Security *and* Privacy via Cryptography

Having your cake and eating it too

WOUTER LUEKS

ISBN: 978-94-92380-65-4
IPA dissertation series: 2017-08

Typeset using LaTeX. Printed by GVO drukkers & vormgevers.
Cover design by Loes Kema at GVO drukkers & vormgevers.

This research is supported by the research program Sentinels (www.sentinels.nl) as project 'Revocable Privacy' (10532). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

Security *and* Privacy via Cryptography

Having your cake and eating it too

Proefschrift ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. J.H.J.M. van Krieken,
volgens besluit van het college van decanen
in het openbaar te verdedigen op

maandag 9 oktober 2017
om 16:30 uur precies

door

Wouter Lueks

geboren op 28 december 1986
te Emmen

PROMOTOR

prof. dr. B.P.F. Jacobs

COPROMOTOR

dr. J.-H. Hoepman

MANUSCRIPTCOMMISSIE

prof. dr. E.R. Verheul
prof. dr. G. Danezis (University College London, Verenigd Koninkrijk)
prof. dr. I.A. Goldberg (University of Waterloo, ON, Canada)
prof. dr. ir. B. Preneel (KU Leuven, België)
dr. G. Neven (IBM Zürich, Zwitserland)

ACKNOWLEDGMENTS

It is hard to believe that with the writing of these acknowledgments I am finishing my PhD thesis, thus completing a project that I have been working on for the past five years. During this time, I had the pleasure to meet and work with many people. Without you, this thesis would not have been possible and life would probably have been boring. Therefore, a heartfelt thank you to all of you! (These thanks apply even more so to those of you who I forgot to mention explicitly.)

First of all, I would like to thank Jaap-Henk, my daily supervisor, for drawing me into the field of privacy, for allowing me to develop myself, for teaching me how to write, for his critical feedback, and for the many interesting discussions we had together (most of which I remember taking place on board a train).

I would also like to thank Bart, my promotor, for always having a listening ear, for helping me in the final stages of writing this thesis, but most of all, for welcoming me into the IRMA project—I still fondly remember our first meetings about it.

As part of the IRMA project, I had the pleasure to work with many people: Brinda, Gergely, Maarten, Pim, Roland, and Ronny. Thank you! During the last two years, I worked with Fabian and Sietse to turn IRMA into a viable product. Gentlemen: it has been a pleasure.

I had many wonderful colleagues in Nijmegen. In particular, Gergely, we went on many trips together and talked a lot. Yet, I too wish we could have talked more. Pim, it was a pleasure working with you on the IRMA project. Merel, while we did not talk often enough, I very much valued our little chats and 'tea breaks'. Thanks, Peter, for always taking the time to answer my questions. Finally, a warm thank you to all my office mates: Anna, Brinda, Fabian, Freek, Gergely, Merel, Paulan, and Sietse, who guaranteed that a visit to Nijmegen was always fun.

Nijmegen was not my only base of operations. In fact, I spent a lot of time at the Cyber Security & Robustness department of TNO in Groningen. I would like to thank my colleagues there for the many lunchtime discussions we had. In particular, Lenny, Henny, Paul, Reinder, Richard, Frank, Jan, and Gerben, thank you. In Groningen, I shared an office with Geert. I will miss our many discussions and his matter-of-fact views on the world. Finally, Maarten, I think it is safe to say we discussed life, the universe, and everything. I would not have wanted to miss it, and I am honored that you agreed to be one of my paranymphs.

Shortly after I began my PhD, the Privacy & Identity Lab was founded.

This multidisciplinary lab allowed me to meet a diverse group of people and showed me that the world revolves around more than technology alone. In particular, I would like to thank Marc, Ronald, Bert-Jaap, Jelte, Jaap-Henk, Dimitra, Paulan, Merel, Martin, Claudia, and Lorenzo for our discussions and collaborations.

In the spring of 2014, I visited Ian Goldberg at the University of Waterloo in Canada. Thanks, Ian, for hosting me, for introducing me to the world of systems security and for showing me how to write a paper without stressing. I am also happy to have had such a great group of fellow students in the CrySP lab: Jalaj, Erinn, Nik, Cecylia, Tao, Tariq, Sukhbir, and Sarah, thank you all for welcoming me to Canada.

I would also like to thank my committee for taking the time to read my thesis, and for providing me with valuable feedback.

This journey would not have been possible without the support of my friends. Lise and Erwin, Maarten, and Paul, thanks for the many visits, cookies and cakes, and discussions. Marie-Sarah, thanks for being my travel companion and showing me how to cut the clutter. Elena and Patricia, thank you for making me feel at home in Madrid. And finally, to the Muppets: Nynke, Erik, and René, thanks for the many trips we went on together, and the many wonderful evenings of cooking, playing games, and chats. It would not have been the same without you.

Finally, I would like to thank my family. Thanks, mum and dad, for always supporting me, whatever I chose to do. Thanks, Maarten, for showing me how to be serious and enjoy life at the same time.

Wouter Lueks
August 2017, Madrid

SUMMARY

Digital technologies play an ever-increasing role in our daily lives. However, these technologies are also frequently data driven. Governments and companies alike collect more and more data about us to offer better services, to fight crime and terrorism, and to prevent fraud. The result: we have less privacy than ever.

Yet, privacy is important. A lack of privacy harms individuals by limiting their freedom to live their personal lives and by limiting their personal development. The chilling effect of a lack of privacy harms people by changing how they behave. They change their behavior not because they do anything illegal, but because of how that behavior could be construed. The actual collection and aggregation of personal data enabled by a lack of privacy is, of course, similarly harmful. It can, for example, lead to exclusions, to incorrect conclusions being drawn about a person, or to unexpected spreading of personal information.

A lack of privacy, however, does not just harm individuals but also societies. Without the protection of privacy, it is much harder to develop the critical mindset that is so essential for a democratic society to function properly.

While privacy is important, there are many arguments why privacy should not be increased. In this thesis we focus on two common arguments against (increasing) privacy, and show that the situation is more nuanced.

The first of these arguments is that an increase of privacy results in a decrease of security. In Chapter 3 we reintroduce the notion of *revocable privacy* to show that it is possible to build systems that offer privacy and security simultaneously. As long as users follow the rules of the revocable privacy system, they are fully anonymous. Only if they violate the rules can their anonymity be reduced. To show the usefulness of this approach, Chapter 3 highlights scenarios that could benefit from revocable privacy, and indicates which systems already exist that implement this notion of revocable privacy.

A common reason for online platforms to disallow anonymous users is the potential for abuse. In Chapter 4 we introduce the revocable privacy system vote-to-link. It enables Wikipedia, and other online platforms, to allow anonymous access for editors (or users in general), while simultaneously enabling Wikipedia to recover from abuse by misbehaving users. By default a user's actions are unlinkable. However, if moderators deem an action to be abusive, they can vote on this action. Once

an action accumulates sufficient votes, the system can link all other actions by the same user within a limited time frame. In this way, the other potentially malicious actions by that user are distinguished from all remaining actions, and can thus quickly be examined or removed.

Whereas the vote-to-link system implements the rule that an action should not be marked malicious by too many moderators, the distributed encryption scheme from Chapter 5 implements the rule that a party should not cause events at too many different locations. The distributed encryption scheme solves the canvas cutters problem—canvas cutters are criminals that rob trucks parked along highway rest stops by cutting their canvas—in a privacy friendly manner by identifying only those cars that stop at many rest stops. We simplify Hoepman and Galindo's original distributed encryption scheme, add proper key evolution—this is essential in many scenarios—and propose a batched solution that is more efficient for small plaintext domains such as the set of license plates.

The second argument against privacy that this thesis addresses is that privacy friendly solutions are not practical. We show, however, that the vote-to-link system and the distributed encryption scheme are efficient enough to use in practice.

Attribute-based credentials (ABCs) are digital alternatives to identity documents, loyalty cards, etc. ABCs have strong privacy guarantees, however, to protect the security of the system when a credential carrier is lost, stolen or abused, it should be possible to revoke credentials. In Chapter 6 we propose the first privacy friendly revocation scheme for ABCs that is fast enough to be practical even when these ABCs are implemented on smart cards.

Private information retrieval (PIR) allows clients to retrieve records from a database, without the operators of that database learning which records it retrieved. Achieving these privacy properties is computationally intensive for the database servers. In Chapter 7 we show how we can batch queries from many clients to reduce the load on the database servers. This new scheme is efficient enough to apply PIR to certificate transparency, a system to detect misbehaving certificate authorities, thereby making it privacy friendly.

