&\Pi_V.\texttt{Verify}(pk_v, V, \Pi_V) = \top,\\ &\texttt{SignVerify}(pk, s, V \parallel \Pi_V \parallel pk \parallel w \parallel \texttt{EncCounter} \parallel \sigma) = \top,\\ &\texttt{SignVerify}(pk_{CA}, \sigma, pk \parallel w \parallel \texttt{EncCounter}) = \top.\end{aligned}$$

2. The PBB checks that neither the encrypted vote V nor the key pk appear in any ballot β' on the bulletin board.

If any of these checks fails, the bulletin board returns $\perp$, otherwise, the PBB returns $\top$.

Tally phase

In the tally phase, the TS takes the ballots from the PBB, adds dummy ballots, and shuffles them. Then, it selects the vote with the highest counter per voter (see Figure 4.14). To prevent dummy voters from causing overhead in the trustees' shuffle and decrypt phase, the TS shuffles the selected ballots and removes all ballots cast by dummy voters. Contrarily to NetVote, VoteAgain hides the counters from adversaries, hence TS needs to tag the dummy ballots to ensure that these do not overwrite real ballots. Finally, the trustees shuffle and decrypt the selected ballots from real voters. Formally, we define two procedures, one to filter votes (Filter), and one to tally the selected ballots (Tally):

Procedure 4.6.5 (Filter)**.** After the election closes, the TS selects the selected votes vi and produces the filter proof θ. If it aborts, it publishes the current θ to the public bulletin board.

1. TS retrieves an ordered list of ballots $[\beta_1, \ldots, \beta_L]$ from the PBB, where $\beta_i = (V_i, \Pi_V, pk_i, w_i, \texttt{EncCounter}_i, \sigma_i, s_i)$. The TS verifies the ballots by running step 1 of Valid and verifies that there are are no duplicate votes V_i or ephemeral public keys pk_i

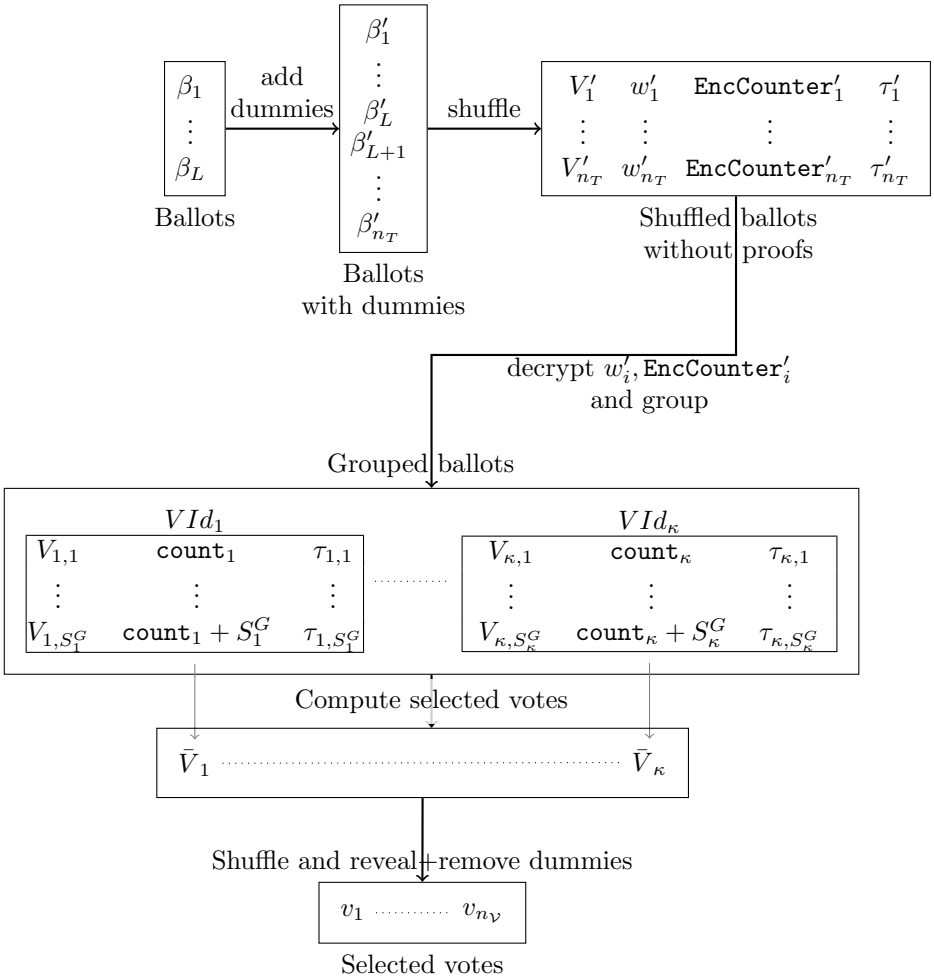


Figure 4.14: High-level overview of ballot filtering and grouping. Let L be the number of ballots, n_D be the number of dummies, $n_T = L + n_D$ be their sum, κ be the number of voters plus number of dummy voters, and S_i^G be the number of (dummy) ballots for (dummy) voter i. First, the TS adds dummy ballots and proves they are well-formed. Then shuffles all ballots without the proofs, hiding which ballots were dummies. Then it verifiably decrypts both the encrypted voter identifiers w'_i and the encrypted indices $\texttt{EncCounter}'_i$ to group the ballots by VId and to select the votes with the highest counters $\bar{V}i$. Finally, it outputs the selected votes vi without dummies.

on the bulletin board. If any of these checks fails, the TS sets $\theta = \perp$, posts it to the bulletin board, and aborts.

2. The TS removes the proofs and signatures to obtain stripped ballots. It provably tags the ballots as 'real' ballots using a deterministic ElGamal encryption (with randomness zero) of the value $g^0 = 1_G$, $\tau_R = \texttt{Enc}(pk_{TS}, g^0) = (g^0, g^0 pk^0) = (1_G, 1_G)$:

$$\beta'_i = (V_i, w_i, \texttt{EncCounter}_i, \tau_R).$$

Next, the TS creates n_D dummy ballots and provably tags them as such using a deterministic ElGamal encryption of the value g, $\tau_D = \texttt{Enc}(pk_{TS}, g) = (1_G, g \cdot pk^0)$:

$$\beta'_i = (V_\epsilon, w_i, \texttt{EncCounter}_i, \tau_D),$$

where $i > L$ and $V_\epsilon = \texttt{VoteZEnc}(pk_v; 0)$. We explain below how the TS determines the number of dummies n_D as well as the values for w_i and $\texttt{EncCounter}_i$. The TS adds the stripped ballots $\mathcal{B} = [\beta'_1, \ldots, \beta'_{L+n_D}]$ to θ.

3. The TS shuffles the stripped ballots $\mathcal{B} = [\beta'_1, \ldots, \beta'_{L+n_D}]$ and randomizes the ciphertexts, to obtain a list of shuffled and randomized stripped ballots $\mathcal{B}_S = [\beta''_1, \ldots, \beta''_{L+n_D}]$, which it adds, together with a proof Π_S that this shuffle was performed correctly, to θ.

4. The TS now operates on each shuffled ballot $\beta''_i = (V'_i, w'_i, \texttt{EncCounter}'_i, \tau'_i)$. It decrypts w'_i to recover the shuffled and decrypted identifier VId_i. It also decrypts $\texttt{EncCounter}'_i$ to obtain the shuffled ballot index $\texttt{count}_i$ and proves it did so correctly:

$$\begin{aligned}\Pi^d = SPK\{(sk_{TS}) : pk_{TS} = g^{sk_{TS}} \wedge \\ VId_i = \texttt{Dec}(sk_{TS}, w'_i) \wedge \\ \texttt{count}_i = \texttt{Dec}(sk_{TS}, \texttt{EncCounter}'_i)\}\end{aligned}$$

It then adds $\mathcal{I} = [(VId_1, \texttt{count}_1, \Pi^d), \ldots, (VId_{L+n_D}, \texttt{count}_{L+n_D}, \Pi^d)]$ to θ. The TS aborts and adds $\perp$ to θ if the decrypted ballot indices $\texttt{count}_i$ are not unique for a given voter identifier. More precisely, it aborts if there exists indices $i, j; i \neq j$ such that $(VId_i, \texttt{count}_i) = (VId_j, \texttt{count}_j)$.

5. The TS groups the ballots with the same voter identifier, and selects the ballot with the highest ballot index from each group. Let $G_1, \ldots, G_\chi$ be the sets of ballot indices grouped by voter identifier. Consider group G_j of size S_i^G. Let $j* = \mathtt{max}_{k,k\in G_j}\mathtt{count}_k$ be the index for which the ballot index $\mathtt{count}_{j*}$ is maximal. Group G_j either corresponds to a real voter, or to a fake voter. The TS produces a reencryption $\bar{V}_j$ of the encrypted votes as follows:

   (a) If the group G_j corresponds to a real voter, then the TS simply reencrypts the vote corresponding to the ballot with the highest counter, i.e., it picks r_j at random and sets
   $$\bar{V}_j = V_{j*} \cdot \mathtt{VoteZEnc}(pk_v; r_j),$$
   to a randomized encryption of V_{j*}.

   (b) If the group G_j corresponds to a fake voter, then picks r_j at random and sets $\bar{V}_j$ to an empty vote:
   $$\bar{V}_j = \mathtt{VoteZEnc}(pk_v; r_j).$$

   The TS proves that it computed the $\bar{V}_j$ correctly. If the corresponding voter is real, then the ballot β''_{j*} selected in (a) should be a real ballot, so $\mathtt{Dec}(sk_{TS}, \tau'_{j*})$ should equal g^0. If the voter is fake, then for all tags τ'_{i_k} with $i_k \in G_j$, we have that $\mathtt{Dec}(sk_{TS}, \tau'_{i_k}) = g^1$. Let $G_j = \{i_1, \ldots, i_{S_j^G}\}$ and $\tau = \prod_{k=1}^{S_j^G} \tau'_{i_k}$, then the TS constructs the proof

$$\begin{aligned}\Pi_R^i = SPK\{(r_j, sk_{TS}) : pk_{TS} = g^{sk_{TS}} \wedge \\ ((g^0 = \mathtt{Dec}(sk_{TS}, \tau'_{j*}) \wedge \bar{V}_j = V_{j*} \cdot \mathtt{VoteZEnc}(pk_v; r_j)) \vee \\ (g^{S_j^G} = \mathtt{Dec}(sk_{TS}, \tau) \wedge \bar{V}_j = \mathtt{VoteZEnc}(pk_v; r_j)))\}.\end{aligned}$$

   The TS adds the list of filtered encrypted votes $v_F = [(VId_1, \bar{V}_1, \Pi_R^1), \ldots, (VId_\chi, \bar{V}_\chi, \Pi_R^\chi)]$ to θ.

6. The list $\mathcal{S}_D = [\bar{V}_1, \ldots, \bar{V}_\chi]$ of selected votes contains ballots by dummy voters. In the next two steps, the TS removes these. First, the TS shuffles and randomizes the ciphertexts to obtain a new list $\mathcal{S}'_D = [\bar{V}'_1, \ldots, \bar{V}'_\chi]$, which it adds, together with a proof Π'_S of correct shuffle, to θ.

7. The TS knows the indices i_D of votes in $\mathcal{S}'_D$ that correspond to dummy voters and randomizers r_i such that $\bar{V}'_i = \texttt{VoteZEnc}(pk_v; r_i)$ for $i \in i_D$. The TS adds i_D and $r_o = [r_i]_{i \in i_D}$ to θ.

8. Finally, the TS publishes the remaining votes $\mathcal{S} = [v_1, \ldots, v_{n_\mathcal{V}}]$ and the full proof θ to the public bulletin board.

The filter procedure ensures that the TS cannot replace ballots by real voters: a selected vote must either correspond to a ballot by a real voter (condition a) or the selected vote is empty and the voter is a dummy voter (condition b). Moreover, the TS can only remove votes cast by dummy voters.

Procedure 4.6.6 (Tally)**.** To compute the final tally, the trustees proceed as follows:

1. The trustees verify that the TS operated honestly by running the VerifyFilter() algorithm (see below). If VerifyFilter returns $\perp$ they return $(z, \Pi_z) = (\perp, \perp)$.

2. Let $\mathcal{S} = [v_1, \ldots, v_{n_\mathcal{V}}]$. The trustees jointly run the $(z, \Pi_z) \leftarrow \texttt{Mix}(pk_v, \mathcal{S})$. They publish the election result z and the zero knowledge proof of correctness Π_z to the public bulletin board.

Verification

Any external auditor can use the PBB to verify that all steps in the tally and filtering phases were performed correctly. We define the following verification procedures:

Procedure 4.6.7 (VerifyFilter)**.** Any party can verify that the filtering processes was performed correctly by running VerifyFilter(). This algorithm examines the content of the bulletin board and performs the following checks:

1. First, check if all ballots are correct and that no duplicate votes or public keys are included in the ballots as per step 1 of Filter. If the checks fail, the bulletin board should contain $\theta = \perp$; VerifyFilter returns $\perp$ if that is not the case. Otherwise, it continues.

2. Retrieve the selected votes $\mathcal{S}$ and the proof θ from the bulletin board and continue as follows:

(a) Verify that stripped real ballots are correctly formed. Consider ballots $[\beta_1, \ldots, \beta_L]$, where $\beta_i = (V_i, \Pi_V, pk_i, w_i, \texttt{EncCounter}_i, \sigma_i, s_i)$ and check that the stripped ballot $\beta'_i = (V_i, w_i, \texttt{EncCounter}_i, \tau_R)$ has been added to θ (where τ_R is as above).

(b) Verify that the dummy ballots on the bulletin board are correctly formed. For ballots $\beta'_{L+1}, \ldots, \beta'_{L+n_D}$ where $\beta'_i = (V_i, w_i, \texttt{EncCounter}_i, \tau_i)$, check that $V_i = V_\epsilon$ and $\tau_i = \tau_D$ (where V_ϵ and τ_D are as above).

(c) Let $\mathcal{B} = [\beta'_1, \ldots, \beta'_{L+n_D}]$ be all stripped ballots, and $\mathcal{B}_S = [\beta''_1, \ldots, \beta''_{L+n_D}]$ the shuffled and randomized ballots. Verify the shuffle proof Π_S to check that $\mathcal{B}_S$ is a correct shuffle of $\mathcal{B}$.

(d) Next, let $\mathcal{I} = [(VId_1, \texttt{count}_1, \Pi^d), \ldots, (VId_{L+n_D}, \texttt{count}_{L+n_D}, \Pi^d)]$ from the bulletin board, and verify the decryption proofs Π^d for each of the shuffled ballots β''_i.

(e) Let VId'_i and $\texttt{count}'_i$ be the plaintexts verified in the previous step. Group the ballots by voter identifier into ballot groups G_j. For each group G_j, find ballot β_{j*} with the highest ballot index, recompute $\tau = \prod_{k=1}^{S_j^G} \tau_{i_k}$, and verify the reencryption proof Π_R.

(f) Let $\mathcal{S}_D$ be the selected votes $[\bar{V}_1, \ldots, \bar{V}_\chi]$ and $\mathcal{S}'_D = [\bar{V}'_1, \ldots, \bar{V}'_\chi]$ the shuffled and randomized votes. Verify the shuffle proof Π'_S for $\mathcal{S}_D$ and $\mathcal{S}'_D$.

(g) Finally, for each $i \in i_D$ verify that $\bar{V}'_i = \texttt{VoteZEnc}(pk_v; r_i)$ and that $\mathcal{S} = [\mathcal{S}_D[i] \mid i \notin i_D]$.

If any of the checks fail, it returns $\perp$, and $\top$ otherwise.

Procedure 4.6.8 (Verify)**.** Any party can verify the result z and proof Π_z against the public bulletin board. To do so, they proceed as follows:

1. Verify that the TS operated honestly by running the VerifyFilter() algorithm. If VerifyFilter returns $\perp$, then return $\top$ if $(z, \Pi_z) = (\perp, \perp)$, otherwise return $\perp$.