These results show that the two common arguments against privacy that we address in this thesis are not universally true. In fact, we show that we can build practical systems that achieve security *and* privacy simultaneously, showing that security can often be achieved without negatively impacting privacy. Hence, reasoning against privacy requires more nuanced arguments than that more privacy always harms security or is not practical.

SAMENVATTING

Digitale technologie speelt een steeds grotere rol in ons dagelijks leven. Vaak worden grote hoeveelheden persoonlijke data gebruikt voor deze technologieën. Overheden en bedrijven verzamelen meer en meer data over ons om betere diensten te kunnen leveren, misdaad en terrorisme te bestrijden, en fraude te voorkomen. Het resultaat: we hebben minder privacy dan ooit.

Toch is privacy belangrijk. Een gebrek aan privacy schaadt individuen. Het beperkt de persoonlijke ontwikkeling van mensen en hun vrijheid om hun privéleven naar eigen inzicht in te richten. Door een gebrek aan privacy veranderen mensen hun gedrag, niet omdat zij iets illegaals doen, maar uit angst over hoe dit gedrag door anderen geïnterpreteerd zou kunnen worden. Van een gebrek aan privacy gaat dus een verlammend effect uit. Bovendien kan het verzamelen en aggregeren van persoonlijke data onder andere leiden tot uitsluiting en het trekken van onjuiste conclusies over individuen. Ook bestaat het gevaar dat persoonlijke data verder verspreid worden dan verwacht.

Een gebrek aan privacy schaadt echter niet alleen individuen, maar berokkent ook schade aan de samenleving als geheel. Zonder de beschermende werking van privacy is het veel moeilijker voor burgers om een kritische blik te ontwikkelen, terwijl deze juist essentieel is voor het goed functioneren van een democratische samenleving.

Hoewel privacy belangrijk is, bestaan er veel argumenten voor het niet beter beschermen van privacy. In dit proefschrift richten we ons op twee veelgebruikte argumenten tegen (het beter beschermen van) privacy, en laten we zien dat de werkelijkheid genuanceerder is.

Het eerste argument is dat het beter beschermen van privacy resulteert in een vermindering van veiligheid. In hoofdstuk 3 herintroduceren we het concept van *revocable privacy* om te laten zien dat het mogelijk is om systemen te bouwen die tegelijkertijd privacy en veiligheid bieden. Zolang gebruikers zich houden aan de regels van het revocable privacy systeem zijn ze volledig anoniem. Alleen als ze deze regels schenden, kan hun anonimiteit ingeperkt worden. Om het nut van deze aanpak te illustreren benoemt hoofdstuk 3 een aantal scenario's die baat hebben bij revocable privacy. Daarnaast geeft dit hoofdstuk aan welke bestaande systemen reeds revocable privacy bieden.

Een veelvoorkomende reden voor online platformen om anoniem gebruik niet toe te staan is het risico op misbruik. In hoofdstuk 4 introduceren we het revocable privacy systeem vote-to-link. Dit systeem

biedt Wikipedia, en andere online platformen, de mogelijkheid om anonieme bewerkingen (of acties van gebruikers in het algemeen) toe te staan, terwijl het tegelijkertijd het platform in staat stelt te herstellen van misbruik door anonieme gebruikers. Standaard zijn de acties van een gebruiker onkoppelbaar. Echter, wanneer moderatoren menen dat een actie onwenselijk is, kunnen zij hun stem uitbrengen op deze actie. Wanneer een actie voldoende stemmen heeft verzameld, kan het systeem alle andere acties van dezelfde gebruiker aan elkaar koppelen (binnen een vooraf vastgelegd tijdsbestek). Op deze manier kunnen de andere potentieel onwenselijke acties van diezelfde gebruiker onderscheiden worden van de overige acties, en daarmee sneller onderzocht of verwijderd worden.

Het vote-to-link systeem implementeert de regel dat een actie niet door te veel moderatoren als onwenselijk mag worden gemarkeerd. Het gedistribueerde versleutelingsschema (Engels: distributed encryption scheme) uit hoofdstuk 5 implementeert de regel dat een partij niet op te veel verschillende locaties gebeurtenissen mag veroorzaken. Het schema lost het zeilsnijdersprobleem op—zeilsnijders zijn criminelen die vrachtwagens die geparkeerd staan langs snelwegen beroven door hun zeil open te snijden. Het schema doet dat door op een privacyvriendelijke manier alleen die auto's te identificeren die op veel parkeerplaatsen stoppen. We vereenvoudigen Hoepman en Galindo's originele gedistribueerde versleutelingsschema, voegen echte sleutelrotatie toe—dit is essentieel voor veel toepassingen—en stellen een gegroepeerde aanpak voor die veel efficiënter is voor kleine berichtruimtes zoals die van kentekenplaten.

Het tweede argument tegen het beter beschermen van privacy dat we onder de loep nemen in dit proefschrift is dat privacyvriendelijke oplossingen niet praktisch zouden zijn. We laten echter zien dat zowel het vote-to-link systeem als het gedistribueerde versleutelingsschema efficiënt genoeg zijn in de praktijk.

Attribuutgebaseerde credentials (ABCs) zijn digitale alternatieven voor identiteitsdocumenten, klantenkaarten, etc. ABCs bieden sterke privacygaranties. Echter, om de veiligheid van het ABC systeem te beschermen moeten credentials ingetrokken kunnen worden wanneer de credentialdrager verloren geraakt, gestolen, of misbruikt is. In hoofdstuk 6 introduceren we het eerste privacyvriendelijke intrekschema voor ABCs dat snel genoeg is, zelfs wanneer deze op een chipkaart geplaatst worden.

Private information retrieval (PIR) maakt het mogelijk voor clients om een rij uit een database op te halen, zonder dat de databaseserver erachter komt welke rij wordt opgehaald. Het bewerkstelligen van deze sterke privacy-eigenschap vergt veel rekenkracht van de databaseserver.

In hoofdstuk 7 laten we zien hoe we verzoeken van vele clients kunnen groeperen om zodoende de rekendruk op de databaseservers te verminderen. Dit nieuwe schema is efficiënt genoeg om PIR toe te passen op certificate transparency, een systeem dat ontwikkeld is om afwijkend gedrag van aanbieders van webcertificaten te detecteren. Zo wordt certificate transparency privacyvriendelijk.

Deze resultaten tonen aan dat de twee veelgebruikte argumenten tegen een betere bescherming van privacy die we behandelen in dit proefschrift niet algemeen waar zijn. Sterker nog, we laten zien dat we praktische systemen kunnen bouwen die zowel veiligheid als privacy bieden, en dus veiligheid bewerkstelligen zonder privacy te verminderen. Kortom, argumenten tegen privacy vereisen meer nuance. Het beter beschermen van privacy gaat niet noodzakelijk ten koste van onze veiligheid en praktische overwegingen zijn geen belemmering.

CONTENTS

1

INTRODUCTION

> Time: January 15, 2014, 14.00h ETZ
> Observations: Subject is searching for information on Tor, has downloaded Tor, and is looking up how Tor hidden services work and what level of protection they provide. Subject is also visiting anonymous whistle blowing systems of major US and Dutch newspapers. Many of them are backed by SecureDrop. (...) Subject is now reading documentation about and analyses of SecureDrop.
> Analysis: What is subject hiding? Does he want to visit the dark web? What reason could there be to do so? Does the subject possess classified information that he wants to leak?

The subject? Me. Did I really look into all these topics? Yes. Was my behavior really observed? Unknown. I do know that I felt self-conscious researching these topics.[1] In fact, I wondered if I would have been better off using Tor to do this research in the first place, or at least should have used an academic rather than domestic internet connection. I did not feel self-concious because I did not have a good reason to make these queries, and not because I felt that *I* was doing anything wrong, but because of how my behavior could be viewed from the outside. I was concerned that I'd be called to explain my interest in this gray area of the internet.

1.1 THE IMPORTANCE OF PRIVACY

The above experience illustrates the harmful chilling effects of (governmental) surveillance—it has the potential to change how people behave. Yet, privacy, and by extent the people it affects, is not harmed by surveillance alone. For example, governments can also harm me by collecting fragmented, possibly outdated information about me from my online behavior and then using these data to deny me access to the country or to other services (particularly, if this decision is made by an opaque machine learning algorithm operating on these data).