2. Given the selected votes $\mathcal{S}$, return the result of $\texttt{VerifyTally}(pk_v, \mathcal{S}, z, \Pi_z)$.

4.6.4 Hiding revoting patterns with dummies

In this section we provide a formal description of the dummy generation algorithm introduced in Section 4.6.1.

Finding a cover. Formally, a cover is a set $\mathcal{C} = \{(\int_i, z_i)\}_i$ formed by groupings $(\int_i, z_i) \in \mathbb{Z}^+ \times \mathbb{Z}^+$. Here, $\int_i$ is the size of the ballot groups within that grouping, and z_i is the upper bound on the number of times that such a ballot group can occur in any distribution of the L real ballots among real voters. We aim to find a cover of minimal size $|\mathcal{C}| = \sum_i \int_i \cdot z_i$ to minimize the number of dummies added.

A sufficient cover. We derive an upper bound on the amount of dummies required to build a cover. We do not use the number of real voters for this bound. Let L be the number of real ballots on the PBB. For simplicity, assume padded group sizes are powers of two, i.e., $\int_i = 2^i$ for $i \geq 0$. Given L ballots, any distribution can have at most $z_0 = L$ groups of size $\int_0 = 1$ (one ballot per voter). Similarly, any distribution can have at most $z_1 = \lfloor L/2 \rfloor$ groups of size $\int_1 = 2$. Recall we pad ballot groups to the next bigger size, so a ballot group of 3 would be padded to one of size $\int_2 = 4$ ballots, therefore $z_2 = \lfloor L/3 \rfloor$. More generally, there can be at most $z_i = \lfloor L/(2^{i-1}+1) \rfloor$ groups of $\int_i = 2^i$ ballots. The biggest possible group (if all ballots were cast by the same voter), has size $2^{\lceil \log_2 L \rceil}$. Therefore, the size of the cover $|\mathcal{C}|$ is bounded by:

$$
\begin{aligned}
|\mathcal{C}| &= \sum_{i=0}^{\lceil \log_2 L \rceil} z_i \cdot \int_i = L + \sum_{i=1}^{\lceil \log_2 L \rceil} 2^i \left\lfloor \frac{L}{2^{i-1}+1} \right\rfloor \\
&\leq L + \sum_{i=1}^{\lceil \log_2 L \rceil} \frac{2^i}{2^{i-1}+1} L \leq L + \sum_{i=1}^{\lceil \log_2 L \rceil} 2L = (1 + 2\lceil \log_2 L \rceil) L.
\end{aligned}
$$

An efficient cover. Knowing the number of real voters n_t enables to obtain a tighter cover. Consider the example of Section 4.6.1 with $n_t = 2$ and $L = 9$. If we only consider $L = 9$, one of the possible distributions of votes would be having $\int_1 = \lfloor 9/2 \rfloor = 4$ groups of size 2. However, knowing $n_t = 2$ rules out this possibility. There can be at most one group of size two: if there were 2 groups, each of the 2 voters could only cast 2 ballots, i.e., 4 ballots in total. However, we know there are 9 ballots so at least one voter has voted more than twice, implying that $\int_1 = 1$.

When the number of ballots grows this reasoning becomes intractable. Consider ballot groups with group sizes, $\int = S^i$ for $i \in [0, \ldots, \lceil \log_S L \rceil]$ for a real number $S > 1$. We assume that $L > n_t$, otherwise the cover would be trivial: $\mathcal{C} = \{(\int_0 = 1, z_0 = n_t)\}$. We compute the cover as follows.

1. Consider groups of size $\int_0 = k^0 = 1$. As $L > n_t$, at least one voter must cast more than one ballot, resulting in $(\int_0, z_0) = (1, n_t - 1)$.
2. Consider groups of size $\int_i = S^i$. We know that given L, there can be at most $\alpha_i = \lfloor L/(S^{i-1}+1) \rfloor$ groups of size S^i. The number of groups is also bound by the number of voters. If $n_t \cdot \int_i \geq L$ then all ballots can be assigned to the n_t voters given groups of maximum size $\int_i$, and we set $n_{t*} = n_t$, otherwise set $n_{t*} = n_t - 1$ so that one voter is not in this grouping. Finally, we need at least $z_i(S^{i-1}+1)$ ballots to make z_i groups, but we must have enough ballots left over to make n_t groups in total, i.e., $L \geq z_i(S^{i-1}+1) + (n_t - z_i)$. Rewriting gives bound $\beta_i = \lfloor (L - n_t)/S^{i-1} \rfloor$. We set $z_i = \min(\alpha_i, n_{t*}\beta_i)$.

Assuming $L > n_t$, the cover has $|\mathcal{C}| = \sum_{i=0}^{\lceil \log_S L \rceil} z_i \int_i > L$ ballots, necessitating dummy ballots, and $\sum_{i=1}^{\lceil \log_S L \rceil} z_i > n_t$ groups, necessitating dummy voters.

Creating dummy voters and allocating dummy ballots. The TS recovers all voter identifiers VId by decrypting the w_is, and the corresponding ballot indices by decrypting the $\texttt{EncCounter}_i$s.

So far, we assumed that ballot index sequences are continuous. However, there can be gaps if some tokens were not used (e.g., the coercer does not use some tokens to identify index gaps in the filtering phase). The TS first requests the number of obtained tokens L' from the PA, and adds exactly $L' - L$ dummy ballots to fill up any gaps, such that L' equals the number of obtained tokens. The TS can create a dummy ballot for voter VId by setting $w = \texttt{Enc}(pk_{TS}, VId)$.

Given the current number of ballots L' and the number of real voters n_t the TS computes a cover $\mathcal{C} = \{(\int_i, z_i)\}_i$. To this end the TS performs a search to find the best k, i.e., the one that gives the smaller cover. In our experiments in Section 4.6.6, k tends to be in the 2 to 4 range, and the search takes less than a second. The TS performs the following steps:

1. For every voter $VId_j, j \in \{1, \ldots, n_t\}$ with t ballots, let $(\int_i, z_i) \in \mathcal{C}$ be the cover group with the smallest size $\int_i$ such that $\int_i \geq t$. To ensure that dummy ballots are never counted, the TS adds $t - \int_i$ dummy votes to VId_j with descending (and unused) ballot counters *smaller* than the last cast vote by this voter.
2. For each grouping $(\int_i, z_i) \in \mathcal{C}$ let z'_i be the number of real voters that were assigned to this group. The TS adds $z_i - z'_i$ dummy voters. For each dummy voter, it picks a random VId and initial ballot index `count` and creates $\int_i$ dummy ballots with increasing ballot indices.

The algorithms Filter and VerifyFilter are quasilinear in the number of real ballots L. The TS first adds n_D dummies, so that the bulletin board contains

a total of $n_T = L + n_D = O(L \log L)$ ballots (see the bound above). All other steps in Filter and Verify filter are linear in n_T. The claim follows.

4.6.5 Security Analysis

In this section we prove that VoteAgain satisfies ballot privacy, verifiability and coercion resistance as defined in Section 4.4

Proof of Ballot privacy

Theorem 9. VoteAgain provides ballot privacy under the DDH assumption in the random oracle model.

Proof. This proof is very similar to the proof of ballot privacy of Helios in the full version of Bernhard et al. [Ber+15]. We start with the adversary playing the ballot privacy game with $b = 0$ and after a sequence of game steps transitions, the adversary finishes playing the ballot privacy game with $b = 1$. We argue that each of these steps are indistinguishable, and therefore the results follows. The proof proceeds along the following sequence of games:

Game G_0: Let game G_0 be the $\mathtt{Exp}^{\mathtt{bpriv},0}_{\mathcal{A},\mathcal{D}}$ game (see Figure 4.3 and Definition 1).

Game G_1: Game G_1 is as in G_0 but we now compute

$$\Pi_0 = \mathtt{SimProof}(PBB_0, z)$$

by simulating the proof using the random oracle instead of using the real proof from $\mathsf{Tally}(PBB_0, sk_v)$. Because of the simulation properties of the zero-knowledge proof system, $\mathcal{A}$ cannot distinguish these two games.

Game G_2: As in game G_1, but now $O\mathtt{tally}(\mathcal{S}, \theta)$ ignores $\mathcal{S}$ and θ provided by $\mathcal{A}$ when computing the result z. In particular, $O\mathtt{tally}$ now proceeds as follows:

$O\mathtt{tally}(\mathcal{S}, \theta)$
  If $\mathsf{VerifyFilter}(PBB_b, \mathcal{S}, \theta) = \bot$ return $\bot$
  $(z, \Pi_0) \leftarrow \mathsf{Tally}(PBB_0 \parallel \mathsf{Filter}(PBB_0, L', sk_{TS}), sk_v)$
  $PBB_b \leftarrow PBB_b \parallel \mathcal{S} \parallel \theta$
  $PBB_{1-b} \leftarrow PBB_{1-b} \parallel \mathsf{Filter}(PBB_{1-b}, L',\ tssk)$
  $\Pi_0 = \mathtt{SimProof}(PBB_0, z)$
  $\Pi_1 = \mathtt{SimProof}(PBB_1, z)$
  return (z, Π_b)

The proofs included in θ ensure that $\mathcal{A}$ honestly computed the filtering step. Therefore, the adversary's view is indistinguishable from that in G_1.

Game G_3: As in game G_2, but in $O\texttt{board}$ we return PBB_1. Note that in G_3 the adversary has the same view as in the $\texttt{Exp}^{\texttt{bpriv},1}_{\mathcal{A},\mathcal{D}}$ game. All that is left to show is that G_2 and G_3 are indistinguishable.

We now show that no adversary $\mathcal{A}$ can distinguish G_2 from G_3. Let N_O be the number of $O\texttt{cast}(\tau, \mathcal{C}_0, \mathcal{C}_1)$ calls that the adversary $\mathcal{A}$ made. In particular, for the ith call to $O\texttt{cast}$, remember the tuple $(\beta_0, \beta_1, \mathcal{C}_0, \mathcal{C}_1)$ of candidates and resulting ballots. We now build a series of games $H_0, \ldots, H_{N_O}$ and proceed by a hybrid argument.

In game H_i we show to the adversary a bulletin board where the first i ballots cast using $O\texttt{cast}$ on PBB_0 are replaced by those of PBB_1. More precisely, in all games H_i we keep track of an additional bulletin board PBB that is shown to the adversary, i.e., $O\texttt{board}$ now returns PBB. Whenever the adversary makes an $O\texttt{cast}(\beta)$ query, we also add β to PBB, i.e., $PBB \leftarrow PBB \parallel \beta$. In game H_i in response to the first i calls to $O\texttt{cast}$, we additionally set $PBB \leftarrow PBB \parallel \beta_1$. For the remaining calls we additionally set $PBB \leftarrow PBB \parallel \beta_0$. Note that $H_0 = G_2$ and that $H_{N_O} = G_3$.

We reduce to the NM-CPA security (see Figure 2.1) of the encryption scheme to show that H_i is indistinguishable from H_{i-1}. To show this, we create an adversary $\mathcal{B}$ against NM-CPA. Internally, $\mathcal{B}$ uses adversary $\mathcal{A}$. Adversary $\mathcal{B}$ receives the public key pk from its challenger. At the start of the game $\mathcal{B}$ runs $\texttt{Setup}$ as normal, but instead it sets $pk_v = pk$. It then answers the jth $O\texttt{cast}(\tau, \mathcal{C}_0, \mathcal{C}_1)$ query as follows:

- For $j < i$ it sets $PBB \leftarrow PBB \parallel \beta_1$
- For $i = i$ it returns $\mathcal{C}_0, \mathcal{C}_1$ to the NM-CPA challenger to receive a challenge ciphertext c^*, and uses that ciphertext when running v to obtain a ballot β^* and set $PBB \leftarrow PBB \parallel \beta^*$.
- For $j > i$ it sets $PBB \leftarrow PBB \parallel \beta_0$

Thereafter $\mathcal{B}$ answers the $O\texttt{tally}$ query as follows. It cannot directly compute the tally, as it does not know the decryption key sk_v. However, it knows sk_{TS} so it can recompute the $\mathsf{Filter}(PBB_0, L', sk_{TS})$ to determine which ballots $\beta_{i_1}, \ldots, \beta_{i_\chi}$ on PBB_0 should be included in the final tally (recall that the result is always computed on PBB_0, and that as per G_2 we do not use θ provided by the adversary). Then proceed as follows: Let $\Gamma = (V_{i_1}, \Pi_V), \ldots, (V_{i_\chi}, \Pi_V)$ be the corresponding vote ciphertexts and proofs. Then, $\mathcal{B}$ computes the result z as follows:

- If $c^* \in \Gamma$, then the ballot for candidate $\mathcal{C} = \mathcal{C}_0$ in query i should be included in the tally as well. Recall that the tally always is computed over PBB_0, therefore, $\mathcal{B}$ sets $(\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_{\chi-1}}) = O_D(\Gamma \setminus \{c^*\})$ and sets
$$z = \tilde{\tau}(\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_{\chi-1}}, \mathcal{C}_0).$$
- Otherwise, $\mathcal{B}$ sets $(\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_\chi}) = O_D(\Gamma \setminus \{c^*\})$ and sets
$$z = \tilde{\tau}(\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_\chi}).$$

Finally, as in game G_2, $\mathcal{B}$ simulates the tally proof. Note that if $b = 0$ in $\mathcal{B}$'s NM-CPA game, then $\mathcal{B}$ perfectly simulates H_{i-1}, and if $b = 1$ then it perfectly simulates H_i. Therefore, any distinguisher between H_i and H_{i-1} breaks the NM-CPA security of the voting scheme.

A standard hybrid argument now shows that $H_0 = G_2$ is indistinguishable from $H_{N_O} = G_3$. This completes the proof. □

Furthermore, we prove that VoteAgain provides strong-consistency and strong-correctness with the following theorem.

Theorem 10. VoteAgain provides strong-consistency and strong-correctness.

Proof. This proof roughly follows that of the strong consistency and strong correctness of Helios in the full version of Bernhard et al. [Ber+15]. To show that VoteAgain is strongly consistent, we define the following `Extract` and Valid algorithms:

- $\texttt{Extract}(\beta, sk_{TS}, sk_v)$ operates on a ballot $\beta = (V, \Pi_V, pk, w, \texttt{EncCounter}, s, \sigma)$. First, it verifies the proof Π_V, and the signatures s and σ as in step 1 of Valid in Procedure 4.6.4. If any check fails, it returns $\perp$. Otherwise, it recovers the candidate $\mathcal{C} = \texttt{Dec}(sk_v, V)$ (note that $\mathcal{C} \in \mathcal{C}$ because Π_V is valid). Then it decrypts w and `EncCounter` to get $(VId, \texttt{count}) = (\texttt{Dec}(sk_{CA}, w), \texttt{Dec}(sk_{CA}, \texttt{EncCounter}))$. It returns $((VId, \texttt{count}), \mathcal{C})$.
- $\mathsf{Valid}(\beta)$ proceeds as in step 1 of Valid in Procedure 4.6.4 to verify the ballot:
$$\begin{aligned}&\Pi_V.\texttt{Verify}(pk_v, V, \Pi_V),\\ &\texttt{SignVerify}(pk, s, V \parallel \Pi_V \parallel pk \parallel w \parallel \texttt{EncCounter} \parallel \sigma),\\ &\texttt{SignVerify}(pk_{CA}, \sigma, pk \parallel w \parallel \texttt{EncCounter}).\end{aligned}$$
It returns $\top$ if all are valid, and $\perp$ otherwise.

First we show that the first condition of strong consistency is satisfied. By the correctness of the zero-knowledge proofs and decryption algorithms, `Extract`will indeed extract the required values for valid ballots.

Since Valid executes a strict subset of the checks in Valid, it follows that the second condition is trivially satisfied.

For the third condition, we need to show that the adversary cannot create a valid bulletin board PBB (i.e., one on which Filter and Tally do not fail), but where the result is incorrect (respect to the output calculated with the extractor function).