1 At the time we were researching whether the Dutch equivalent to crime Crime Stoppers, Stichting M, could safely offer anonymous reporting via the Internet rather than via the telephone system. This research resulted in a report for the WODC (Dutch: Wetenschappelijk Onderzoek- en Documentatiecentrum, English: Research and Documentation Centre) an independent organization within the Dutch ministry of Safety and Justice [85]. Later, a journal paper [84] summarizing the report was published.

Privacy is also not just affected by governments. I can equally easily be harmed by companies collecting data about me, be it from public social media pages or from more private browsing habits. Such harm occurs, for example, when such a profile about me is later used to determine whether I am eligible for a loan, or whether I can obtain an insurance product only at a higher price.

These scenarios exemplify several types of privacy harms. In his privacy taxonomy, Solove identifies many more types of harms [151] (including many interesting examples). Besides harms resulting from the collection of information, such as surveillance as in the example at the start of this chapter, harm may also result from information processing, such as exclusion based on information collected and aggregation of data to make it more valuable as in the two examples above. Another class of privacy harms is caused by dissemination of data, for example, because information is spread more widely than you expected or because the information that is disseminated is not even correct.

The harms resulting from a lack of privacy are not always universally accepted. For example, the chilling effect of surveillance has long been called into question (it is very difficult to create a control group). However, recent research has confirmed its existence. To study the chilling effect of mass surveillance, Marthews and Tucker [118] as well as Penney [137] compared the online behavior of internet users before and after the Snowden revelations in June 2013. Marthews and Tucker showed a significant change in search behavior: after the Snowden revelations, users were less likely to google for topics that could 'get them in trouble' with the government. Similarly, Penney showed that users significantly less frequently looked up Wikipedia articles related to terrorism after the revelations.

A common argument against a call for more privacy (in particular in relation to government surveillance) is that 'I have nothing to hide, therefore I have nothing to fear'. The leading example of this chapter shows why it is flawed. The argument presumes that you only have something to hide if you do something wrong. Yet, I felt like hiding my behavior, not because I was doing something wrong, but because of how it could be construed. See Solove [152] for a more comprehensive discussion of the issues with the nothing-to-hide argument.

So far, we have focussed on how (a lack of) privacy harms individuals. However, it also affects society as a whole. As Solove paraphrases John Dewey: "the value of protecting individual rights emerges from their contribution to society" [152]. For a democratic society to function, journalists, researchers, activists, writers and others should be free to explore, report, and criticise. However, as Cohen argues [49], a lack of privacy has even further reaching consequences. Privacy is a prerequi-

site for a critical mindset—which is essential in a democratic society—to develop in the first place. Without privacy, a person's developing mindset may succumb to the pressure of companies and governments to become "fixed, transparent, and predictable" [49]. Therefore, privacy should not only be protected because of the harm that is otherwise inflicted upon individuals, it should even more so be protected to ensure the functioning of a democratic society as a whole.

So, we should protect privacy. However, what do we mean by privacy? In general, privacy is difficult to define [97, 151]. We briefly summarize Koops et al. [97] to show the pluriformity of privacy. In an effort to conceptualize privacy, Koops et al. identified two groups of privacy types ("things that can be 'watched' or intruded upon" [97]). The first group is captured by the notion of 'being let alone'. It encompasses the right to control access to one's body (bodily privacy); to control access to one's private space, such as a home (spatial privacy); to freedom from interference in communication (communicational privacy); and to freedom from interference with property used in public (proprietary privacy). The second group is captured by the notion of 'self-development'. It encompasses the freedom to develop one's interests and beliefs (intellectual privacy); to make intimate decisions (decisional privacy); to be free to associate with others (associational privacy), and the freedom to be one's self even in public (behavioral privacy).

As the scenarios described above show, data *about* people has the potential to affect many of these types. That is why Koops et al. consider the informational privacy as an overarching aspect that "does not protect the body, space communications, behaviors, etc., directly, but protects the information about these" [97]. It is this digital type of privacy that we seek to protect in this thesis.

The new general data protection regulation (GDPR) aims to protect the personal data of EU citizens [135]. It describes several principles by which personal data should be protected. For example, the GDPR requires that data is processed 'lawfully, fairly and in a transparent manner', that the data are only used for the purpose for which they have been collected ('purpose limitation'), and that they are only stored for as long as this purpose requires ('storage limitation'). On the technical side, the GDPR requires that the integrity and confidentiality of data is guaranteed and that only the minimal amount of data required for the processing should be stored. In this thesis we focus on data minimization as the primary means to protect personal data, and thereby digital privacy. The reason for this focus is that cryptography has the potential to drastically minimize the amount of (identifiable) data stored, without impacting the usefulness of that data.

1.2 RESEARCH QUESTION

It seems then, that, if privacy is so important, we should aim to maximize it. However, there are also many arguments against privacy. Companies frequently claim to need your personal data to offer better services. Governments, too, claim to need personal data to offer better services as well as to combat fraud, identity theft, crime, and terrorism.

Underlying these arguments is a subtle balancing act. The argument is usually not that privacy is not important. No, privacy, when compared to other goals, is simply found to be less important. In this thesis, I focus on two such comparisons:

1. more privacy means less security, so, security being more important than privacy, we cannot increase privacy; and

2. we cannot use privacy friendly solutions as they are too inefficient (or not possible at all) compared to non privacy-friendly solutions.

This thesis will show that the situation is more nuanced and that these comparisons are not generally true.

When balancing security and privacy, there is a tendency to rely solely on procedural and legal measures to protect (personal) data and thereby privacy. Instead in this thesis, we aim to provide rigid cryptographic protections to ensure that these data cannot be abused.[2] In particular, in this thesis we answer the following main research question.

> Can we use cryptography to construct new practical systems that offer security and privacy simultaneously, without having to rely on procedural or legal measures?

To answer this question, we consider three subquestions:

1. What do we mean by systems that offer security and privacy simultaneously, but do not primarily rely on procedural or legal measures?

2. Can we build new systems that use cryptography to offer security and privacy simultaneously?

3. Can we make our systems offering both security and privacy efficient enough to be used in practice?

To show how this thesis answers these subquestions, we revisit the two leading comparisons.

2 Simply encrypting data is not the answer. While encrypting data protects against some attacks by outsiders, internally, the data can simply be decrypted and then misused.

1.2.1 *Privacy versus security*

There seem to be limits to how much privacy we can enjoy. One way for people to obtain privacy online is to ensure that they are anonymous. Yet, in a real-world society where everyone is unconditionally anonymous, nobody can be held accountable for their behavior, and the security of the society suffers (for example, it would be impossible to hold criminals accountable for their actions because they are indistinguishable from other citizens). This might be as undesirable as having no privacy. In fact, this lack of accountability is reason for David Davenport to argue that we should not seek anonymous communication on the internet, because the price of anonymity would be too high [53].

Davenport's argument implies a trade-off between security and privacy. He is not the only one to make this argument. For example, the current head of the Dutch intelligence agency AIVD recently argued that "you should wonder how much security your privacy is worth to you" [120].[3]

NSA security consultant Ed Giorio phrased the trade-off between security and privacy in even stricter terms in an interview with The New Yorker. He states that "privacy and security are a zero-sum game" [163]. A zero-sum game is a game in which the gain of one party is compensated by an equal loss of the other party. Or, applied to security and privacy: any gain in privacy results in an equal loss of security. Many have argued that this is a false trade-off [82, 140, 142, 143]. Indeed, it is easy to give examples where increased security comes at no cost to privacy: to protect your bike against theft you secure it with a good lock (rather than putting it under round-the-clock surveillance) and to protect bank notes against forgeries we use strong anti-counterfeit measures (rather than tracing all bank notes). In fact, by increasing your privacy, you may even increase your personal security. For example, you can protect yourself against identity fraud by decreasing the amount of personal information you disseminate.

The rhetoric of security versus privacy is compelling: of course people want more security—surely we all want to stop terrorists, pedophiles and tax evaders—and people are often willing to give up privacy to achieve this. But this is precisely why the security versus privacy rhetoric is dangerous. If you accept the premise that increased security comes at a cost to privacy, you will not look for equally effective solutions that have no (or a reduced) privacy impact. Yet, the preceding examples demonstrate: we *can* gain security without losing any privacy in the real world.

3 Translated from Dutch: "Dan moet je je dus afvragen hoeveel veiligheid is je privacy je eigenlijk waard?"

In the digital realm it is equally possible to have security without impacting privacy. Consider the example of anonymous digital cash. At first sight, the existence of such a system might seem impossible: how can we have digital cash, which is therefore trivial to duplicate, that can be spend anonymously, so that double spenders cannot be identified? Already in 1988, Chaum, Fiat and Naor proposed an anonymous cash system that found a way out of this conundrum [44]: you are anonymous as long as you spend your digital coins once. If, however, you double spend, the system can identify you and hold you accountable. While in this system not everyone is unconditionally anonymous anymore, you do have a guaranteed level of anonymity as long as you do not misbehave, i.e., double spend.

Hoepman rekindled the term *revocable privacy*—the term was first used by Stadler in his PhD thesis [155]—to emphasize the fact that it is often possible to find digital solutions that have security and privacy simultaneously [82]. The notion of revocable privacy embodies the same 'anonymous until you misbehave' concept as was already present in Chaum et al.'s work. In this thesis—see Chapter 3 for a more thorough discussion—we use the following definition for revocable privacy, where the notion of a *rule* serves to define desired or good behavior:

> A system implements *revocable privacy* if the architecture of the system guarantees a predefined level of anonymity for a participant as long as she does not violate a predefined rule.

This definition demands that a user's privacy is protected through architectural means, that is, the way the system is constructed using cryptography, rather than through procedures or regulations. In other words, if you do not violate the predefined rule, there is no way your anonymity can be reduced (except by breaking the underlying cryptographic assumptions). As Kapor argues, the architecture "more than the regulations which govern its use, significantly determines what people can and cannot do" [91]. Regulations are easy to change, even retroactively, can be sidestepped, or simply ignored. By encoding the rules into the architecture of the system, we make them much more resilient.

A system that uses a trusted third party to enforce a rule, as suggested, for example, by Micali [119] and by Stadler in his earlier work on revocable privacy [155], does not offer the required level of architectural protection, and thus does not satisfy this notion of revocable privacy. (It is easy to see why: the trusted third party can always change the rule.) If trusted parties are used, the rule should explicitly specify their role in the system. This approach to revocable privacy therefore also forces transparency: all ways in which the user can (potentially) lose anonymity need to be captured by the rules of the system.

The notion of revocable privacy, and the systems that implement it, clearly refute the claim that to have security, we need to give up privacy. It also suggests an alternative to Davenport's plea for no anonymous communication: rather than having no anonymity at all, a revocable privacy system can be used to hold those that misbehave accountable without impacting the anonymity of others.

Chapter 3 revisits Hoepman's research on revocable privacy, and more thoroughly discusses what we mean by revocable privacy. That chapter thereby also answers the first subquestion "*What do we mean by systems that offer security and privacy simultaneously, but do not primarily rely on procedural or legal measures?*" Chapter 3 also explores potential use cases that could benefit from revocable privacy. While this overview is by no means exhaustive, it does provide a starting point for our further research. Finally, this chapter briefly recalls some existing systems and constructions that already solve some of these use cases.

To answer the second subquestion, "*Can we build new systems that use cryptography to offer security and privacy simultaneously?*", Chapters 4 and 5 respectively demonstrate two specific revocable-privacy systems: vote-to-link and forward-secure distributed encryption.

The vote-to-link system helps online platforms such as Wikipedia recover from malicious anonymous users. While allowing anonymous editors would have clear benefits for Wikipedia, such anonymity also enables abusive behavior that is hard to recover from.[4] Whereas existing systems can prevent anonymous users that misbehaved from misbehaving again, the vote-to-link system helps to quickly identify past actions of misbehaving users. To do so, it implements the following rule: fewer than k moderators should mark the user's edits as malicious. If a user violates this rule, the system can link all actions of that user within a limited time frame. This eases recovery from abuse by highlighting the other possibly malicious actions performed by the user within that time frame. On the other hand, if a user's edits are never marked by at least k moderators, all its edits remain fully unconditionally anonymous and thus unlinkable.

The vote-to-link system from Chapter 4 uses trusted parties, the moderators, to decide if users behave maliciously, because misbehavior is difficult to codify. Yet, we claim that the vote-to-link system does implement revocable privacy. The crucial distinction is in how we phrase the rule: rather than requiring that users do not behave maliciously (and letting the trusted third parties enforce this rule), we require that fewer than k moderators mark the user's behavior as malicious. Hence, the

4 Since edits are fully anonymous, i.e., unlinkable, it is not possible to determine who made them, even worse, it is not even possible to determine which edits were made by the same user. So, even if some malicious edits are identified it is not possible to determine which other edits were made by the same malicious user.

role of the trusted parties, the moderators in this case, is captured by the rule.

The forward-secure distributed encryption scheme from Chapter 5 implements the rule that parties should cause suspicious events at fewer than k different locations. The distributed encryption scheme can be used to detect criminals that rob trucks parked at rest stops along a highway. Typically, these criminals will visit more truck stops than regular road users, therefore, picking a suitable threshold k makes it possible to distinguish the criminals from the regular vehicles. To do so, automatic number plate recognition systems create distributed encryption ciphertexts corresponding to each car that visits a rest stop. Only if a user violates the rule can the identity of that users (i.e., the license plate) be recovered from these ciphertexts.

1.2.2 *The efficiency of privacy-enhancing technologies*

The perception that privacy and security are at odds with one another is not the only issue that affects the adoption of privacy-enhancing technologies. Efficiency is another one. Privacy-enhancing technologies typically require more cryptography than their non-private counterparts. (The latter need to ensure only that the security requirements are met.) As a result, using privacy-enhancing technologies might incur a performance penalty.

As an example, consider ABC systems. An ABC contains a collection of attributes describing the user. These attributes can be identifying, such as name, date of birth, and social security number; and non-identifying, such as place of residence, gender, and age. The credential is digitally signed by an issuer to attest to the validity of the attributes contained in the credential.

When using privacy-friendly ABCs, a user can decide which attributes to reveal to a verifier and which ones to hide (the verifier can still determine the validity of the revealed attributes). Privacy-friendly ABCs are often also unlinkable: if a verifier is shown a credential with the same non-identifying attributes twice, the verifier cannot determine whether those credentials belong to the same user or to two different users. In particular, it has no way of recognizing the credential.

The core functionality of ABCs is to allow users to show credentials to verifiers. To make an ABC system useable, these operations should be fast. As part of the 'I Reveal My Attributes" (IRMA) project,[5] we developed a fast, user-friendly, implementation of ABCs based on Idemix [36,

5 See https://www.irmacard.org and https://privacybydesign.foundation/irma/. Recently the IRMA project has shifted focus to a smart phone based implementation because they offer a much better user interface than smart cards.

87]. One of the IRMA team's contributions was to make a full smart card implementation of ABCs that can prove possession of a credential in approximately one second [162].

However, a practical deployment of ABCs also requires that credentials can be revoked when the smart card that holds them is lost, stolen or abused. In essence this is a security measure designed to protect the security of the system as a whole. The question is, how do you revoke a credential that you cannot recognize? (If you could recognize the credential, it would be linkable.)

There already existed many theoretical revocation schemes with excellent privacy protection. Some of them even have a constant time overhead (i.e., independent of the number of revoked credentials). However, in practice they were simply too slow (even the constant-time schemes caused a slowdown of a least a couple of seconds). In Chapter 6 we achieve a truly practical revocation scheme with minimal overhead for smart cards by very slightly loosening the privacy guarantees.

This example again shows the need to think about security and privacy simultaneously. Privacy-friendly ABCs offer good privacy, but they would not be deployed without revocation as a security measure to prevent abuse. Furthermore, it shows the importance of efficiency. Even though all the cryptographic systems existed to build privacy-friendly ABCs with revocation, they were too slow to practically deploy. So, not only is it important to design privacy-enhancing technologies, it is equally important to analyse and, if necessary, improve their efficiency.

To answer the third subquestion, "*Can we make our systems offering both security and privacy efficient enough to be used in practice?*", Chapters 4 and 5 contain experimental results to show that the vote-to-link systems and the distributed encryption schemes are practical.

Starting with Chapter 6 we change tack and no longer focus on the type of conditional anonymity offered by revocable privacy, but seek to design efficient systems that offer both security and privacy. The first of these is the revocation scheme in Chapter 6 we discussed above. The second is a more efficient private information retrieval method in Chapter 7. We show how we can answer queries from clients more efficiently by batching them, rather than answering them one by one. While private information retrieval by itself is already a very useful construction in general, we also show how certificate transparency (which offers security against rogue certificate authorities) in particular can be made privacy friendly (so that users do not leak the sites they visit) using our batched private information retrieval scheme. The sublinear scaling of our new scheme reduces the performance concerns in this application.