Note that by the checks in steps 1 and 4 of Procedure 4.6.5, we know that the identifier pairs $(VId, \texttt{count})$ are unique. Consider the group G_j of ballots corresponding to VId_j. The ideal result function ρ includes the vote where the ballot index is highest. In exactly the same way, Filter sets $\bar{V}_j$ to V_{j*} where the index $j*$ maximizes the ballot index $\texttt{count}_{j*}$. The equivalence of the ideal result and the result produced by tally now follows.

To show that VoteAgain is strongly correct we need to prove that an adversary cannot create a ballot box PBB such that an honest ballot by an honest voter will be rejected, i.e., $\mathsf{Valid}(PBB, \beta) = \perp$. Note that the verification in $\mathsf{Valid}(\beta)$ is twofold. First, it verifies the validity of the ballot. It is trivial to see that this check passes for an honestly generated ballot. Second, it checks that the ephemeral public key pk and encrypted vote V do not yet appear on the bulletin board. Clearly, V does not appear because it was just generated honestly by the user. Moreover, neither does the public key pk appear before, because it was just freshly generated by the CA. Given that these two values contain a source of randomness when generated, it proceeds that $\mathcal{A}$ can only win with negligible probability. □

Proof of Coercion Resistance

Theorem 11. VoteAgain provides coercion resistance under the DDH assumption in the random oracle model.

Proof. We first specify how to construct `SimProof` and `SimFilter`. As in the ballot privacy proof, $\texttt{SimProof}(PBB, z)$ simply simulates the proof of shuffle and the proof of correct decryption in Tally, so that regardless of the values in $\mathcal{S}$, z is the correct outcome.

The algorithm $\texttt{SimFilter}(PBB, L', z)$ proceeds similarly. It takes as input the bulletin board PBB, which it uses to determine the number of ballots L, the number of registrations L', and the result z. Moreover, it derives the number of real voters $n_{\mathcal{V}}$ using z. It uses these data to compute the cover, and it adds the correct number of dummy ballots (for these, it sets

w and $\texttt{EncCounter}$ to random ciphertexts) to obtain $\mathcal{B}$. Then it computes a list of zero ciphertexts (encryptions of zero) of equal length, and simulates the shuffle proof Π_S. It then generates fake voter identifiers VId and $\texttt{count}$ corresponding to the cover it computed earlier, associates these to shuffled ballot β_i, and simulates the proofs Π^d. Next, for each resulting group, it generates a random encryption of zero $\bar{V}_j = \texttt{VoteZEnc}(pk_v, r_j)$ and simulates the corresponding proof Π_R. Then, it returns the randomness r_j and the indices of the dummy voters corresponding to the cover it computed early. Finally, for each remaining vote, it generates a random v_j and simulates the shuffle proof Π'_S.

In this proof, we will step by step replace all the ciphertexts that depend on the bit b by random ciphertexts. In particular, we first show that the adversary learns nothing about b during the election phase. We then show that it also learns nothing about b during the tally phase. The result follows.

Game G_1: Game G_1 is as the $\texttt{Exp}^{\texttt{cr},b}_{(,\epsilon}, \mathcal{E}, \mathcal{C})$ experiment. (Note that contrary to the proof of ballot privacy we do not fix the value for b.)

Game G_2: Game G_2 is as game G_1, but we compute the result directly based on the ballots on PBB_0. Let $[\beta_1, \ldots, \beta_L]$ be the list of ballots where

$$\beta_i = (V_i, \Pi_V, pk_i, w_i, \texttt{EncCounter}_i, s_i, \sigma_i).$$

Let $\mathcal{C}_i = \texttt{Dec}(sk, V_i)$, $VId_i = \texttt{Dec}(sk_{TS}, w_i)$, and $\texttt{count}_i = \texttt{Dec}(sk_{TS}, \texttt{EncCounter}_i)$. Then compute the result:

$$z = \rho(((VId_1, \texttt{count}_1), \mathcal{C}_1), \ldots, ((VId_L, \texttt{count}_L), \mathcal{C}_L))$$

As per strong consistency, games G_2 and G_1 are indistinguishable.

Game G_3: Game G_3 is as game G_2, but with all the zero-knowledge proofs replaced by simulations. This includes the shuffle proof Π_S, the decryption proofs of Π^d of the shuffled w'_i and $\texttt{EncCounter}'_i$s, the reencryption proofs Π_R, and the shuffle proof Π'_S produced in Filter; as well as the tally proof Π_0 which we replace by the output of $\texttt{SimProof}(PBB_0, z)$. We use the random oracle to simulate this step, which is indistinguishable by the simulatability of the zero-knowledge proof system.

Game G_4: Game G_4 is as game G_3 but we do not decrypt the w_i and $\texttt{EncCounter}_i$ anymore when running Filter. Instead, we proceed as follows. All ballots $\beta_i = (V_i, \Pi_V, pk_i, w_i, \texttt{EncCounter}_i, s_i, \sigma_i)$ on the bulletin boards are valid. Hence, σ_i is a valid signature by CA_0 resp. CA_1 on w_i and $\texttt{EncCounter}_i$. Since the signature scheme is unforgeable, we know

these ciphertexts were created by CA_0 resp. CA_1. Hence, we can associate to them the corresponding plaintexts VId_i and $\texttt{count}_i$. Moreover, we know the permutation used by the TS during Filter, so we can also provide the correct plaintexts in step 4 of Filter on PBB_0 (recall the proofs of decryption Π^d are already simulated).

Game G_5: Game G_5 is as game G_4, but we replace the ciphertexts w_i and $\texttt{EncCounter}_i$ in the token τ_i by random ciphertexts for all tokens. Similarly, we replace the w_i and $\texttt{EncCounter}_i$ ciphertexts for the dummy ballots by random ciphertexts. Note that per the change in game G_4 we still associate the correct plaintexts VId_i and $\texttt{count}_i$ in the Filter protocol. A hybrid argument with reductions to the CPA security of the ElGamal encryption scheme shows that games G_5 and G_4 are indistinguishable. This reduction is possible since we no longer need to decrypt these ciphertexts.

Game G_6: Game G_6 is as game G_5, but we replace the encrypted votes V_i in the $O\texttt{cast}()$ call by encryptions of the zero vector, i.e., $V_i = \texttt{VoteZEnc}(pk_v, r)$ for a uniformly random randomizer r. Given that we assume TS is honest, and hence the adversary cannot decrypt the ciphertexts, a hybrid argument with a reduction to the NM-CPA security of the ElGamal encryption scheme with zero-knowledge proof shows that games G_6 and G_5 are indistinguishable. Note that in this reduction we use the O_D of the NM-CPA challenger to decrypt votes in the adversary-determined ballots before computing the result z.

Note that as of game G_6, the adversary's view of the bulletin board before calling $O\texttt{tally}()$ is independent of the value of b (the ballots resulting from the $O\texttt{cast}$ call also contain a random ephemeral public key pk and the signatures s and σ, but these are also independent of the actual voter selected).

We now proceed to show that the adversary also cannot learn anything from the output of Filter. Notice that, regardless of the value of b, the filter step is computed with the same number of voters n_t, the same number of ballots L and the same number of obtained tokens L'. Therefore, the output of Filter applied to PBB_0 and that of $\texttt{SimFilter}$ applied to PBB_1 should be indistinguishable. In the following game steps we replace the ciphertexts after shuffling by zero-ciphertexts and show that these steps are indistinguishable for the adversary.

Game G_7: Game G_7 is the same as game G_6, but we replace the ciphertexts $w'_i, \texttt{EncCounter}'_i$ and τ'_i after shuffling by random encryptions of zero.

We proceed as if they still decrypt to the correct values. Note that we already simulate the shuffle proof and decryption proofs. Again, a hybrid argument with reductions to the CPA security of the ElGamal encryption scheme shows that the games G_7 and G_6 are indistinguishable. This reduction is possible since we no longer need to decrypt these ciphertexts.

Game G_8: Game G_8 is the same as game G_7, but we replace the shuffled encrypted votes V_i' by random encryptions of zero. Similarly, we replace the randomizations, r_o, of the votes corresponding to dummy voters by the corresponding new randomization. This causes the pre-selected votes $\bar{V}_j$ per group to be incorrect, but this does not matter as we simulate the second shuffle proof, Π_S', anyway. As before, the indistinguishability of this step follows from the NM-CPA security of the vote encryption scheme.

Game G_9: Game G_9 is the same as game G_8, but we replace the second shuffled votes v_j by random encryption of zero. This causes the selected votes v_j after the shuffle to be incorrect with respect to the result, but this does not matter as we simulate the proof of the tally. As before, the indistinguishability of this step follows from the NM-CPA security of the vote encryption scheme.

Game G_{10}: Game G_{10} is as game G_8, but we replace the filter and tally proofs on PBB_0 by simulations: we set $(\mathcal{S}_0, \theta_0) \leftarrow \texttt{SimFilter}(PBB_0, L', z)$ and $\Pi_0 \leftarrow \texttt{SimProof}(PBB_0, z)$. Note that this difference is purely syntactic, as per the changes we made before, we already computed exactly the output of `SimFilter` on PBB_0 and the result z.

Clearly the resulting view is independent of b. And coercion resistance follows. □

Proof of Verifiability

Theorem 12. VoteAgain is verifiable under the DDH assumption in the random oracle model.

Proof. At the end of the filter procedure, the TS (or in our case, the adversary) outputs a list of selected votes $\mathcal{S} = \{v_1, \ldots, v_{n_\mathcal{V}}\}$, a proof θ, the result z and the tally proof Π. Because Π verifies, we know that the result z is the addition of the votes contained in $\mathcal{S}$.

We first show that each encrypted vote v_j contains a vote for a single candidate or an empty vote. Given that Π_S and Π_S' validate, we know that

the vote v_j corresponds to a group of ballot indices $G_j = \{i_1, \ldots, i_{\nu_j}\}$, corresponding to voter identifier VId_j. Moreover, the index $j*$ is such that the decrypted index $\texttt{count}_{j*}$ is maximal. Because Π_R is valid, we know that either

1. v_j is the reencryption of vote V_{j*} and $\texttt{Dec}(\tau_{j*}) = g^0$,
2. v_j is the encryption of zero and $\texttt{Dec}(\prod_{k=1}^{\nu_j} \tau_{i_k}) = g^{\nu_j}$.

We now show that in the first case V_{j*} must be the encryption of a single candidate. Because $\tau_{j*} = g^0$ and the correctness of the shuffle proofs Π_S, Π'_S, we know that V_{j*} originates from a valid ballot cast by a voter. This ballot included a proof that V_{j*} is the encryption of a single candidate.

Let n_t be the number of voters that requested a voting token and for which at least one ballot is included on the bulletin board. We argue that the number of non-zero ballots that is included in the tally equals n_t. Let $VId_{i_1}, \ldots, VId_{i_{n_t}}$ be the corresponding voter identifiers. Because of the correctness of the shuffle, Π_S, there exists corresponding groups $G_{i_1}, \ldots, G_{i_{n_t}}$ to these voter identifiers after shuffling.

We show that any other group G_ξ contributes an empty vote to the tally. Let VId_ξ be the corresponding voter identity. The adversary cannot forge signatures by the PA, so any ballot with voter identity VId_ξ was added as a dummy ballot with τ_D as a tag. Therefore, in group G_ξ, each tag encrypts g^1, so the only possible disjunct in Π_R is therefore the second, and thus $\bar{V}_\xi$ is the encryption of zero.

We show that only such encrypted votes may be removed after the second shuffle. The adversary needs to find a random r such that $v_i = \texttt{VoteZEnc}(pk_v, r)$. Given that the DL-assumption holds, the adversary can only find such r if the underlying plaintext is zero with very high probability. Given that the encryption of candidate zero is not a permitted option for real voters, and given the correctness of Π_V, only votes corresponding to the above groups may be removed by the adversary after the second shuffle.

So, we now know that only the groups $G_{i_1}, \ldots, G_{i_{n_t}}$ each contribute exactly one candidate to the tally, and no more candidates are added by the other groups. We assign each group to one of the three groups in the game: the voters in Checked, the voters in Unchecked, or the voters in Corrupted. The result then follows.

We now show that the correct values are tallied for each of the voters in Checked that verified that their ballots were correctly cast. Consider a voter $i \in$ Checked with voter identifier VId_i. Let `ctr` be the last ballot that it verified. We need to show that the tally includes either i's ballot `ctr`, or a ballot with a higher counter. We know ballot `ctr` was added to the bulletin board.

Therefore, the corresponding group G_i (matching voter identifier VId_i) containing ν_i ballots, must contain a shuffled ballot $(V'_j, VId_i, \texttt{count}_j, \tau'_j)$ corresponding to the original ballot `ctr` (because the shuffle proof and decryption proofs are valid). Note that τ'_j must be a decryption of g^0 by construction, therefore the tags in group G_j (which must be encryptions of g^0 or g^1) can never sum to g^{ν_i} and therefore, we must take the first disjunct in the reencryption proof Π_R: $\bar{V}_i$ must be the reencryption of an encrypted ballot $j*$ where τ_{j*} decrypts to g^0. Therefore, Π_R must contain the encrypted vote corresponding to a real ballot cast by voter i. Finally, since $j*$ maximizes $\texttt{count}_{j*}$ in the group, we know in particular, that $\texttt{count}_{j*} \geq \texttt{count}_j$ corresponding to the verified ballot. Therefore, we conclude that indeed $\bar{V}_i$ reencrypts either ballot `ctr` by voter i, or a ballot with a higher counter by voter i. Given the correctness of the second shuffled proved with Π'_S, there will be a selected ballot v_i encrypting the same value as $\bar{V}_i$. Finally, given the correctness of the mixnet and decryption proofs in Tally, either `ctr` by voter i, or a ballot with a higher counter by voter i, will be counted in the final tally.

Now suppose a group G_i corresponds to a voter i in Unchecked. Then, by the same argument as for voters in Checked, we know that the tally must either drop all ballots or include one of the ballots cast by voter i.

Finally, any remaining groups correspond to voters in Corrupted. Notice that any voter that is not in Checked or Unchecked must be in Corrupted. Since, each remaining group corresponds to an actual voter, and this voter is not in either of the former groups, it must indeed correspond to a voter in Corrupted. □

4.6.6 Performance Evaluation

We evaluate the performance of VoteAgain using a Python prototype implementation of its core cryptographic operations.[13] We did not implement the `GetToken`protocol, but we note that as it relies on standard cryptography it can be implemented easily and cheaply; nor did we implement the bulletin board as it is not core to our design. We use the petlib [Dan] binding to OpenSSL for the group operations using the fast NIST P-256 curve. We ran all experiments in Linux on a single core of an Intel i3-8100 processor running at 3.60GHz. We expect nation-wide elections to have much more processing power available. For example, the Swiss CHVote system, which aims to support 8 million voters, has around 32 cores available per party in the sys-

[13]The code is open source and can be found here: `https://github.com/spring-epfl/voteagain`.