The performance aspects of these four chapters answer the third subquestion.

1.3 ORGANIZATION OF THIS THESIS

The chapters in this thesis are mostly self-contained. However, Chapters 4 to 7 rely on cryptography to achieve their goals. We introduce the relevant cryptographic concepts and notation in Chapter 2.

It is also advisable to read Chapter 3 before reading Chapters 4 and 5 as the latter rely on the notion of revocable privacy, which we introduce in Chapter 3.

1.4 CONTRIBUTION PER CHAPTER

The following list summarizes each chapter, indicates corresponding publications, and highlights my own contributions therein.

CHAPTER 3 revisits the principle of revocable privacy and explores potential applications of revocable privacy. In particular, we consider two ways for the user to interact with the system: interactive and passive. The type of interaction implies different security guarantees. Furthermore, we explore different consequences of violating rules of the system. To apply these ideas, we also explore some real-world use cases that can benefit from the ideas of revocable privacy, and derive the underlying abstract rules. This approach allows us to highlight which abstract rules can already be implemented, for which ones we propose solutions in this thesis, and which remain open.

This chapter is based on the article

> Wouter Lueks, Maarten H. Everts, and Jaap-Henk Hoepman. "Revocable Privacy: Principles, Use Cases, and Technologies". In: APF 2015. LNCS vol. 9484. Springer, 2016, pp. 124–143. DOI: 10.1007/978-3-319-31456-3_7.

This article was jointly written by Maarten Everts, Jaap-Henk Hoepman and myself. Maarten Everts and I conducted the interviews and brainstorm sessions to extract the revocable privacy use cases, and extract the underlying abstract rules. In addition, I improved the original definition of revocable privacy, analyzed the consequences of different types of user interaction and related the rules to existing primitives. Jaap-Henk Hoepman provided many helpful suggestions and improvements.

CHAPTER 4 solves the problem of recovering from abuse by anonymous users. Whereas other systems effect future consequences

of misbehavior, our vote-to-link system helps to recover from earlier, anonymous, misbehavior by linking actions by misbehaving users within a limited time frame. To protect the privacy of the user, such linking is only possible when sufficiently many moderators agree. In addition, we propose a variant that also allows the moderators to vote anonymously.

This chapter is based on the article

> Wouter Lueks, Maarten H. Everts, and Jaap-Henk Hoepman. "Vote to Link: Recovering from Misbehaving Anonymous Users". In: WPES 2016. ACM, 2016, pp. 111–122. DOI: 10.1145/2994620.2994634.

The conception of the idea of the vote-to-link scheme is joint work by the authors. My contribution is the formalization of the vote-to-link scheme, proving its correctness, designing the moderator-anonymous solutions, implementing the schemes to assess their practicality, and writing the paper. Maarten Everts and Jaap-Henk Hoepman provided many helpful comments and suggestions to improve the paper.

CHAPTER 5 presents improved distributed encryption (DE) schemes that can be used to implement threshold rules. More precisely, it allows recovery of a message only if that same message has been encrypted by a sufficient number of different senders. Our solution improves the original scheme by Hoepman and Galindo by relying on weaker trust assumptions and only requiring a regular cyclic group setting, rather than a bilinear group setting. In addition, we present a batched solution that is faster for small message spaces. Furthermore, both schemes support proper (rather than trivial) key-evolution.

This chapter is based on the article

> Wouter Lueks, Jaap-Henk Hoepman, and Klaus Kursawe. "Forward-Secure Distributed Encryption". In: PETS 2014. LNCS vol. 8555. Springer, 2014, pp. 123–142. DOI: 10.1007/978-3-319-08506-7_7.

This article is the culmination of many brainstorm sessions of Jaap-Henk Hoepman and myself. In addition, Klaus Kursawe proposed the batching scheme and sketched a proof of its security. This approach inspired the improved regular DE scheme. I formalized the proof sketch, constructed the forward-secure key-evolution scheme and wrote most of the paper.

CHAPTER 6 presents a fast revocation scheme for attribute-based credentials that is fast enough to be implemented on smart cards,

while being efficient for verifiers too. To achieve this speed-up, we trade a small amount of linkability: user's authentications are linkable within small time frames.

This chapter is based on the journal article

> Wouter Lueks, Gergely Alpár, Jaap-Henk Hoepman, and Pim Vullers. "Fast revocation of attribute-based credentials for both users and verifiers". In: *Computers & Security* 67 (2017), pp. 308–323. ISSN: 0167-4048. DOI: 10.1016/j.cose.2016.11.018.

which is an extended and revised version of the conference article

> Wouter Lueks, Gergely Alpár, Jaap-Henk Hoepman, and Pim Vullers. "Fast Revocation of Attribute-Based Credentials for Both Users and Verifiers". In: IFIP SEC 2015. IFIP AICT vol. 455. Springer, 2015, pp. 463–478. DOI: 10.1007/978-3-319-18467-8_31.

The initial idea of using a single generator per epoch and per verifier is joint work by the authors. Gergely Alpár and I constructed a first rough draft of the paper, that I then refined into the current chapter. In particular, I formalized the scheme, proved its security, proposed extensions, and analyzed the storage requirements. Pim Vullers analyzed the smart card performance, whereas I estimated the verifier performance. Gergely Alpár and Jaap-Henk Hoepman also provided many helpful comments that improved this paper.

CHAPTER 7 proposes a sublinear scaling scheme for multi-client information-theoretic PIR. We decrease the per-query computation time for PIR servers by allowing the servers to batch queries and answer them simultaneously.

This chapter is based on the article

> Wouter Lueks and Ian Goldberg. "Sublinear Scaling for Multi-Client Private Information Retrieval". In: FC 2015. LNCS vol. 8975. Springer, 2015, pp. 168–186. DOI: 10.1007/978-3-662-47854-7_10.

Ian Goldberg proposed the use of Strassen's algorithm to speed up the PIR servers, and he recognized the relation between Strassen's algorithm and batch codes. I analyzed this relation, implemented Strassen's algorithm in Percy++ [74], a C++ library for PIR, and fine-tuned and analyzed its performance. In addition, I examined certificate transparency as a potential use case for our improved batched PIR scheme. The paper is joint work by the authors.

2

PRELIMINARIES

In this chapter we cover preliminaries that are used in the following chapters of this thesis. We refer to Katz and Lindell [93] for a more general introduction to the topics we discuss in Sections 2.2 to 2.5 and of modern cryptography in general.

2.1 NOTATION

We first introduce some notation. We write $\mathbb{Z}_p$ to denote the integers modulo p, and $\mathbb{Z}_p^*$ to denote its units. In this thesis, p is usually a prime, so then $\mathbb{Z}_p$ is a field. We write $a \in_R A$ to indicate that a is chosen uniformly at random from the (finite) set A. We denote the set $\{1, \ldots, n\}$ by $[n]$. The cardinality of a set A is denoted by $|A|$.

We indicate the set of all strings by $\{0,1\}^*$, while the set of all strings of length ℓ is denoted by $\{0,1\}^\ell$. We write $x \parallel y$ to denote the concatenation of the strings x and y. We denote the length of a string $x \in \{0,1\}^*$ in bits by $|x|$. A matrix $\mathbf{A}$ is typeset in bold. We denote the XOR operator, which can be applied (component wise) to bits, strings, vectors and matrices, by $\oplus$.

We write $\mathcal{A}, \mathcal{B}$, etc., for algorithms and adversaries. Since these algorithms are often randomized, we write $x \leftarrow \mathcal{A}(y)$ to denote that x is the random variable denoting the output of the algorithm $\mathcal{A}$ when run on input y. The symbols $\top$ and $\bot$ denote success and failure respectively.

Finally, we write $\lg a$ to denote the logarithm base 2 of a, and $\ln a$ to denote the natural logarithm of a.

2.2 GROUPS AND BILINEAR MAPS

The schemes in this thesis rely on regular cyclic groups and bilinear groups. We introduce these here.

2.2.1 *Cyclic groups*

Almost all schemes in this thesis rely on cyclic groups (the exception is the PIR scheme in Chapter 7 that only relies on fields). We give a short introduction here, and refer to Katz and Lindell [93, Section 8.3] for a more extensive treatment. Typically, we write $\mathbb{G}$ for a cyclic group, and g, with $g \in \mathbb{G}$, for its generator. In this thesis, we write all groups

multiplicatively. Moreover, all groups we consider are of prime order p.

To precisely define the hardness of problems such as the discrete logarithm problem—see Section 2.4—we use a polynomial time algorithm $\mathcal{G}$ that generates a description of a group of a given size. More precisely, $\mathcal{G}$ takes as input a security parameter 1^ℓ and outputs a tuple $(\mathbb{G}, p, g)$ describing a group $\mathbb{G}$ of order p generated by g, such that p is ℓ bits.[1]

In our schemes we often use hash functions that map strings to group elements. Often, such hash functions can indeed be constructed. For example, it is possible to hash to cyclic groups that are constructed using regular elliptic curves over a prime field [23], to the quadratic residues (apply a regular hash function and then square the result to obtain a quadratic residue), and to (subgroups of) $\mathbb{Z}_p^*$ [87].

2.2.2 *Bilinear maps*

Our vote-to-link scheme (see Chapter 4) and Hoepman and Galindo's distributed encryption scheme (see Section 5.3.3) rely on bilinear maps. We summarize the important facts and definitions, and refer to Galbraith [68] and Galbraith et al. [69] for a more extensive treatment. Consider a pair of cyclic groups $(\mathbb{G}_1, \mathbb{G}_2)$, both of prime order p, with generators g and h respectively. We call such a pair a bilinear group pair when there exists a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ mapping these groups into a target group $\mathbb{G}_T$ of prime order p such that

- the map is bilinear, i.e., for all $a, b \in \mathbb{Z}_p$ we have $\hat{e}(g^a, h^b) = \hat{e}(g, h)^{ab}$;
- the map is non-degenerate, i.e., $\hat{e}(g, h)$ is a generator of $\mathbb{G}_T$; and
- the map is efficiently computable.

Unless otherwise specified, we write g, g_0, g_1, etc., for generators of group $\mathbb{G}_1$, and h, h_0, h_1, etc., for generators of group $\mathbb{G}_2$. We frequently call such a bilinear map a *pairing*.

Galbraith et al. identify three different types of pairings [69]:

TYPE 1 In type 1 pairings $\mathbb{G}_1 = \mathbb{G}_2$.

TYPE 2 In type 2 pairings $\mathbb{G}_1 \neq \mathbb{G}_2$, but there exists an efficiently computable homomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$.

TYPE 3 In type 3 pairings $\mathbb{G}_1 \neq \mathbb{G}_2$, and no efficiently computable homomorphisms between the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ exist.

1 The unary encoding 1^ℓ of ℓ ensures that the group generator algorithm $\mathcal{G}$ runs in polynomial time with respect to its inputs. See Section 2.3.

There are also type 4 pairings, but they are less common, so we omit them here. In this thesis we also do not use type 1 pairings, as our schemes require the decisional Diffie-Hellman (DDH) problem (see Definition 2.5 below) to be hard in at least $\mathbb{G}_1$, which is not the case if $\mathbb{G}_1 = \mathbb{G}_2$. In addition, there exist serious attacks on type 1 pairings defined over fields of characteristic 2 or 3 [1, 2, 75], basically rendering them insecure. Finally, even when defined over a large prime field, type 1 pairings are significantly slower than type 3 alternatives [69].

Type 3 pairings are generally more efficient than type 2 pairings and, additionally, allow efficient hashing onto $\mathbb{G}_2$ (hashing onto $\mathbb{G}_1$ is always possible) [69]. However, type 3 pairings are also not free from attacks. Recent improvements by Kim and Barbulescu [96] imply that the discrete logarithm (DL) problem (see Definition 2.4 below) in the common 256 bits Barreto-Naehrig curve is significantly weaker than previously expected. Rather than obtaining 128 bits security, these curves now only offer approximately 96 bits of security.[2]

As for regular cyclic groups, we define a generation algorithm $\mathcal{IG}$ that takes as input a security parameter 1^ℓ and outputs a tuple $(\hat{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, g_T)$ describing the bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, the bilinear groups $\mathbb{G}_1, \mathbb{G}_2$ and the target group $\mathbb{G}_T$ generated by g, h and g_T respectively, and the group order p (which is ℓ bits). Another reason not to use type 1 pairings is that such a polynomial time generator typically does not exist for these schemes.

2.3 MODELLING SECURITY AND ADVERSARIES

In this thesis we prove the security (or privacy) of our schemes by first defining a game that captures the notion of security (or privacy) that we wish to achieve, and then showing that no realistic adversary can win this game. We refer to Katz and Lindell [93, Section 3.1.2] for a more extensive introduction. Realistic in this case means: adversaries run in probabilistic polynomial time. Our schemes are not secure against unbounded adversaries (with the notable exception of the information-theoretic private information scheme that we extend in Chapter 7).

DEFINITION 2.1 An algorithm $\mathcal{A}$ is called *polynomial time* if there exists a polynomial p such that $\mathcal{A}$ running on any input $x \in \{0,1\}^*$ terminates within $p(|x|)$ steps.[3]

A probabilistic polynomial time algorithm is, additionally, allowed to make coin tosses with a fair coin, and to use this randomness in

2 https://ellipticnews.wordpress.com/2016/09/02/crypto-and-ches-2016-santa-barbara-ca-usa/, last accessed February 17, 2017

3 A step is a basic, constant time, operation of the machine running the algorithm.

its computation. In fact, for cryptography this randomness is essential. Without it, private keys, for example, could not be generated. An alternative formulation, and the one we use here, is that the algorithm has access to an extra input r, traditionally called the *random tape* because these algorithms are formally modelled as Turing machines, containing uniformly random bits. This input should be long enough. If the algorithm runs in $p(n)$ steps for an input of length n, then a random tape $r \in_R \{0,1\}^{p(n)}$ is surely long enough.

DEFINITION 2.2 An algorithm $\mathcal{A}$ is called *probabilistic polynomial time* (PPT) if the algorithm $\mathcal{A}'(\cdot) = \mathcal{A}(\cdot, r)$ is polynomial time where r is a string of uniformly random bits of sufficient length. (Note that $\mathcal{A}'$ needs to run in polynomial time with respect to the length of its input, independent of r.)

We cannot, usually, achieve perfect security. Often, there is a very small chance that an adversary simply breaks the scheme as a result of a lucky guess, for example, by guessing the private key of a party. To allow for these events, we still call a scheme secure, as long as these 'lucky' events occur only with extremely small probability. We formalize this notion of being extremely small using the notion of negligibility.

DEFINITION 2.3 A function f from the natural numbers to the non-negative real numbers is called *negligible* if for every positive polynomial p there exists an N such that $f(n) < p(n)^{-1}$ for all $n > N$. We write $\mathsf{negl}(n)$ to denote an arbitrary negligible function.

This definition is asymptotic. So, while the function $100 \cdot 1.01^{-n}$ is negligible and will eventually be smaller than any inverse polynomial function, it is certainly not small for reasonable sizes of n.

2.3.1 *Random oracle model*

The proofs of our schemes are in the random oracle model. In the random oracle model for a hash function $H : \{0,1\}^* \to Y$, the adversary is given access to a random oracle that it can query on an input x whenever it wishes to evaluate the hash function $H(x)$. This oracle models a random function. More precisely, on a new input x the oracle picks a random $y \in_R Y$ and returns it, while on a repeated input x it repeats the same output y as it gave earlier.

In the regular world, though, we instantiate the scheme by replacing the random oracle with a regular hash function. Hence, a proof of security in the random oracle model is only a heuristic argument that the scheme as composed with the real hash function is also secure.

Despite evidence that this random oracle model is not perfect—it is hard to formalize what it means for a hash function to be a good approximation of a random oracle, and there exist (contrived) examples in which this heuristic fails altogether—we prefer to use the random oracle model because it yields schemes that are generally more elegant and efficient, while the security proof still provides a reasonable guarantee of security and privacy. We refer to Katz and Lindell [93, Section 5.5] for a more thorough discussion.

In our security reductions, we control the random oracle that is provided to the adversary. In particular, we can replace the random output $y \in_R Y$ with specifically crafted outputs for some of the inputs x. For example, rather than outputting a random element $y \in_R \mathbb{G}$ from the group $\mathbb{G}$ of order p, we could return an element g^α for some known randomizer $\alpha \in_R \mathbb{Z}_p$. Such changes cannot be detected by the adversary provided two conditions are met:

1. The adversary did not query the input x before (as the random oracle needs to provide consistent answers).
2. The distribution of the special outputs is still indistinguishable from uniform. This is the case in the example: g^α is uniformly distributed over $\mathbb{G}$.