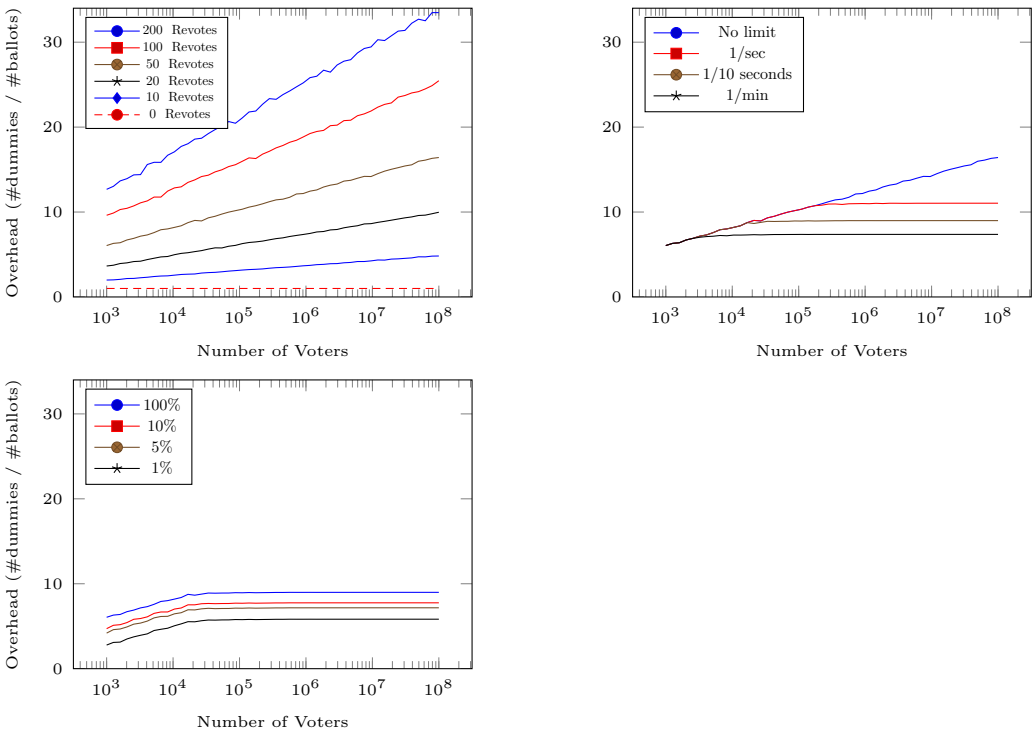


Figure 4.15: Dummy ballots overhead: Varying percentages of extra ballots with respect to the total number of voters (top left); effect of a rate-limit on revoting voters assuming at most 50% extra ballots (top right); and limiting the percentage of voters that revotes assuming at most 50% extra ballots and a rate-limit of 1 ballot per 10 seconds (bottom left).

tem. We also include performance *estimates* of running the system on a large machine with 8 Intel Xeon Platinum 8280L processors with 28 cores each, running at 2.7Ghz. As our scheme is almost completely parallelizable (only the hash functions for the non-interactive zero-knowledge proofs need to be computed sequentially), we estimate a 90% parallelization gain: a speedup of 170 times when using the 8x28 cores with respect to the single core.

For all experiments we empirically select the best cover size S by sweeping over values from 1 to 64. In the majority of cases the optimal S is in the range $[2, 4]$.

Creating a ballot. We use an ElGamal ciphertext to encrypt the voter's choice, and a Bayer and Groth [BG13] zero-knowledge proof of membership to show that the selected candidate is eligible. Creating a ballot from 1000 eligible candidates costs 1.2 seconds, while verifying its correctness costs 0.17 seconds. The size of this proof is 1.5 kB.

Impact of revoting. Figure 4.15 shows the overhead depending on the

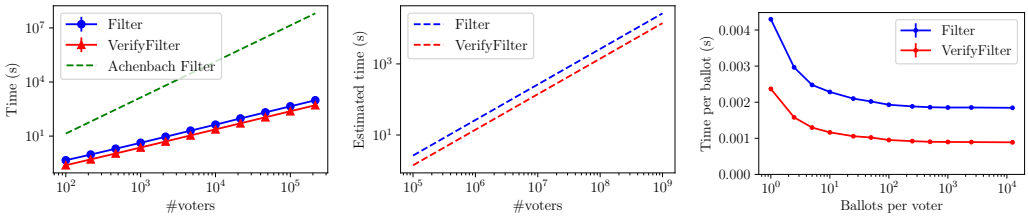


Figure 4.16: Cost of Filter and VerifyFilter: Measured cost on single core (left); estimated cost on 8 processor machine (8×28 cores, center); and effect of different distributions of 50 000 ballots (including dummies) among voters (right). Note that one ballot per voter causes the highest processing time per vote.

number of votes, in terms of number of dummies per real ballot. This overhead influences the computation time of shuffling and filtering in the tally phase. In the leftmost figure we model users' revoting behaviour as a percentage of the number of voters: 50% models that half of the voters revoted once, and 200% models that all voters revote twice. We note that the overhead of 100% voters revoting once is equivalent to, for example, 25% of the voters revoting 4 times. As expected, the overhead increases with both the number of voters and the number of revoted ballots. However, even for 100 million voters revoting twice (200% revotes), the overhead is at most a factor of 32 (Figure 4.15 top left).

Casting a vote takes time. Thus, revoting patterns are constrained by the number of ballots that can be cast during an election. We consider an election period of 24h (larger than most countries), and bound how often a *single* voter can vote (1 ballot per second, per ten seconds, and per minute). As this limits the number of voters with a large amount of ballots, we do not need large covers, reducing the overhead (see Figure 4.15, top right). Similarly, assuming that all voters will revote is very conservative. In a normal election one expects the vast majority of voters to vote once. In Figure 4.15, bottom left, we show the overhead when the number of voters that cast more than one vote is limited. As fewer voters revote, the total amount of votes is smaller and so are the covers.

Filtering. We implemented a non-optimized version of Bayer-Groth's verifiable shuffle protocol [BG12] to implement steps 3 and 6 of Procedure 4.6.5. We measure the execution time of filtering and verifying, when varying the number of voters. Figure 4.16 left shows the times to run Filter and VerifyFilter on a single core machine. Figure 4.16 middle shows the estimated processing times on the big 8 processor Xeon machine. We estimate that the 8 processor machine can filter and tally the second round presidential election in Brazil (147 million registered voters) in 65 minutes if no voter revotes,

and within 11 hours assuming 50% extra ballots and at most one ballot per voter per ten seconds. We note that elections usually tally ballots per state, city, or smaller electoral district. In general we expect the number of ballots to be much smaller. All ballot groups in Figure 4.16 left and center have size one. Figure 4.16 right shows the effect of larger ballot groups resulting from revoting and dummy voters. As the average group size increases, the computation time goes down. Therefore, Figure 4.16 gives an upper bound on the processing time, given a known cover size.

For comparison we computed a lower bound on the filter cost of Achenbach et al.'s filter method by counting the number of group operations needed per ballot. We used this number to compute the estimate in Figure 4.16 left. A small-town election with 100000 ballots takes 5 core months to filter in their scheme. Even on the large Xeon machine, an election with 1 million ballots takes over four months to complete. Our method needs respectively 7 core minutes and 30 seconds. The sizes of the tally proofs in VoteAgain for these examples are 54 and 501 MB respectively.

Smaller regions. Many countries report election results per region, such as a province, a city, or a neighborhood. In those cases, results can be computed per region at lower computation cost. However, even in this setting, Achenbach et al.'s quadratic approach scales poorly. We note that the allowable size of reporting regions depend on local regulations, with the smallest regions likely being cities or neighborhoods, which can easily total 100000s of voters. As Figure 4.16 (left) shows, even in this configuration, Achenbach et al.'s quadratic approach requires 3 to 4 orders of magnitude more computation resources than VoteAgain.

Tallying. We also measured the execution time of a single step of the mix network – a single shuffle and one verifiable decryption – using our verifiable shuffle implementation. Our results show that one step is a factor of three times faster than our filter protocol, e.g., mix-and-decrypting the 100000 ballots takes less than 2 core minutes and 1 million ballots take less than 7 seconds on the Xeon machine.

4.7 Wrapping up

In this chapter we have presented two internet voting schemes. Both use the re-voting method to prevent coercion attacks. To provide deniability at the time of re-voting, we used dummy votes. Including a random number of dummy votes is the trivial solution, and as we have presented, barely affects the complexity of filtering. However, by having this random variable, it is much harder to formalise security properties, and we were forced to trade-off

	Filtering				Coercion		
	Deniable	Verifiable	Complexity	Crypto State	Assumption	Strict	IC
JCJ	No	Yes	n^2	Yes	K_T+ AC	No	No
Rønne *et al.*	No	Yes	n	Yes	K_T+ AC	No	Yes
Blackbox	TTP	No	n	Yes	TTP	No	No
Achenbach *et al.*	K_T	Yes	n^2	Yes	K_T+ AC	No	No
Locher *et al.*	K_T	Yes	n^2	Yes	K_T+ AC	No	No
NetVote	TTP	Yes	n	No	TTP	Yes	No
VoteAgain	TTP	Yes	$n \log n$	No	TTP	No	No

Table 4.2: Comparison of existing schemes. NetVote constitutes the first re-voting scheme with linear complexity, which is verifiable and does not depend on immature cryptography (IC). We denote a t-out-of-k assumption by K_T. We also compare the schemes on whether they require a cryptographic state (user needs to securely store cryptographic material) or not. We introduce the new notion of *strict coercion resistance* to achieve linear filtering.

on these. Using a deterministic dummy vote aggregation technique is the best solution for something as delicate as internet voting —it is of interest to have the highest guarantees on the security properties, and while the trade off was moving from linear to quasi-linear filtering time, we showed that this overhead was not a limitation for nation wide elections. In table 4.2 we present a comparison of the two schemes presented in this chapter with the state of the art. Our principal goal was to present a scheme that was usable by non-technical users, and therefore not requiring a cryptographic state was a strong requirement. We have shown that not only have we achieved that goal, but we have performed without an impact in usability compared with the best current schemes for the case of NetVote, and a sub-linear decline for the case of VoteAgain. Moreover, only a scheme with no deniability [Røn+20], and one without verifiability [Gjø10] achieve the same assymptotic complexity of our scheme.

CHAPTER 5

Conclusions, Contributions and Future Work

The goal of this thesis was to explore the possibility of constructing usable, scalable privacy-preserving protocols that offered the same, or close to equal, functionality to their non-private counterparts. In this chapter we conclude the objectives set at the beginning of the thesis, followed by a concrete list of contributions. Then, we proceed with a list of how the research presented in this thesis can be improved in future work. Finaly, we finish the chapter and thesis with some closing remarks.

5.1 Conclusions

As shown throughout the thesis, we achieve at a high level of success the objectives, P1–P4 and V1–V3, set a the start of the project (see Section 1.2). In summary, zkSENSE presents a novel design allowing for a mechanism to detect bots that does not require to send any data from the user to the verifying party, but that it also requires no interaction from the user. The attestation, instead, happens by using natural actions that a user would make in a normal use of a mobile device. Similarly, our evaluation confirmed that our solution was usable and scalable. However, in this thesis we did not formally study how this solution affects people that suffer some sort of disability, or how can the solution be extended to cover this affected portion of users.

On the other hand, NetVote and VoteAgain offer scalable and usable internet voting protocols with different trade-offs between scalability and se-

curity. A voting scenario with low concerns on coercion would make use of NetVote, which offers a slightly better performance in the tallying procedure. On the other hand, a senario with higher concerns on coercion would use VoteAgain, which offers stricter guarantees in terms of coercion. Both schemes, however, require minimal effort from the user's perspective and offer competitive performance in comparison with the state-of-the-art voting schemes. In terms of the security properties, we had to lower our initial expectations of providing maximum guarantees. Both schemes need to rely on the correct behavior of certain parties to provide the desired properties, but we believe that this trade-off makes both solutions interesting to be used in real life elections. Similarly, a user study would make the statement of 'user-friendly' stronger. Both schemes make minimal assumption on what are the user actions (no cryptographic material, no anonymous credentials, no usage of two distinct devices, etc). However, a user study comparing our construction with other existing constructions would further strengthen that statement.

5.2 Contributions

In this section we list the contributions made in the thesis, mainly in Chapters 3 and 4.

5.2.1 zkSENSE

As presented in Section 2.3, current bot detection mechanisms offer solutions which either require interaction by the user, or that are invasive to their privacy. In this thesis we presented zkSENSE: a novel friction-less and privacy-preserving mechanism for humanness attestation that aims to replace CAPTCHA systems in mobile devices. In particular, Chapter 3 makes the following contributions:

1. We designed a human attestation system (zkSENSE) that is both friction-less and privacy-preserving(see Section 3). We formally proved that it provides privacy and verifiability(see Section 3.5). zkSENSE leverages mobile motion sensors to verify that user actions (e.g. type/touch events) on a mobile device are triggered by an actual human. Such classification takes place by studying the output of the mobile device's motion sensors during the particular user action. To set the ground truth, we instrumented an actual Android browser app to capture both user clicks and sensor traces from a small set of real users. Our approach is tested under various

scenarios: (i) when device is resting (on a table), (ii) when there is artificial movement from device's vibration, or (iii) from an external swinging cradle.

2. We modified and implemented a sub-linear inner product proof into its zero-knowledge counterpart, which is of independent interest. We use it to implement zkSVM (which we provide open-source[14]): a zero-knowledge based library for enclosing results of an SVM classifier into zero-knowledge proofs (see Section 3.3). zkSVM leverages arithmetic properties of commitment functions and prover-effective proofs to ensure the integrity of the classification result reported to a remote server.

3. We implemented an Android SDK and a demo app to showcase the detection accuracy of zkSENSE (see demo video[15]). zkSENSE is *transparent to the user* and capable of verifying their humanness with accuracy higher than related proposals [Gue+18] (92%). Performance evaluation results of our prototype show that the entire attestation operation takes less than 3 seconds (when the solution of a visual CAPTCHA by a human takes 9.8 seconds [All13][16]) and consumes less than 5 mAh of power (see Section 3.6).

5.2.2 i-voting

Chapter 4 presents two internet voting schemes, which provide different trade offs with respect to scalable internet voting solutions. In Section 2.3.2 we presented how current solutions are either not scalable or construct unusable voting procedures. NetVote and VoteAgain provide two novel constructions with particular care in usability and scalability. In particular, Chapter 4 makes the following contributions:

1. We introduced NetVote and VoteAgain, two novel remote electronic revoting schemes based on well defined and widely used cryptographic constructions. See Sections 4.5 and 4.6 respectively.

2. Our constructions do not require any effort from the user except for authentication and vote selection. We do not store any state (cryptographic or non-cryptographic) in the user's device, meaning that they can vote with whatever device they wish.

[14] zkSVM source code: `https://github.com/zkSENSE/zkSVM`
[15] zkSENSE demo: `https://youtu.be/FARbekFO1d0`
[16] Note that while this result is from 2013, it was performed over humans

3. We introduced two novel efficient padding schemes that hide revoting at a low cost. Our first padding strategy is probabilistic, achieving a filtering phase complexity of $O(n)$, while the second result, a deterministic padding strategy, offers a filtering complexity $O(nlogn)$ where n is the number of ballots. See Sections 4.5.2 and 4.6.4. Our experiments show that in many practical scenarios the cost can be even lower.

4. In Section 4.4 we showed that previous definitions of coercion resistance in the revoting setting are vacuous. We provided a new coercion resistance definition and we adapt modern definitions of ballot privacy [Ber+15] and verifiability [Cor+14; Cor+16] to the revoting setting. We proved that the schemes presented satisfy these definitions.

5. We introduced two new game definitions for the security properties of internet voting solutions: practical everlasting privacy and strict coercion resistance (see Sections 4.4.2 and 4.4.5 respectively). The latter is a more relaxed property (with respect to coercion resistance) that allows us to offer linear filtering in NetVote. We proved that NetVote satisfies both (in addition to verifiabiliy and ballot privacy).

6. In Section 4.5.2 we performed a simulation of NetVote, to understand how the probabilistic dummy strategy affects the filtering procedure. Then, in Section 4.6.6 we evaluated the scalability of VoteAgain, our most complex scheme, on a prototype implementation of the core cryptographic primitives. Our results show that NetVote scales linearly with respect to the number of cast votes, and VoteAgain can support elections with millions of users.

5.3 Future Work

The results of this thesis set a promising ground for advances in both fields (bot detection and i-voting). However, there are certain areas that remain open and that are left for future work. In particular, bot detection could be improved in the following aspects:

Closer guarantees to current bot-detection. In Chapter 3 we presented a fast, privacy-preserving, and non-interactive human detection mechanism. However, we did not not cover a mechanism that ensures that the data indeed comes from a non-rooted device, or specifically, from the sensors of the user's mobile phone. The lines of research that would resolve this question consist in providing the zkSENSE app together with a mechanism to detect rooted devices or work over devices

that have their sensors within a trusted platform. Similarly, improving the ML models (without hindering running times) would improve the guarantees that bots are correctly detected.

zkSENSE for voice commands. Our main assumption for zkSENSE is that users touch their screen. However, it would be of interest to further extend this to visually impaired people that use the devices mainly with voice commands. To this end, extending the classifier to work with the buttons of the devices (i.e. lock/unlock, volume up/down) would make zkSENSE a more complete construction.

Extend zkSENSE for keyboard/mouse movements. Authentication using keyboard or mouse patters already exists [Feh+12; Mes+11]. A natural extension of zkSENSE would be to study whether we can perform private keystroke biometrics by leveraging the idea explored in this thesis, i.e. evaluating the model in zero-knowledge.

On the other hand, for internet voting, the aspects that can be improved with further research are the following:

User study for voting schemes. The solutions we provided seem ideal in what concerns usability and scalability. While evaluation for the scalability of the solution has been provided, no user study has been performed for them. Understanding user opinion for an internet voting protocol (or human attestation mechanism) seems the next natural step before adoption.

Less trust in voting authorities. Our focus was to design a usable and scalable internet voting scheme with the required security guarantees. To do so, we made certain trade-offs, assuming parties in the protocol to be honest. An interesting line of research would be to understand whether these assumptions can be reduced without hindering user experience, or presenting complex solutions.

Complexity of NetVote with security of VoteAgain. In NetVote we weakened the definition of coercion resistance to strict coercion resistance to achieve linear filtering. This is due to the fact that the probability of voters selling votes is not negligible. Nonetheless, it is still small. It would be of interest to make a more thorough analysis of the probability of success of selling votes. More generally, it would be ideal to close this line of research with a proposal that satisfies the strong definition of coercion resistance while maintaining the linear filtering complexity.

5.4 Closing remarks

In this thesis we have shown that privacy can be achieved without hindering usability or scalability. It is well accepted that privacy comes at a cost, and certain trade-offs need to be assimilated: this thesis is no exception to it. With the results presented in this thesis we hope to shift the debate on privacy-preserving solutions. First of all, with our work, we show how the claimed dilemma of "security or privacy" is not true —user's personal data does not need to be disclosed to achieve secure solutions. Secondly, the cost does not need to be suffered by the user or the scalability. By presenting scalable user friendly protocols we aim to increase the privacy-preserving constructions that do put users and their experience as a determining factor for the success of the solution. With this thesis, we hope to shift, by a bit, the balance towards the users' benefits.

Bibliography

[Ach+15] Dirk Achenbach, Carmen Kempka, Bernhard Löwe, and Jörn Müller-Quade. "Improved Coercion-Resistant Electronic Elections through Deniable Re-Voting". In: *USENIX Journal of Election Technology and Systems (JETS)* 2 (2015), pp. 26–45. URL: https://www.usenix.org/conference/jets15/workshop-program/presentation/achenbach (cit. on pp. 33, 85, 88, 89).

[Adi08] Ben Adida. "Helios: Web-based Open-audit Voting". In: *Proceedings of the 17th Conference on Security Symposium*. SS'08. USENIX Association, 2008, pp. 335–348 (cit. on p. 104).

[Adj+13] Idris Adjerid, Alessandro Acquisti, Laura Brandimarte, and George Loewenstein. "Sleights of Privacy: Framing, Disclosures, and the Limits of Transparency". In: *Proceedings of the Ninth Symposium on Usable Privacy and Security*. SOUPS '13. Association for Computing Machinery, 2013, 9:1–9:11. DOI: 10.1145/2501604.2501613 (cit. on p. 2).

[Adr+18] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice". In: *Communications of the ACM* 62.1 (2018), 106–114. DOI: 10.1145/3292035 (cit. on p. 3).

[AFA19] Ismail Akrout, Amal Feriani, and Mohamed Akrout. *Hacking Google reCAPTCHA v3 using Reinforcement Learning*. arXiv. 2019. URL: http://arxiv.org/abs/1903.01003 (cit. on p. 7).

[All13] Tim Allen. *Having a CAPTCHA is Killing Your Conversion Rate*. 2013. URL: https://moz.com/blog/having-a-captcha-is-killing-your-conversion-rate (Last accessed 03/26/2022) (cit. on pp. 29, 64, 139).

[Ano19] Anonymous. *Demo: Privacy-Preserving Bot Detection for Mobile Devices*. 2019. URL: https://youtu.be/FARbekFO1d0 (cit. on p. 57).

[ANS20] ANSSI. *Recommandations pour les Architectures des Systèmes d'Information Sensibles ou Diffusion Restreinte*. https://www.ssi.gouv.fr/guide/recommandations-pour-les-architectures-des-systemes-dinformation-sensibles-ou-diffusion-restreinte/. Agence Nationale de la Sécurité des Systèmes d'Information. 2020 (cit. on p. 9).

[Ara+13] Myrto Arapinis, Véronique Cortier, Steve Kremer, and Mark Ryan. "Practical Everlasting Privacy". In: *Principles of Security and Trust - Second International Conference, POST*. Ed. by David A. Basin and John C. Mitchell. Vol. 7796. LNCS. Springer, 2013, pp. 21–40. DOI: 10.1007/978-3-642-36830-1_2 (cit. on p. 83).

[Ara+16] Roberto Araújo, Amira Barki, Solenn Brunet, and Jacques Traoré. "Remote Electronic Voting Can Be Efficient, Verifiable and Coercion-Resistant". In: *Financial Cryptography and Data Security - 2016 International Workshops, BITCOIN, VOTING, and WAHC*. Vol. 9604. LNCS. Springer, 2016, pp. 224–232. DOI: 10.1007/978-3-662-53357-4_15 (cit. on pp. 31, 33).

[Bad+21] Christian Badertscher, Peter Gaži, Iñigo Querejeta-Azurmendi, and Alexander Russell. *On UC-Secure Range Extension and Batch Verification for ECVRF*. Technical report. https://iohk.io/en/research/library/papers/on-uc-secure-range-extension-and-batch-verification-for-ecvrf/. 2021 (cit. on p. vii).

[Bal15] James Ball. *Cameron wants to ban encryption – he can say goodbye to digital Britain*. 2015. URL: https://www.theguardian.com/commentisfree/2015/jan/13/cameron-ban-encryption-digital-britain-online-shopping-banking-messaging-terror (Last accessed 03/26/2022) (cit. on p. 3).

[Bar16] Elaine B Barker. *SP 800-57. NIST Recommendation for Key Management, Part 1: General*. Tech. rep. https://doi.org/10.6028/NIST.SP.800-57pt1r5. Gaithersburg, MD, United States, 2016 (cit. on pp. 9, 17).

[Bat19] BatteryLab. *A Distributed Platform for Battery Measurements*. 2019. URL: https://batterylab.dev (cit. on p. 58).

[Ber+13] Daniel J Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko van Someren. "Factoring RSA Keys from Certified Smart Cards: Coppersmith in the Wild". In: *19th International Conference on the Theory and Application of Cryptology and Information Security —ASIACRYPT*. Ed. by Kazue Sako and Palash Sarkar. Vol. 8270. LNCS. Springer, Berlin, Heidelberg, 2013, pp. 341–360. DOI: 10.1007/978-3-642-42045-0_18 (cit. on p. 9).

[Ber+15] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. "SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions". In: *IEEE Symposium on Security and Privacy, SP*. IEEE Computer Society, 2015, pp. 499–516. DOI: 10.1109/SP.2015.37 (cit. on pp. 79, 81–83, 104, 123, 125, 140).

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. "Non-Interactive Zero-Knowledge and Its Applications". In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC '88. Association for Computing Machinery, 1988, 103–112. DOI: 10.1145/62212.62222 (cit. on pp. 24, 27).

[BG12] Stephanie Bayer and Jens Groth. "Efficient Zero-Knowledge Argument for Correctness of a Shuffle". In: *Advances in Cryptology – EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer Berlin Heidelberg, 2012, pp. 263–280. DOI: 10.1007/978-3-642-29011-4_17 (cit. on pp. 75, 133).

[BG13] Stephanie Bayer and Jens Groth. "Zero-Knowledge Argument for Polynomial Evaluation with Application to Blacklists". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques - EUROCRYPT*. Vol. 7881. LNCS. Springer, Berlin, Heidelberg, 2013. DOI: 10.1007/978-3-642-38348-9_38 (cit. on p. 132).

[BGC17] Attaullah Buriro, Sandeep Gupta, and Bruno Crispo. "Evaluation of Motion-Based Touch-Typing Biometrics for Online Banking". In: *International Conference of the Biometrics Special Interest Group, (BIOSIG)*. Ed. by Arslan Brömme, Christoph Busch, Antitza Dantcheva, Christian Rathgeb, and Andreas Uhl. Vol. P-270. LNI. GI / IEEE, 2017, pp. 219–226. DOI: 10.23919/BIOSIG.2017.8053504 (cit. on p. 28).

[BGR12] Sergiu Bursuc, Gurchetan S Grewal, and Mark D Ryan. "Trivitas: Voters Directly Verifying Votes". In: *E-Voting and Identity: Third International Conference, VoteID*. Ed. by Aggelos Kiayias and Helger Lipmaa. Vol. 7187. LNCS. Springer Berlin Heidelberg, 2012, pp. 190–207. DOI: 10.1007/978-3-642-32747-6_12 (cit. on p. 31).

[Bit+12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS '12. Association for Computing Machinery, 2012, 326–349. DOI: 10.1145/2090236.2090263 (cit. on p. 25).

[BL16] Katie Benner and Eric Lichtblau. *US says it has unlocked iPhone without Apple*. 2016. URL: https://www.nytimes.com/2016/03/29/technology/apple-iphone-fbi-justice-department-case.html (Last accessed 03/26/2022) (cit. on p. 2).

[BLN14] Joppe W. Bos, Kristin E. Lauter, and Michael Naehrig. "Private predictive analysis on encrypted medical data". In: *J. Biomed. Informatics* 50 (2014), pp. 234–243. DOI: 10.1016/j.jbi.2014.04.003 (cit. on p. 30).

[BLN16] Daniel J. Bernstein, Tanja Lange, and Ruben Niederhagen. "Dual EC: A Standardized Back Door". In: *The New Codebreakers - Essays Dedicated to David Kahn on the Occa-*

sion of His 85th Birthday. Ed. by Peter Y. A. Ryan, David Naccache, and Jean-Jacques Quisquater. Vol. 9100. LNCS. Springer, 2016, pp. 256–281. DOI: 10.1007/978-3-662-49301-4_17 (cit. on p. 3).

[Boc+17] Kevin Bock, Daven Patel, George Hughey, and Dave Levin. "unCaptcha: A Low-Resource Defeat of reCaptcha's Audio Challenge". In: *11th USENIX Workshop on Offensive Technologies*. Ed. by William Enck and Collin Mulliner. USENIX Association, 2017 (cit. on p. 7).

[Boo+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016*. Ed. by Marc Fischlin and Jean S. Coron. Vol. 9666. LNCS. Springer-Verlag New York, Inc., 2016, pp. 327–357. DOI: 10.1007/978-3-662-49896-5_12 (cit. on pp. 31, 64).

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. "How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios". In: *18th International Conference on the Theory and Application of Cryptology and Information Security - ASIACRYPT*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. LNCS. Springer, 2012, pp. 626–643. DOI: 10.1007/978-3-642-34961-4_38 (cit. on pp. 18, 105).

[BR11] Mikhail Bilenko and Matthew Richardson. "Predictive client-side profiles for personalized advertising". In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Ed. by Chid Apté, Joydeep Ghosh, and Padhraic Smyth. ACM, 2011, pp. 413–421. DOI: 10.1145/2020408.2020475 (cit. on p. 30).

[Bra00] Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000 (cit. on p. 76).

[Bra20] Brave Software Inc. *Brave Android Browser*. 2020. URL: https://brave.com/ (Last accessed 03/26/2022) (cit. on p. 38).

[BSI21] BSI. *Cryptographic Mechanisms: Recommendations and Key Lengths*. https://www.bsi.bund.de/EN/Service-Navi/Publications/TechnicalGuidelines/tr02102/tr02102_node.html. Bundesamt für Sicherheit in der Informationstechnik, BSI TR-02102-1 version 2022-01, 2021 (cit. on p. 9).

[Buc17] Mark Buchanan. *Why fake news spreads like wildfire on Facebook*. Sept. 2017. URL: https://www.chicagotribune.com/opinion/commentary/ct-perspec-fake-news-google-facebook-0904-story.html (Last accessed 03/26/2022) (cit. on p. 1).

[Bün+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020 (cit. on pp. 25, 26, 31, 42, 50, 56, 64, 65, 68, 75, 97).

[Bur+10] Elie Bursztein, Steven Bethard, Celine Fabry, John C. Mitchell, and Daniel Jurafsky. "How Good Are Humans at Solving CAPTCHAs? A Large Scale Evaluation". In: *31st IEEE Symposium on Security and Privacy, S&P*. IEEE Computer Society, 2010, pp. 399–413. DOI: 10.1109/SP.2010.31 (cit. on p. 29).

[Cad18] Carole Cadwalladr. *AggregateIQ: the obscure Canadian tech firm and the Brexit data riddle*. Mar. 2018. URL: https://www.theguardian.com/uk-news/2018/mar/31/aggregateiq-canadian-tech-brexit-data-riddle-cambridge-analytica (Last accessed 03/26/2022) (cit. on pp. 1, 3).

[CH11] Jeremy Clark and Urs Hengartner. "Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance". In: *Financial Cryptography and Data Security - 15th International Conference, FC*. Ed. by George Danezis. Vol. 7035. LNCS. Springer, Berlin, Heidelberg, 2011, pp. 47–61. DOI: 10.1007/978-3-642-27576-0_4 (cit. on p. 31).

[Cha18] Jefferson Chase. *German election could be won by early voting*. 2018. URL: http://www.dw.com/en/german-election-could-be-won-by-early-voting/a-40296550 (Last accessed 03/26/2022) (cit. on p. 8).

[Cha+21] David Chaum, Richard T Carback, Jeremy Clark, Chao Liu, Mahdi Nejadgholi, Bart Preneel, Alan T Sherman, Mario Yaksetig, Filip Zagórski, and Bingsheng Zhang. *VoteXX: Coercion Resistance for the Real World (preliminary extended abstract)*. `https://votexx.org/votexx-extended-abstract.pdf`. 2021 (cit. on p. 32).

[Cha81] David L. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Communications of the ACM* 24.2 (1981), 84–90. DOI: `10.1145/358549.358563` (cit. on p. 27).

[Cha83] David Chaum. "Blind Signatures for Untraceable Payments". In: *Advances in Cryptology Proceedings of Crypto 82*. Ed. by D. Chaum, R.L. Rivest, and A.T. Sherman. 1983, pp. 199–203. DOI: `10.1007/978-1-4757-0602-4_18` (cit. on p. 27).

[Che+21] Jun Chen, Xiangyang Luo, Liyan Zhu, Qikun Zhang, and Yong Gan. "Handwritten CAPTCHA recognizer: a text CAPTCHA breaking method based on style transfer network". In: *Multimedia Tools and Applications* (2021). Ed. by Borko Furht. DOI: `10.1007/s11042-021-11485-9` (cit. on p. 28).

[Chi+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *Advances in Cryptology - EUROCRYPT*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, 2020, pp. 738–768. DOI: `10.1007/978-3-030-45721-1_26` (cit. on p. 25).

[Chi92] Nancy Chinchor. "MUC-4 Evaluation Metrics". In: *Proceedings of the 4th Conference on Message Understanding*. MUC4 '92. Association for Computational Linguistics, 1992, 22–29. DOI: `10.3115/1072064.1072067` (cit. on p. 39).

[CL02] Jan Camenisch and Anna Lysyanskaya. "A Signature Scheme with Efficient Protocols". In: *Security in Communication Networks, Third International Conference, SCN*. Ed. by Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano. Vol. 2576. LNCS. Springer, 2002, pp. 268–289. DOI: `10.1007/3-540-36413-7_20` (cit. on p. 76).

[Cla19] Thomas Claburn. *Google's reCAPTCHA favors – you guessed it – Google: Duh, only a bot would refuse to sign into the Chocolate Factory*. 2019. URL: https://www.theregister.co.uk/2019/06/28/google_recaptcha_favoring_google/ (Last accessed 03/26/2022) (cit. on pp. 7, 29).