While the random oracle model also allows us to learn the inputs to the random oracle, we do not use this feature in the proofs in this thesis.

2.4 CRYPTOGRAPHIC ASSUMPTIONS

The security of the schemes in this thesis depends on a number of cryptographic assumptions. These assumptions are typically of the form: "the probability that an adversary succeeds is negligible". To allow for this asymptotic definition of security, we parametrise our schemes with a security parameter ℓ. This parameter dictates the difficulty of the scheme. It typically determines the length of keys, and the sizes of groups. For example, the assumptions we define here depend on the group generation algorithms $\mathcal{G}(1^\ell)$ and $\mathcal{IG}(1^\ell)$ from, respectively, Sections 2.2.1 and 2.2.2. To allow these algorithms to run in polynomial time with respect to this security parameter, we pass the security parameter to these algorithms in its unary encoding 1^ℓ (i.e., the string of ℓ consecutive 1-bits).

We now introduce the complexity assumptions we use in this thesis. In the security proofs in the remainder of this thesis we use these to show that, provided these assumptions hold, the adversary also has a negligible probability of breaking the security or privacy of our new schemes.

DEFINITION 2.4 The *discrete logarithm (DL) problem* in a cyclic group $\mathbb{G}$ is defined as follows. On input of a tuple $(g, X) \in \mathbb{G}^2$, output x such that $X = g^x$. We say the *DL assumption holds relative to* $\mathcal{G}$ if for all PPT algorithms $\mathcal{A}$ there exists a negligible function negl such that

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

where the probability is over the output $(\mathbb{G}, g, p) \leftarrow \mathcal{G}(1^\ell)$, the choice of $X \in_R \mathbb{G}$ and the random bits of $\mathcal{A}$.

DEFINITION 2.5 The *decisional Diffie-Hellman (DDH) problem* in a cyclic group $\mathbb{G}$ is defined as follows. On input of a tuple $(g, X = g^x, Y = g^y, Z = g^z) \in \mathbb{G}^4$, output 1 if $z = xy$ and 0 otherwise. We say the *DDH assumption holds relative to* $\mathcal{G}$ if for all PPT algorithms $\mathcal{A}$ there exists a negligible function negl such that

$$\Big|\Pr[\mathcal{A}((\mathbb{G}, p, g), g^x, g^y, g^{xy}) = 1)] - \Pr[\mathcal{A}((\mathbb{G}, p, g), g^x, g^y, g^z) = 1)]\Big| \leq \mathsf{negl}(\ell),$$

where the probability is over the output $(\mathbb{G}, g, p) \leftarrow \mathcal{G}(1^\ell)$, the random choices of $x, y, z \in_R \mathbb{Z}_p$ and the random bits of $\mathcal{A}$.

The hardness of the DL and DDH problems in the bilinear groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ is defined by the obvious extension of these definitions where the algorithm $\mathcal{A}$ is now also supplied with the description of the other groups and the bilinear map. More precisely, we say that the assumption holds in $\mathbb{G}_i$ with respect to the bilinear group generation algorithm $\mathcal{IG}$, if in the above definitions algorithm $\mathcal{IG}$ replaces $\mathcal{G}$, $g \in \mathbb{G}_i$, and $\mathcal{A}$ is supplied with the full output $(\hat{e}(\cdot, \cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, g_T)$ of $\mathcal{IG}$ instead of the output $(\mathbb{G}, p, g)$ of $\mathcal{G}$.

The Bilinear Diffie-Hellman problem in a type 1 pairing setting—in this case $\mathbb{G}_1 = \mathbb{G}_2$ and $\mathbb{G}_1$ and $\mathbb{G}_2$ are generated by g—asks to compute $\hat{e}(g, g)^{abc}$ given (g, g^a, g^b, g^c) [90]. In this thesis we require a decision version of this problem [22], but in the type 3 setting rather than the type 1 setting. There exist different versions of this assumption in the literature [42]. These versions differ in whether elements with the exponents a, b, and c are given in just one group ($\mathbb{G}_1$ or $\mathbb{G}_2$) or both. A common variant is the following type 3 version.

DEFINITION 2.6 The *decisional bilinear Diffie-Hellman (DBDH-3) problem* in a type 3 bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follows [41]. On input of a tuple $(g, A_1 = g^a, B_1 = g^b, C_1 = g^c, h, A_2 = h^a, C_2 = h^c, Z = g_T^z) \in \mathbb{G}_1^4 \times \mathbb{G}_2^3 \times \mathbb{G}_T$ output 1 if $z = abc$ and 0 otherwise. We

say the *DBDH-3 assumption holds relative to $\mathcal{IG}$* if for all PPT algorithms $\mathcal{A}$ there exists a negligible function negl such that

$$\Big|\Pr[\mathcal{A}(\gamma, g^a, g^b, g^c, h^a, h^c, g_T^{abc}) = 1] - \Pr[\mathcal{A}(\gamma, g^a, g^b, g^c, h^a, h^c, g_T^z) = 1]\Big| \leq \mathsf{negl}(\ell),$$

where the probability is over the output $\gamma = (\hat{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, g_T) \leftarrow \mathcal{IG}$, the random choices of $a, b, c, z \in_R \mathbb{Z}_p$ and the random bits of $\mathcal{A}$.

In fact, in the proof of our vote-to-link scheme we can do with a slightly weaker variant, DBDH-3b, where we omit $C_1 = g^c$ from the problem instance.

DEFINITION 2.7 The *decisional bilinear Diffie-Hellman (DBDH-3b) problem* in a type 3 bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follows [41]. On input of a tuple $(g, A_1 = g^a, B_1 = g^b, h, A_2 = h^a, C_2 = h^c, Z = g_T^z) \in \mathbb{G}_1^3 \times \mathbb{G}_2^3 \times \mathbb{G}_T$ output 1 if $z = abc$ and 0 otherwise. We say the *DBDH-3b assumption holds relative to $\mathcal{IG}$* if for all PPT algorithms $\mathcal{A}$ there exists a negligible function negl such that

$$\Big|\Pr[\mathcal{A}(\gamma, g^a, g^b, h^a, h^c, g_T^{abc}) = 1] - \Pr[\mathcal{A}(\gamma, g^a, g^b, h^a, h^c, g_T^z) = 1]\Big| \leq \mathsf{negl}(\ell),$$

where the probability is over the output $\gamma = (\hat{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, g_T) \leftarrow \mathcal{IG}$, the random choices of $a, b, c, z \in_R \mathbb{Z}_p$ and the random bits of $\mathcal{A}$.

The following version of the q-SDH problem is from the Journal of Cryptography version of Boneh and Boyen's paper [21]. In particular, it no longer relies on the existence of an efficiently computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$.

DEFINITION 2.8 The *q-Strong Diffie-Hellman (q-SDH) problem* in a bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$, both of prime order p, generated by g and h respectively, is defined as follows [21]. On input of a tuple $(g, g^x, g^{x^2}, \ldots, g^{x^q}, h, h^x) \in \mathbb{G}_1^{q+1} \times \mathbb{G}_2^2$, output a pair $(c, g^{1/(x+c)}) \in \mathbb{Z}_p \times \mathbb{G}_1$ for a freely chosen value $c \in \mathbb{Z}_p \setminus \{-x\}$. We say the *$q$-SDH assumption holds relative to $\mathcal{IG}$* if for all PPT algorithms $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathcal{A}(\gamma, g^x, g^{x^2}, \ldots, g^{x^q}, h^x) = (c, g^{1/(x+c)})] \leq \mathsf{negl}(\ell),$$

where the probability is over the output $\gamma = (\hat{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, g_T) \leftarrow \mathcal{IG}$, the random choice of $x \in \mathbb{Z}_p$, and the random bits of $\mathcal{A}$.

We sometimes write that a problem, for example the DDH problem, is *hard*. By this we mean that the corresponding assumption holds.

2.5 SECRET SHARING

Secret sharing allows a secret to be distributed among many parties, so that only qualified subsets of these parties can recover any information about the secret. A common variant is threshold secret sharing, or k-out-of-n secret sharing, where any k parties (out of a total of n parties) can recover the secret, while any coalition of size strictly less than k obtains no information at all about the secret.