[CM99] Jan Camenisch and Markus Michels. "Proving in Zero-Knowledge that a Number is the Product of Two Safe Primes". In: *Advances in Cryptology — EUROCRYPT '99*. Ed. by Jacques Stern. LNCS. Springer Berlin Heidelberg, 1999, pp. 107–122. DOI: 10.1007/3-540-48910-X_8 (cit. on p. 26).

[Com18] European Commission. *2018 reform of EU data protection rules*. May 25, 2018. URL: https://ec.europa.eu/info/sites/default/files/data-protection-factsheet-role-edpb_en.pdf (Last accessed 03/26/2022) (cit. on pp. 2, 4).

[Con19] World Wide Web Consortium. *Captcha Alternatives and thoughts*. 2019. URL: https://www.w3.org/WAI/GL/wiki/Captcha_Alternatives_and_thoughts (Last accessed 03/26/2022) (cit. on p. 29).

[Cor+14] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. "Election Verifiability for Helios under Weaker Trust Assumptions". In: *European Symposium on Research in Computer Security - ESORICS*. Vol. 8713. LNCS. Springer Cham, 2014. DOI: 10.1007/978-3-319-11212-1_19 (cit. on pp. 85, 140).

[Cor+16] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. "SoK: Verifiability Notions for E-Voting Protocols". In: *IEEE Symposium on Security and Privacy (SP)*. 2016, pp. 779–798. DOI: 10.1109/SP.2016.52 (cit. on pp. 85, 140).

[CPJ18] Hanbyul Choi, Jonghwa Park, and Yoonhyuk Jung. "The role of privacy fatigue in online privacy behavior". In: *Computers in Human Behavior* 81 (2018), pp. 42 –51. DOI: 10.1016/j.chb.2017.12.001 (cit. on p. 2).

[Cry18] CryptoZ. *Chainanalysis- 3.8 million Bitcoin is lost forever.* `https://steemit.com/cryptocurrency/@crypto-z/chainanalysis-3-8-million-bitcoin-is-lost-forever`. 2018. (Last accessed 03/26/2022) (cit. on p. 73).

[CS97] Jan Camenisch and Markus Stadler. "Efficient Group Signature Schemes for Large Groups (Extended Abstract)". In: *Proceedings on Advances in Cryptology—CRYPTO '97*. Ed. by Burton S.Jr. Kaliski. LNCS. Springer-Verlag, 1997, 410–424. DOI: 10.1007/BFb0052252 (cit. on p. 24).

[CSJ09] A A Chandavale, A M Sapkal, and R M Jalnekar. "Algorithm to Break Visual CAPTCHA". In: *Second International Conference on Emerging Trends in Engineering Technology*. IEEE, 2009, pp. 258–262. DOI: 10.1109/ICETET.2009.24 (cit. on p. 28).

[CSP12] Helen Cripps, Craig Standing, and Vesna Prijatelj. "Smart Health Case Cards: Are they applicable in the Australian context?" In: *25th Bled eConference eDependability: Reliable and Trustworthy eStructures, eProcesses, eOperations and eServices for the Future* (2012) (cit. on p. 9).

[Dan] George Danezis. *Petlib: A python library that implements a number of Privacy Enhancing Technolgies*. Github. URL: `https://github.com/gdanezis/petlib` (Last accessed 03/26/2022) (cit. on p. 131).

[Dan+12] George Danezis, Markulf Kohlweiss, Benjamin Livshits, and Alfredo Rial. "Private Client-side Profiling with Random Forests and Hidden Markov Models". In: *Proceedings of the 12th International Conference on Privacy Enhancing Technologies*. PETS'12. Springer-Verlag, 2012, pp. 18–37. DOI: 10.1007/978-3-642-31680-7_2 (cit. on p. 30).

[Dav+17] Erhan Davarci, Betul Soysal, Imran Erguler, Sabri Orhun Aydin, Onur Dincer, and Emin Anarim. "Age group detection using smartphone motion sensors". In: *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE, 2017, pp. 2201–2205. DOI: 10.23919/EUSIPCO.2017.8081600 (cit. on p. 29).

[Dav+18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. "Privacy Pass: Bypassing Internet Challenges Anonymously". In: *Proceedings on Privacy*

Enhancing Technologies 2018.3 (2018), pp. 164 –180. DOI: 10.1515/popets-2018-0026 (cit. on pp. 27, 62).

[DBC16] Anupam Das, Nikita Borisov, and Matthew Caesar. "Tracking Mobile Web Users Through Motion Sensors: Attacks and Defenses." In: *The Network and Distributed System Security Symposium (NDSS)*. 2016. DOI: 10.14722/NDSS.2016.23390 (cit. on p. 29).

[DC07] Roberto Di Cosmo. *On privacy and anonymity in electronic and non electronic voting: the ballot-as-signature attack*. HAL Archives Ouvertes. https://hal.archives-ouvertes.fr/hal-00142440. 2007 (cit. on p. 91).

[Dev18] Google Developers. *reCAPTCHA v3*. 2018. URL: https://developers.google.com/recaptcha/docs/v3 (Last accessed 03/26/2022) (cit. on p. 28).

[Dev20] Android Developers. *Android Debug Bridge (adb)*. 2020. URL: https://developer.android.com/studio/command-line/adb (Last accessed 03/26/2022) (cit. on p. 38).

[DFL14] Drew Davidson, Matt Fredrikson, and Benjamin Livshits. "MoRePriv: Mobile OS Support for Application Personalization and Privacy". In: *Proceedings of the 30th Annual Computer Security Applications Conference*. ACSAC '14. ACM, 2014, pp. 236–245. DOI: 10.1145/2664243.2664266 (cit. on p. 30).

[DH76] W. Diffie and M. Hellman. "New directions in cryptography". In: *Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638 (cit. on p. 27).

[Dim20] Tassos Dimitriou. "Efficient, Coercion-free and Universally Verifiable Blockchain-based Voting". In: *Computer Networks* 174 (2020). DOI: 10.1016/j.comnet.2020.107234 (cit. on p. 31).

[DL+12] Alexander De Luca, Alina Hang, Frederik Brudy, Christian Lindner, and Heinrich Hussmann. "Touch Me Once and I Know It's You!: Implicit Authentication Based on Touch Screen Patterns". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. ACM, 2012, pp. 987–996. DOI: 10.1145/2207676.2208544 (cit. on p. 28).

[Dou21] Zhongchen Dou. "The Text Captcha Solver: A Convolutional Recurrent Neural Network-Based Approach". In: *International Conference on Big Data Analysis and Computer Science (BDACS)*. IEEE, 2021, pp. 273–283. DOI: 10.1109/BDACS53596.2021.00067 (cit. on p. 28).

[Dow+16] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. Vol. 48. ICML'16. JMLR.org, 2016, pp. 201–210 (cit. on p. 30).

[Dro20] Subramanyam Dronamraju. *Nearly 90% Of Businesses Fail To Comply With California Consumer Privacy Act (CCPA), Says New Research By InfoSecEnforcer*. Oct. 2020. URL: https://infosecenforcer.com/nearly-90-of-businesses-fail-to-comply-with-california-consumer-privacy-act-ccpa-says-new-research-by-infosecenforcer/ (Last accessed 03/26/2022) (cit. on p. 2).

[Dzi19] Josh Dzieza. *Why CAPTCHAS have gotten so difficult*. 2019. URL: https://www.theverge.com/2019/2/1/18205610/google-captcha-ai-robot-human-difficult-artificial-intelligence (Last accessed 03/26/2022) (cit. on p. 7).

[EC18] Electoral-Comission. *The administration of the June 2017 UK general election*. 2018. URL: https://www.electoralcommission.org.uk/__data/assets/pdf_file/0003/238044/The-administration-of-the-June-2017-UK-general-election.pdf (Last accessed 03/26/2022) (cit. on p. 8).

[Eco16] The Economist. *The world's most valuable resource is no longer oil, but data*. May 2016. URL: https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data (Last accessed 03/26/2022) (cit. on p. 3).

[EG85] Taher El Gamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Proceedings of CRYPTO 84 on Advances in Cryptology*. Ed. by David Chaum George R. Blakley. Vol. 196. LNCS. Springer, Berlin,

Heidelberg, 1985, 10–18. DOI: 10.1007/3-540-39568-7_2 (cit. on pp. 17, 27).

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. "A Randomized Protocol for Signing Contracts". In: *Commun. ACM* 28.6 (June 1985), 637–647. DOI: 10.1145/3812.3818 (cit. on p. 27).

[ET18] Jacob Eberhardt and Stefan Tai. "ZoKrates - Scalable Privacy-Preserving Off-Chain Computations". In: *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1084–1091. DOI: 10.1109/Cybermatics_2018.2018.00199 (cit. on p. 56).

[Feh+12] Clint Feher, Yuval Elovici, Robert Moskovitch, Lior Rokach, and Alon Schclar. "User identity verification via mouse dynamics". In: *Information Sciences* 201 (2012), pp. 19–36. DOI: 10.1016/j.ins.2012.02.066 (cit. on p. 141).

[Fen+20] Yunhe Feng, Qing Cao, Hairong Qi, and Scott Ruoti. "SenCAPTCHA: A Mobile-First CAPTCHA Using Orientation Sensors". In: *Proceedings ACM Interactive Mobile Wearable Ubiquitous Technologies* 4.2 (2020). DOI: 10.1145/3397312 (cit. on p. 28).

[FG20] Alisa Frik and Alexia Gaudeul. "A measure of the implicit value of privacy under risk". In: *Journal of Consumer Marketing* 37 (2020), pp. 457–472. DOI: 10.1108/JCM-06-2019-3286 (cit. on p. 2).

[Fou16] Interaction Design Foundation. *Killing the CAPTCHA for better UX*. 2016. URL: https://www.interaction-design.org/literature/article/killing-the-captcha-for-better-ux (Last accessed 03/26/2022) (cit. on p. 29).

[Fre19] FreePrivacyPolicy. *Privacy Policy for ReCAPTCHA*. 2019. URL: https://www.freeprivacypolicy.com/blog/recaptcha-privacy-policy/ (Last accessed 03/26/2022) (cit. on p. 7).

[Fre97] Louis J. Freeh. *Statement of Louis J. Freeh, Director Federal Bureau of Investigation Before the Senate Judiciary Committee*. 1997. URL: https://archive.epic.org/crypto/legislation/freeh_797.html (Last accessed 03/26/2022) (cit. on p. 3).

[FS87] Amos Fiat and Adi Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology — CRYPTO' 86*. Ed. by Andrew M. Odlyzko. LNCS. Springer Berlin Heidelberg, 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12 (cit. on pp. 22, 24, 42).

[GCF11] Saikat Guha, Bin Cheng, and Paul Francis. "Privad: Practical Privacy in Online Advertising". In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 169–182 (cit. on p. 30).

[Gjø10] Kristian Gjøsteen. *Analysis of an Internet Voting Protocol*. Tech. rep. https://eprint.iacr.org/2010/380.pdf. 2010 (cit. on pp. 33, 135).

[GLN12] Thore Graepel, Kristin Lauter, and Michael Naehrig. "ML Confidential: Machine Learning on Encrypted Data". In: *Information Security and Cryptology – ICISC 2012*. Ed. by Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon. Vol. 7839. LNCS. Springer Berlin Heidelberg, 2012, pp. 1–21. DOI: 10.1007/978-3-642-37682-5_1 (cit. on p. 30).

[GMR85] S Goldwasser, S Micali, and C Rackoff. "The Knowledge Complexity of Interactive Proof-systems". In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC '85. ACM, 1985, pp. 291–304. DOI: 10.1145/22145.22178 (cit. on pp. 22, 27).

[GN16] Ruti Gafni and Idan Nagar. "CAPTCHA–Security affecting user experience". In: *Issues in Informing Science and Information Technology* 13 (2016), pp. 063–077. DOI: 10.28945/3469 (cit. on p. 7).

[Gol20] Jeffrey Goldberg. *Apple and Google's contact tracing is privacy-preserving*. 1Password. May 2020. URL: https://blog.1password.com/contact-tracing/ (Last accessed 03/26/2022) (cit. on p. 3).

[Goo+13] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. *Multi-digit number recognition from street view imagery using deep convolutional neural networks*. arXiv. 2013. URL: https://arxiv.org/abs/1312.6082 (cit. on p. 28).

[Goo14] Google. *Are you a robot? Introducing "No CAPTCHA reCAPTCHA"*. 2014. URL: https://security.googleblog.com/2014/12/are-you-robot-introducing-no-captcha.html (Last accessed 03/26/2022) (cit. on p. 28).

[Goo19] Google. *Choosing the type of reCAPTCHA*. 2019. URL: https://developers.google.com/recaptcha/docs/versions (Last accessed 03/26/2022) (cit. on p. 28).

[Goo21] Google. *SafetyNet reCAPTCHA API*. 2021. URL: https://developer.android.com/training/safetynet/recaptcha (Last accessed 03/26/2022) (cit. on p. 62).

[Gov20] United Kingdom Government. *How to vote: voting by post*. 2020. URL: https://www.gov.uk/voting-in-the-uk/postal-voting (Last accessed 03/26/2022) (cit. on p. 8).

[GPZ19] Panagiotis Grontas, Aris Pagourtzis, and Alexandros Zacharakis. *Security models for everlasting privacy*. Cryptology ePrint Archive, Report 2019/1193. https://eprint.iacr.org/2019/1193. 2019 (cit. on p. 84).

[Gri16] Jack Grigg. *Bellman: Zero-knowledge Cryptography in Rust*. 2016. URL: https://github.com/zkcrypto/bellman (Last accessed 03/26/2022) (cit. on p. 56).

[Gro09] Jens Groth. "Linear Algebra with Sub-linear Zero-Knowledge Arguments". In: *Advances in Cryptology - CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. LNCS. Springer Berlin Heidelberg, 2009, pp. 192–208. DOI: 10.1007/978-3-642-03356-8_12 (cit. on pp. 21, 31, 56).

[Gro10] Jens Groth. "A Verifiable Secret Shuffle of Homomorphic Encryptions". In: *Journal of Cryptology* 23.4 (Oct. 2010), pp. 546–579. DOI: 10.1007/s00145-010-9067-9 (cit. on p. 75).

[Gro16] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *Advances in Cryptology - EUROCRYPT 2016*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. LNCS. Springer, 2016, pp. 305–326. DOI: 10.1007/978-3-662-49896-5_11 (cit. on pp. 25, 30, 56).

[Gro+19] Panagiotis Grontas, Aris Pagourtzis, Alexandros Zacharakis, and Bingsheng Zhang. "Towards Everlasting Privacy and Efficient Coercion Resistance in Remote Electronic Voting". In: *Financial Cryptography and Data Security*. Ed. by Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala. Vol. 10958. LNCS. Springer Berlin Heidelberg, 2019, pp. 210–231. DOI: 10.1007/978-3-662-58820-8_15 (cit. on pp. 31, 34).

[Gue+15] Meriem Guerar, Mauro Migliardi, Alessio Merlo, Mohamed Benmohammed, and Belhadri Messabih. "A completely automatic public physical test to tell computers and humans apart: A way to enhance authentication schemes in mobile devices". In: *International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2015, pp. 203–210. DOI: 10.1109/HPCSim.2015.7237041 (cit. on pp. 28, 29).

[Gue+16] Meriem Guerar, Mauro Migliardi, Alessio Merlo, Mohamed Benmohammed, Francesco Palmieri, and Aniello Castiglione. "Using screen brightness to improve security in mobile social network access". In: *IEEE Transactions on Dependable and Secure Computing* 15.4 (2016), pp. 621–632. DOI: 10.1109/TDSC.2016.2601603 (cit. on p. 28).

[Gue+18] Meriem Guerar, Alessio Merlo, Mauro Migliardi, and Francesco Palmieri. "Invisible CAPPCHA: A usable mechanism to distinguish between malware and humans on the mobile IoT". In: *Computers and Security* 78 (2018), pp. 255–266. DOI: 10.1016/j.cose.2018.06.007 (cit. on pp. 7, 28, 29, 37, 139).

[GVUSS14] Manuel Gimeno, Blanca Villamía-Uriarte, and Víctor Suárez-Saa. *eEspaña - Informe anual sobre el desarrollo de la sociedad de la información en España*. 2014. URL: https://www.proyectosfundacionorange.es/docs/eE2014/Informe_eE2014.pdf (Last accessed 03/26/2022) (cit. on p. 9).

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Report 2019/953. https://eprint.iacr.org/2019/953. 2019 (cit. on p. 25).

[HA+13] Jorge L. Hernandez-Ardieta, Ana I. Gonzalez-Tablas, Jose M. De Fuentes, and Benjamin Ramos. "A Taxonomy and Survey of Attacks on Digital Signatures". In: *Comput. Secur.* 34 (2013), pp. 67–112. DOI: 10.1016/j.cose.2012.11.009 (cit. on p. 9).

[HA+21a] Luis Hernández-Álvarez, José María de Fuentes, Lorena González-Manzano, and Luis Hernández Encinas. "Smart-CAMPP - Smartphone-based continuous authentication leveraging motion sensors with privacy preservation". In: *Pattern Recognition Letters* 147 (2021), pp. 189–196. DOI: 10.1016/j.patrec.2021.04.013 (cit. on p. 7).

[HA+21b] Luis Hernández-Álvarez, José María de Fuentes, Lorena González-Manzano, and Luis Hernández Encinas. "Privacy-Preserving Sensor-Based Continuous Authentication and User Profiling: A Review". In: *Sensors* 21.1 (2021). DOI: 10.3390/s21010092 (cit. on p. 7).

[Hel20] HelpNetSecurity. *85% of COVID-19 tracking apps leak data*. Sept. 2020. URL: https://www.helpnetsecurity.com/2020/09/30/covid-19-tracking-apps-leak-data/ (Last accessed 03/26/2022) (cit. on p. 3).

[HIT16] Manik Hapsara, Ahmed Imran, and Timothy Turner. "E-Voting in Developing Countries". In: *International Joint Confrence on Electronic Voting - EVOTE ID*. Ed. by Robert Krimmer, Melanie Volkamer, Jordi Barrat, Josh Benaloh, Nicole Goodman, Peter Y. A. Ryan, and Vanessa Teague. Vol. 10141. LNCS. Springer, 2016, pp. 36–55. DOI: 10.1007/978-3-319-52240-1_3 (cit. on p. 8).

[HKH16] Thomas Hupperich, Katharina Krombholz, and Thorsten Holz. "Sensor Captchas: On the Usability of Instrumenting Hardware Sensors to Prove Liveliness". In: *International Conference on Trust and Trustworthy Computing*. Ed. by Michael Franz and Panos Papadimitratos. Vol. 9824. LNCS. Springer International Publishing, 2016, pp. 40–59. DOI: 10.1007/978-3-319-45572-3_3 (cit. on pp. 28, 29).

[HKR19] Max Hoffmann, Michael Klooß, and Andy Rupp. "Efficient Zero-Knowledge Arguments in the Discrete Log Setting, Revisited". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. Association for Computing Machinery, 2019, 2093–2110. DOI: 10.1145/3319535.3354251 (cit. on p. 26).

[HL09] James Heather and David Lundin. "The Append-Only Web Bulletin Board". In: *Formal Aspects in Security and Trust*. Ed. by Pierpaolo Degano, Joshua Guttman, and Fabio Martinelli. Vol. 5491. LNCS. Springer Berlin Heidelberg, 2009, pp. 242–256. DOI: 10.1007/978-3-642-01465-9_16 (cit. on p. 76).

[Hol+19] Scott Hollier, Janina Sajka, Jason White, and Michael Cooper. *Inaccessibility of CAPTCHA: Alternatives to Visual Turing Tests on the Web*. 2019. URL: https://www.w3.org/TR/turingtest/ (Last accessed 03/26/2022) (cit. on pp. 7, 29).

[Hor19] Louise Horton. *Businesses failing to comply with GDPR*. Feb. 2019. URL: https://www.chdlimited.com/2019/02/05/businesses-failing-comply-gdpr/ (Last accessed 03/26/2022) (cit. on p. 2).

[Hug19] Matthew Hughes. *Bots drove nearly 40% of internet traffic last year - and the naughty ones are getting smarter*. 2019. URL: https://thenextweb.com/security/2019/04/17/bots-drove-nearly-40-of-internet-traffic-last-year-and-the-naughty-ones-are-getting-smarter/ (Last accessed 03/26/2022) (cit. on p. 6).

[Ily] Ilya Khrennikov. *Telegram loses bid to block Russia from encryption keys*. URL: https://www.bloomberg.com/news/articles/2018-03-20/telegram-loses-bid-to-stop-russia-from-getting-encryption-keys (Last accessed 03/26/2022) (cit. on p. 3).

[Int19] Intuition Machines Inc. *hCaptcha: Earn money with a captcha*. 2019. URL: https://www.hcaptcha.com (cit. on pp. 7, 29).

[Iri18] Roberto Iriondo. *Breaking CAPTCHA Using Machine Learning in 0.05 Seconds*. 2018. URL: https://medium.com/towards-artificial-intelligence/breaking-

captcha-using-machine-learning-in-0-05-seconds-9feefb997694 (Last accessed 03/26/2022) (cit. on p. 7).

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. "Coercion-resistant Electronic Elections". In: *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*. WPES '05. ACM, 2005, pp. 61–70. DOI: 10.1145/1102199.1102213 (cit. on pp. 31, 32, 85, 88).

[JKP10] Muhammad Asim Jamshed, Wonho Kim, and KyoungSoo Park. "Suppressing bot traffic with accurate human attestation". In: *Proceedings of the first ACM asia-pacific workshop on Workshop on systems*. SIGCOMM '10. Association for Computing Machinery, 2010, pp. 43–48. DOI: 10.1145/1851276.1851287 (cit. on p. 7).

[JMO21a] Solly Ross Joel Martin Samuel Mannehed and Pierre Ossman. *NoVNC - the open source VNC client*. 2021. URL: https://github.com/novnc/noVNC (Last accessed 03/26/2022) (cit. on p. 62).

[JMO21b] Solly Ross Joel Martin Samuel Mannehed and Pierre Ossman. *NoVNC: HTML VNC Client Library and Application*. 2021. URL: https://novnc.com/noVNC/ (Last accessed 03/26/2022) (cit. on p. 62).

[Jus18] State of California Department of Justice. *California Consumer Privacy Act (CCPA)*. 2018. URL: https://oag.ca.gov/privacy/ccpa (Last accessed 03/26/2022) (cit. on p. 2).

[Kah20] Richard Kahn. *CAPTCHA and reCAPTCHA: How Fraudsters Bypass It*. 2020. URL: https://www.anura.io/blog/how-the-use-of-captcha-can-hurt-user-experience (Last accessed 03/26/2022) (cit. on p. 29).

[Lee18] Gregory T Lee. *Abstract Algebra. An Introductory Course*. Springer, 2018. DOI: 10.1007/978-3-319-77649-1 (cit. on p. 13).

[LH18] Paul Lewis and Paul Hilder. *Leaked: Cambridge Analytica's blueprint for Trump victory*. Mar. 2018. URL: https://www.theguardian.com/uk-news/2018/mar/23/leaked-cambridge-analyticas-blueprint-for-trump-victory (Last accessed 03/26/2022) (cit. on pp. 1, 3).

[LHK16] Philipp Locher, Rolf Haenni, and Reto E Koenig. "Coercion-Resistant Internet Voting with Everlasting Privacy". In: *Financial Cryptography and Data Security*. Vol. 9604. LNCS. Springer Berlin Heidelberg, 2016, pp. 161–175. DOI: 10.1007/978-3-662-53357-4_11 (cit. on pp. 33, 34).

[Liu18] Wei Liu. *Introducing reCAPTCHA v3: the new way to stop bots*. 2018. URL: https://webmasters.googleblog.com/2018/10/introducing-recaptcha-v3-new-way-to.html (Last accessed 03/26/2022) (cit. on pp. 7, 29).

[Loz19] Andrea L. Lozano. *Why is CAPTCHA killing your conversion rate?* 2019. URL: https://blog.arengu.com/why-captcha-is-killing-your-conversion-rate/ (Last accessed 03/26/2022) (cit. on p. 29).

[LQAT20] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. "VOTEAGAIN: A Scalable Coercion-Resistant Voting System". In: *Proceedings of the 29th USENIX Conference on Security Symposium*. USA: USENIX Association, 2020 (cit. on pp. v, 71).

[LV20] Isis Agora Lovecruft and Henry de Valence. *curve25519-dalek*. Version 2.0.0. 2020. URL: https://crates.io/crates/curve25519-dalek/2.0.0 (cit. on p. 56).

[Mal+18] Mohammad Malekzadeh, Richard G Clegg, Andrea Cavallaro, and Hamed Haddadi. "Protecting Sensory Data Against Sensitive Inferences". In: *Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems*. W-P2DS'18. ACM, 2018, 2:1–2:6. DOI: 10.1145/3195258.3195260 (cit. on p. 29).

[Mal19] Daniyal Malik. *Global Ad-Blocking Behaviors In 2019 - Stats & Consumer Trends*. 2019. URL: https://www.digitalinformationworld.com/2019/04/global-ad-blocking-behaviors-infographic.html (Last accessed 03/26/2022) (cit. on p. 4).

[Mal+19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. Association for Computing Machinery, 2019, 2111–2128. DOI: 10.1145/3319535.3339817 (cit. on p. 25).

[Mar16] Bernard Marr. *Why Data Minimization Is An Important Concept In The Age of Big Data*. Mar. 2016. URL: https://www.forbes.com/sites/bernardmarr/2016/03/16/why-data-minimization-is-an-important-concept-in-the-age-of-big-data/ (Last accessed 03/26/2022) (cit. on p. 4).

[Mar19] Marea Granate. *Calendario electoral voto exterior 2019*. https://mareagranate.org/2019/02/calendario-electoral-voto-exterior-2019/. 2019 (cit. on p. 73).

[MCC08] Andrew C Myers, Michael Clarkson, and Stephen Chong. "Civitas: Toward a Secure Voting System". In: *IEEE Symposium on Security and Privacy*. IEEE, 2008, pp. 354–368. DOI: 10.1109/SP.2008.32 (cit. on p. 31).

[Mes+11] Arik Messerman, Tarik Mustafić, Seyit Ahmet Camtepe, and Sahin Albayrak. "Continuous and non-intrusive identity verification in real-time environments based on free-text keystroke dynamics". In: *International Joint Conference on Biometrics (IJCB)*. IJCB '11. IEEE Computer Society, 2011, pp. 1–8. DOI: 10.1109/IJCB.2011.6117552 (cit. on p. 141).

[MIB19] David McCabe, Mike Isaac, and Katie Benner. *Facebook and Barr Escalate Standoff Over Encrypted Messages*. 2019. URL: https://www.nytimes.com/2019/12/10/technology/whatsapp-barr-encryption.html (Last accessed 03/26/2022) (cit. on p. 2).

[Mie+13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. "Zerocoin: Anonymous Distributed E-Cash from Bitcoin". In: *Symposium on Security and Privacy*. IEEE, 2013. DOI: 10.1109/SP.2013.34 (cit. on p. 27).

[Min] Ministerio del Interior. *Voto desde fuera de España*. URL: https://infoelectoral.interior.gob.es/opencms/es/proceso-electoral/preguntas-frecuentes/como-votar/voto-desde-fuera-de-espana/ (Last accessed 03/26/2022) (cit. on p. 8).

[Min06] Ministerio del Interior. *La Subsecretaria Soledad López explica en el Senado las características del DNI electrónico*. 2006. URL: https://goo.gl/r6MVYJ (Last accessed 03/26/2022) (cit. on p. 9).

[Min17] Web Accessibility In Mind. *Screen Reader User Survey #7 Results*. 2017. URL: https://webaim.org/projects/screenreadersurvey7/ (Last accessed 03/26/2022) (cit. on p. 29).

[Min20] Ministère de l'Europe et des Affaires étrangères. *Vote par correspondance*. 2020. URL: https://www.diplomatie.gouv.fr/fr/services-aux-francais/voter-a-l-etranger/modalites-de-vote/vote-par-correspondance/ (Last accessed 03/26/2022) (cit. on p. 8).

[MK17] Shailin Dhar Mikko Kotila Ruben Cuevas Rumin. *Compendium of ad fraud knowledge for media investors*. 2017. URL: https://wfanet.org/knowledge/item/2016/06/03/Compendium-of-ad-fraud-knowledge-for-media-investors (Last accessed 03/26/2022) (cit. on p. 6).

[MN06] Tal Moran and Moni Naor. "Receipt-Free Universally-Verifiable Voting with Everlasting Privacy". In: *Proceedings of the 26th Annual International Conference on Advances in Cryptology - CRYPTO*. Vol. 4117. LNCS. Springer-Verlag, 2006, 373–392. DOI: 10.1007/11818175_22 (cit. on p. 34).

[Mon19] Monsoon Solutions Inc. *High Voltage Power Monitor*. 2019. URL: https://www.msoon.com/online-store/High-Voltage-Power-Monitor-Part-Number-AAA10F-p90002590 (Last accessed 03/26/2022) (cit. on p. 58).

[MSH17] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. "A Smart Contract for Boardroom Voting with Maximum Voter Privacy". In: *Financial Cryptography and Data Security*. Ed. by Aggelos Kiayias. Vol. 10322. LNCS. Springer, 2017, pp. 357–375. DOI: 10.1007/978-3-319-70972-7_20 (cit. on p. 76).

[MW19] Joe Myers and Kate Whiting. *These are the biggest risks facing our world in 2019*. Jan. 2019. URL: https://www.weforum.org/agenda/2019/01/these-are-the-biggest-risks-facing-our-world-in-2019/ (Last accessed 2021) (cit. on p. 4).

[Nem+17] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas. "The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli". In: *24th ACM Conference on Computer and Communications Secu-*

rity (CCS'2017). ACM, 2017, pp. 1631–1648. DOI: 10.1145/3133956.3133969 (cit. on p. 9).

[Net+18] André Silva Neto, Matheus Leite, Roberto Araújo, Marcelle Pereira Mota, Nelson Cruz Sampaio Neto, and Jacques Traoré. "Usability Considerations For Coercion-Resistant Election Systems". In: IHC 2018 (2018). DOI: 10.1145/3274192.3274232 (cit. on p. 73).

[Net20] Nym Network. *Incentivised test-net*. 2020. URL: https://nymtech.net/docs/stable/run-nym-nodes/incentives/ (cit. on p. 27).

[O'R15] Lara O'Reilly. *Google's new CAPTCHA security login raises 'legitimate privacy concerns'*. 2015. URL: https://www.businessinsider.com/google-no-captcha-adtruth-privacy-research-2015-2?r=US&IR=T (Last accessed 03/26/2022) (cit. on p. 28).

[Ove18] Jarrod Overson. *Bypassing CAPTCHAs with Headless Chrome*. 2018. URL: https://medium.com/@jsoverson/bypassing-captchas-with-headless-chrome-93f294518337 (Last accessed 03/26/2022) (cit. on p. 7).

[Pap+17] Elias P. Papadopoulos, Michalis Diamantaris, Panagiotis Papadopoulos, Thanasis Petsas, Sotiris Ioannidis, and Evangelos P. Markatos. "The Long-Standing Privacy Debate: Mobile Websites vs Mobile Apps". In: *Proceedings of the 26th International Conference on World Wide Web*. WWW '17. 2017, pp. 153–162. DOI: 10.1145/3038912.3052691 (cit. on p. 29).

[Par+09] SangHwan Park, Haeryong Park, YooJae Won, JaeIl Lee, and Stephen Kent. *Traceable Anonymous Certificate*. Tech. rep. 5636. Internet Engineering Task Force, 2009 (cit. on p. 76).

[Ped91a] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*. Ed. by Joan Feigenbaum. Vol. 576. CRYPTO '91. Springer-Verlag, 1991, 129–140. DOI: 10.1007/3-540-46766-1_9 (cit. on p. 21).

[Ped91b] Torben Pryds Pedersen. "A Threshold Cryptosystem without a Trusted Party". In: *Advances in Cryptology—EUROCRYPT'91*. Ed. by Donald W. Davies. Vol. 547. LNCS. Springer Berlin Heidelberg, 1991, pp. 522–526. DOI: 10.1007/3-540-46416-6_47 (cit. on p. 20).

[Pes+20] Gonçalo Pestana, Iñigo Querejeta-Azurmendi, Panagiotis Papadopoulos, and Benjamin Livshits. *THEMIS: Decentralized and Trustless Ad Platform with Reporting Integrity*. 2020. arXiv: 2007.05556 [cs.CR] (cit. on p. vi).

[Phi15] Drew Phillips. *What is Securimage?* 2015. URL: https://www.phpcaptcha.org/ (Last accessed 03/26/2022) (cit. on pp. 7, 29).

[Pre20] Ted Pretty. *Terrorist Attacks in France and Austria are the Cause of the New Changes in the Encryption Field*. Nov. 2020. URL: https://cipherpoint.com/blog/encryption-ban-france-austria/ (Last accessed 03/26/2022) (cit. on p. 3).

[QA+17] Iñigo Querejeta-Azurmendi, Jorge L. Hernandez-Ardieta, Víctor Gayoso Martínez, Luis Hernandez Encinas, and David Arroyo. "A coercion-resistant and easy-to-use Internet e-voting protocol based on traceable anonymous certificates". In: *III Jornadas Nacionales de Investigación en Ciberseguridad*. Selected **Best Research Article**. May 2017 (cit. on pp. iv, 71).

[QA+19] Iñigo Querejeta-Azurmendi, Luis Hernández Encinas, David Arroyo Guardeño, and Jorge L. Hernandez-Ardieta. "An Internet Voting Proposal Towards Improving Usability and Coercion Resistance." In: *CISIS-ICEUTE*. Vol. 951. Advances in Intelligent Systems and Computing. Springer, 2019, pp. 155–164. DOI: 10.1007/978-3-030-20005-3_16 (cit. on p. iv).

[QA+20] Iñigo Querejeta-Azurmendi, David Arroyo Guardeño, Jorge L. Hernández-Ardieta, and Luis Hernández Encinas. "NetVote: A Strict-Coercion Resistance Re-Voting Based Internet Voting Scheme with Linear Filtering". In: *Mathematics* 8.9 (2020). DOI: 10.3390/math8091618 (cit. on pp. iv, 71).

[QAHEHA18] Iñigo Querejeta-Azurmendi, Luis Hernández Encinas, and Jorge L. Hernández Ardieta. "Don't shoot the messenger, How a trusted channel may not be a necessary assumption for remote code-voting". In: *IV Jornadas Nacionales de Investigación en Ciberseguridad*. 2018 (cit. on p. vi).

[Que17] Iñigo Querejeta. "A different approach to code voting". In: *PhD Colloquium of Electronic Voting*. Ed. by Robert Krimmer, Melanie Volkamer, Nadja Braun Binder, Norbert Kersting, Olivier Pereira, and Carsten Schürmann. Springer International Publishing, 2017 (cit. on p. vi).

[Que+21] Iñigo Querejeta-Azurmendi, Panagiotis Papadopoulos, Matteo Varvello, Antonio Nappa, Jiexin Zhang, and Benjamin Livshits. "ZKSENSE: A Friction-less Privacy-Preserving Human Attestation Mechanism for Mobile Devices". In: *Proc. Priv. Enhancing Technol.* 2021.4 (2021), pp. 6–29. DOI: 10.2478/popets-2021-0058 (cit. on pp. v, 35, 43).

[RC13] Gerardo Reynaga and Sonia Chiasson. "The usability of CAPTCHAs on smartphones". In: *2013 International Conference on Security and Cryptography (SECRYPT)*. IEEE, 2013, pp. 1–8. URL: https://ieeexplore.ieee.org/document/7223194 (cit. on p. 28).

[Reu17] Reuters. *France drops electronic voting for citizens abroad over cybersecurity fears*. 2017. URL: http://www.reuters.com/article/us-france-election-cyber-idUSKBN16D233 (Last accessed 03/26/2022) (cit. on p. 8).

[RO+16] Jorge-L. Reyes-Ortiz, Luca Oneto, Albert Samà, Xavier Parra, and Davide Anguita. "Transition-Aware Human Activity Recognition Using Smartphones". In: *Neurocomputing* 171 (2016). Ed. by T. Heskes, pp. 754–767. DOI: 10.1016/j.neucom.2015.07.085 (cit. on p. 29).

[Røn+20] Peter B. Rønne, Arash Atashpendar, Kristian Gjøsteen, and Peter Y. A. Ryan. "Short Paper: Coercion-Resistant Voting in Linear Time via Fully Homomorphic Encryption". In: *Financial Cryptography and Data Security*. Ed. by Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala. Vol. 11599. LNCS. Springer Cham, 2020, pp. 289–298. DOI: 10.1007/978-3-030-43725-1_20 (cit. on p. 135).

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications ACM* 21.2 (Feb. 1978). Ed. by Robert L. Ashenhurst, 120–126. DOI: 10.1145/359340.359342 (cit. on pp. 19, 27).

[Rya20] Johnny Ryan. *https://brave.com/dpa-report-2020/*. Apr. 2020. URL: https://brave.com/dpa-report-2020/ (Last accessed 03/26/2022) (cit. on p. 2).

[Sch19] Katharine Schwab. *Google's new reCAPTCHA has a dark side*. 2019. URL: https://www.fastcompany.com/90369697/googles-new-recaptcha-has-a-dark-side (Last accessed 03/26/2022) (cit. on pp. 7, 29).

[Sch22] Berry Schoenmakers. *Lecture Notes, Cryptographic Protocols*. https://www.win.tue.nl/~berry/CryptographicProtocols/LectureNotes.pdf. Technical University of Eindhoven, 2022. (Last accessed 03/26/2022) (cit. on p. 13).

[Sch90] Claus P. Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by Gilles Brassard. Vol. 435. LNCS. Springer New York, 1990, pp. 239–252. DOI: 10.1007/0-387-34805-0_22 (cit. on p. 19).

[Seb19] Armin Sebastian. *Buster: Captcha Solver for Humans*. 2019. URL: https://github.com/dessant/buster (Last accessed 03/26/2022) (cit. on p. 29).

[Seg19] Cuadernos de Seguridad. *Una de cada cuatro pymes europeas sufren brechas de datos en 2019*. Oct. 2019. URL: https://cuadernosdeseguridad.com/2019/10/informe-brecha-datos-pymes/ (Last accessed 03/26/2022) (cit. on p. 4).

[SHD10] Olivier Spycher, Rolf Haenni, and Eric Dubuis. "Coercion-resistant hybrid voting systems". In: *4th International Conference - EVOTE*. Ed. by Robert Krimmer and Rüdiger Grimm. Vol. 167. LNI. GI, 2010, pp. 269–282 (cit. on p. 33).

[Sig21] Signal Technology Foundation. *Signal*. Version 5.3.12. Feb. 2021. URL: https://signal.org/ (cit. on p. 27).

[Smi05] W.D. Smith. "New cryptographic election protocol with best-known theoretical properties". In: *Workshop Frontiers in Electronic Elections (FEE)*. 2005 (cit. on p. 32).

[SOG20] SOG-IS. *SOG-IS Crypto Evaluation Scheme. Agreed Cryptographic Mechanisms. Version 1.2*. https://www.sogis.eu/uk/supporting_doc_en.html. Senior Officials Group-Information Systems Security, Crypto Working Group. 2020 (cit. on p. 9).

[SPK16] Suphannee Sivakorn, Iasonas Polakis, and Angelos D. Keromytis. "I Am Robot: (Deep) Learning to Break Semantic Image Captchas". In: *IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2016, pp. 388–403. DOI: 10.1109/EuroSP.2016.37 (cit. on pp. 7, 28).

[SRA81] Adi Shamir, Ronald L. Rivest, and Leonard M. Adleman. "Mental Poker". In: *The Mathematical Gardner*. Ed. by David A. Klarner. Springer US, 1981, pp. 37–43. DOI: 10.1007/978-1-4684-6686-7_5 (cit. on p. 27).

[SS+18] Rubén San-Segundo, Henrik Blunck, José Moreno-Pimentel, Allan Stisen, and Manuel Gil-Martín. "Robust Human Activity Recognition using smartwatches and smartphones". In: *Engineering Applications of Artificial Intelligence* 72 (2018). Ed. by A. Abraham, pp. 190–202. DOI: 10.1016/j.engappai.2018.04.002 (cit. on p. 29).

[SSH13] Babins Shrestha, Nitesh Saxena, and Justin Harrison. "Wave-to-Access: Protecting Sensitive Mobile Device Services via a Hand Waving Gesture". In: *Cryptology and Network Security*. Ed. by Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab. Vol. 8257. LNCS. Springer International Publishing, 2013, pp. 199–217. DOI: 10.1007/978-3-319-02937-5_11 (cit. on pp. 28, 29).

[Str21] Sebastian Strangio. *Singapore Backtracks on COVID-19 Tracking App Privacy Pledge*. Jan. 2021. URL: https://thediplomat.com/2021/01/singapore-backtracks-on-covid-19-tracking-app-privacy-pledge/ (Last accessed 03/26/2022) (cit. on p. 3).

[Sü21] Ahmet Ali Süzen. "UNI-CAPTCHA: A Novel Robust and Dynamic User-Non-Interaction CAPTCHA Model Based on Hybrid biLSTM+Softmax". In: *Journal of Information Security and Applications* 63 (2021). DOI: 10.1016/j.jisa.2021.103036 (cit. on p. 28).

[Tam+09] Jennifer Tam, Jiri Simsa, Sean Hyde, and Luis V Ahn. "Breaking audio captchas". In: *Advances in Neural Information Processing Systems*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou. Curran Associates Inc., 2009, pp. 1625–1632 (cit. on p. 29).

[Tas+12] Aimilia Tasidou, Pavlos S Efraimidis, Yannis Soupionis, Lilian Mitrou, and Vasilios Katos. "Privacy-preserving, user-centric VoIP CAPTCHA challenges: An integrated solution in the SIP environment." In: *Information and Computer Security*. Vol. 24. 1. Emerald Group Publishing Limited, 2012, pp. 2–19. DOI: 10.1108/ICS-07-2014-0046 (cit. on p. 28).

[The17] The Guardian. *Dutch will count all election ballots by hand to thwart hacking*. 2017. URL: https://www.theguardian.com/world/2017/feb/02/dutch-will-count-all-election-ballots-by-hand-to-thwart-cyber-hacking (Last accessed 03/26/2022) (cit. on p. 8).

[Thr18] ThreatMetrix. *H2 2018 Cybercrime Report*. 2018. URL: https://www.threatmetrix.com/info/h2-2018-cybercrime-report/ (Last accessed 03/26/2022) (cit. on p. 6).

[Tro+20] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, Ludovic Barman, Sylvain Chatel, Kenneth Paterson, Srdjan Čapkun, David Basin, Jan Beutel, Dennis Jackson, Marc Roeschlin, Patrick Leu, Bart Preneel, Nigel Smart, Aysajan Abidin, Seda Gürses, Michael Veale, Cas Cremers, Michael Backes, Nils Ole Tippenhauer, Reuben Binns, Ciro Cattuto, Alain Barrat, Dario Fiore, Manuel Barbosa, Rui Oliveira, and José Pereira. *Decentralized Privacy-Preserving Proximity Tracing*. arXiv. https://arxiv.org/abs/2005.12273. 2020 (cit. on p. 3).

[TVD17] Theja Tulabandhula, Shailesh Vaya, and Aritra Dhar. *Privacy-preserving Targeted Advertising*. arXiv. 2017. URL: http://arxiv.org/abs/1710.03275 (cit. on p. 30).

[Val+15] Luke Valenta, Shaanan Cohney, Alex Liao, Joshua Fried, Satya Bodduluri, and Nadia Heninger. *Factoring as a Service*. Cryptology ePrint Archive, Report 2015/1000. https://eprint.iacr.org/2015/1000. 2015 (cit. on p. 3).

[Var+19] Matteo Varvello, Kleomenis Katevas, Wei Hang, Mihai Plesa, Hamed Haddadi, Fabián E. Bustamante, and Benjamin Livshits. "BatteryLab, a Distributed Power Monitoring Platform for Mobile Devices: Demo Abstract". In: *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. SenSys '19. Association for Computing Machinery, 2019, 386–387. DOI: 10.1145/3356250.3361946 (cit. on p. 58).

[Var+21] Matteo Varvello, Iñigo Querejeta Azurmendi, Antonio Nappa, Panagiotis Papadopoulos, Gonçalo Pestana, and Benjamin Livshits. "VPN-Zero: A Privacy-Preserving Decentralized Virtual Private Network". In: *2021 IFIP Networking Conference (IFIP Networking)*. 2021, pp. 1–6. DOI: 10.23919/IFIPNetworking52078.2021.9472843 (cit. on p. vi).

[VCA04] Ana Viseu, Andrew Clement, and Jane Aspinall. "Situating Privacy Online". In: *Information, Communication & Society* 7.1 (2004), pp. 92–114. DOI: 10.1080/1369118042000208924 (cit. on p. 1).

[VYA20] Henry de Valence, Cathie Yun, and Oleg Andreev. *Bulletproofs*. Version 2.0.0. 2020. URL: https://crates.io/crates/bulletproofs (cit. on p. 56).

[Win19] Davey Winder. *Data Breaches Expose 4.1 Billion Records In First Six Months Of 2019*. Aug. 2019. URL: https://www.forbes.com/sites/daveywinder/2019/08/20/data-breaches-expose-41-billion-records-in-first-six-months-of-2019/ (Last accessed 03/26/2022) (cit. on p. 4).

[WKY10] Chamila Walgampaya, Mehmed Kantardzic, and Roman Yampolskiy. "Real time click fraud prevention using multi-level data fusion". In: *Proceedings of the World Congress on Engineering and Computer Science*. Vol. 1. 2010, pp. 20–22 (cit. on p. 28).

[Ye+21] Kaili Ye, Dong Zheng, Rui Guo, Jiayu He, Yushuang Chen, and Xiaoling Tao. "A Coercion-Resistant E-Voting System Based on Blockchain Technology". In: *International Journal of Network Security* 23.5 (2021), pp. 791–806. DOI: 10.6633/IJNS.202109_23(5).06 (cit. on p. 31).

[YEA08] Jeff Yan and Ahmad Salah El Ahmad. "A Low-cost Attack on a Microsoft CAPTCHA". In: *Proceedings of the 15th ACM conference on Computer and communications security*. ACM. Machinery, New York, 2008, pp. 543–554. DOI: 10.1145/1455770.1455839 (cit. on pp. 7, 28).

[ZBS19] Jiexin Zhang, Alastair R Beresford, and Ian Sheret. "SensorID: Sensor Calibration Fingerprinting for Smartphones". In: *Proceedings of the 40th IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019. DOI: 10.1109/SP.2019.00072 (cit. on p. 29).

[Zho+18] Yuan Zhou, Zesun Yang, Chenxu Wang, and Matthew Boutell. "Breaking Google reCaptcha V2". In: *Journal of Computational Sciences in Colleges* 34.1 (Oct. 2018), pp. 126–136 (cit. on p. 28).

[Zi+20] Yang Zi, Haichang Gao, Zhouhang Cheng, and Yi Liu. "An End-to-End Attack on Text CAPTCHAs". In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 753–766. DOI: 10.1109/TIFS.2019.2928622 (cit. on p. 28).

[ZoK19] ZoKrates community. *ZoKrates: A toolbox for zkSNARKs on Ethereum*. Github. https://github.com/Zokrates/ZoKrates. 2019 (cit. on p. 25).