Many of the schemes developed in this thesis rely on some form of threshold structure: in our distributed encryption scheme, a plaintext can only be recovered when it has been encrypted by k senders; in our vote-to-revoke scheme, a user's actions can only be linked when at least k moderators agree; and in the IT-PIR scheme, secret sharing is used to recover the original record from k servers.

SHAMIR'S SECRET SHARING SCHEME. The most common secret sharing scheme, and the one we use in our schemes, is Shamir's secret sharing scheme [146]. To share a secret s in a field $\mathbb{F}$[4] such that any k out of n parties can recover it, first generate $k-1$ coefficients $f_1, \ldots, f_{k-1} \in_R \mathbb{F}$ and construct the *secret-sharing polynomial* $f(X) = s + \sum_{i=1}^{k-1} f_i X^i$. Each party i, with index $i \in \mathbb{F} \setminus \{0\}$ is then given the *secret share* $s_i = (i, f(i))$.

Given k of these secret shares we can recover the underlying secret using Lagrange coefficients.

DEFINITION 2.9 Let $\mathcal{I} \subset \mathbb{F}$ be a set of indices of size k, then the corresponding *Lagrange polynomials* are given by $\lambda_i^{\mathcal{I}}(X) = \prod_{j \in \mathcal{I}, j \neq i} \frac{X-j}{i-j}$. Any polynomial $f(X) \in \mathbb{F}[X]$ of degree $k-1$ can then be interpolated by k points $(i, f(i))$, $i \in \mathcal{I}$, as $f(X) = \sum_{i \in \mathcal{I}} \lambda_i^{\mathcal{I}}(X) f(i)$. With some abuse of notation, we define the *Lagrange coefficients* as $\lambda_i^{\mathcal{I}} = \lambda_i^{\mathcal{I}}(0)$. For a polynomial f as before, we have $f(0) = \sum_{i \in \mathcal{I}} \lambda_i^{\mathcal{I}} f(i)$.

Note that recovering the secret s from the shares is a linear operation.

ADDITIVE SECRET SHARING. We also use a simpler secret sharing structure, called additive secret sharing, where the secret is simply split into a number of random shares such that the sum of these shares equals the secret. The distribution of the shares determines the access

4 In this thesis, we often work with the field $\mathbb{Z}_p$ (p is prime in this case) where p corresponds to the order of a group. However, Shamir's secret sharing can be defined for any field, as we do here, or even for a module, provided some inverses in the underlying ring exist.

structure. While any access structure is possible, we focus on threshold k-out-of-n access structures as for Shamir's secret sharing.

To additively share a secret s with a k-out-of-n access structure, create additive shares $r_A \in_R \mathbb{Z}_p$ for each set $A \subset [n]$ of cardinality $n-(k-1)$ subject to the constraint that

$$s = \sum_{\substack{A \subset [n] \\ |A| = n-(k-1)}} r_A.$$

Give share r_A to party i if $i \in A$. Any coalition of at least k parties jointly knows all the shares (each additive share is *not* known to precisely $k-1$ parties, therefore every share is known by at least one member of the coalition), hence the secret can be recovered by adding the shares. For any coalition with at most $k-1$ parties there is at least one share r_A that none of the members know, and hence this is indeed a proper k-out-of-n secret sharing scheme.

2.5.1 *Distributed generation of secret shares*

While secret sharing schemes typically provide perfect protection against unauthorized recovery of the secret, this protection does not extend to the party—traditionally called the dealer—distributing the secret in the first place. If the dealer cheats, the parties have no guarantee that the threshold is as claimed, that the secret can even be recovered, nor that the dealer does not leak the secret. Yet, schemes based on secret sharing generally assume that no party by itself knows the shared secret nor the complete secret sharing. One solution is to assume that the dealer is trusted. Another is to let the parties themselves jointly generate a new secret sharing, without the use of a trusted dealer. In this section we explore two interactive protocols to achieve this, in the next we discuss a non-interactive protocol.

Pedersen's verifiable secret sharing scheme

We first introduce Pedersen's verifiable secret sharing protocol [136]. This interactive protocol allows n parties to jointly compute a k-out-of-n Shamir secret sharing of a random secret. As long as at least one party behaves honestly the secret is indeed random, while at least k parties are necessary to recover the shared secret.

In Pedersen's verifiable secret sharing protocol, each party creates a random secret-sharing polynomial. The sum of these—although it will never be constructed in one place—is the final secret-sharing polynomial. To prove correctness, each party commits to its polynomial.

These commitments require a group $\mathbb{G}$ in which the DL problem is hard to ensure that the commitments are hiding.[5]

PROTOCOL 2.10 Pedersen's verifiable secret-sharing (vss) protocol [136], denoted by $\mathsf{VSS}(n, k, \mathbb{G}, g, p)$, is an interactive protocol between n parties, numbered 1 through n. The secret is in a field $\mathbb{Z}_p$ and there is a corresponding cyclic group $\mathbb{G}$ of prime order p, generated by g, such that the DL problem in $\mathbb{G}$ is hard. Every party has a public signing key that is known to all the other parties. The parties proceed as follows.

1. Every party i generates a secret $x_i \in \mathbb{Z}_p$ and coefficients $f_{i,1}, \ldots, f_{i,k-1} \in_R \mathbb{Z}_p$ to create a $k-1$ degree secret sharing polynomial $f_i(X) = x_i + f_{i,1}X + \cdots + f_{i,k-1}X^{k-1}$.
2. Party i commits to its polynomial. It does this by calculating $F_{i,0} = g^{x_i}$ and $F_{i,j} = g^{f_{i,j}}$ for $j = 1, \ldots, k-1$, signing these values and publicly broadcasting them.
3. When every party has broadcast its polynomial commitments, every party i sends its secret share for party j, $s_{ij} = f_i(j)$, and a signature on s_{ij} securely to party j for $j = 1, \ldots, n$.
4. Every party i checks if it received the correct secret share by comparing the share against the polynomial commitments. In particular, for every party j, it checks that $g^{s_{ji}} = F_{j,0}F_{j,1}^{i} \cdots F_{j,k-1}^{i^{k-1}}$. If any of these checks fail, party i will broadcast the corresponding signature and secret share to identify the misbehaving party. If any misbehaving parties are identified in this step, the protocol aborts.[6]
5. Finally, party i computes its share $s_i = \sum_{j=1}^{n} s_{ji}$ of the shared secret.

When all the parties start out with a $k-1$ degree secret sharing polynomial f_i then the resulting $k-1$ degree secret sharing polynomial is given by $f = f_1 + \ldots + f_n$. Hence this protocol creates a k-out-of-n secret sharing of $f(0) = \sum_{i=1}^{n} f_i(0) = \sum_{i=1}^{n} x_i$.

Sometimes it is useful to construct a public key corresponding to the shared secret and the secret shares. We can easily extend Pedersen's vss scheme to do so.

5 For the security of this protocol and the following protocol (for additive shares) in isolation, it is sufficient that the DL problem is hard in $\mathbb{G}$. However, when using them inside other protocols, we need to take into account that the commitments are public 'representations' of the secret shares. In practice, our schemes already require that the DDH assumption holds in $\mathbb{G}$ as well, assuring that the commitments cannot be abused.

6 In practice a more fine-grained approach might be warranted, like excluding the shares by the misbehaving parties. However, care is required to ensure that parties cannot be excluded from participating by using this mechanism.

PROTOCOL 2.11 The VSS^+ variant of Pedersen's vss protocol, denoted by $\mathsf{VSS}^+(n, k, \mathbb{G}, g, p)$, proceeds by first running the original $\mathsf{VSS}(n, k, \mathbb{G}, g, p)$ protocol. Then the public key h corresponding to the shared secret $\sum_{i=1}^{n} x_i$ and the public key h_i corresponding to secret shares s_i can be calculated using solely public information

$$h = g^{\sum_{i=1}^{n} x_i} = \prod_{i=1}^{n} g^{x_i} = \prod_{i=1}^{n} F_{i,0},$$
$$h_i = g^{s_i} = g^{\sum_{j=1}^{n} s_{ji}} = \prod_{j=1}^{n} g^{s_{ji}} = \prod_{j=1}^{n} F_{j,0} F_{j,1}^{i} \cdots F_{j,k-1}^{i^{k-1}}.$$

Verifiable secret sharing of additive shares

Similarly to Pedersen's vss scheme, we can also verifiably create an additive secret sharing for a k-out-of-n access structure. The idea is similar: each party creates a random secret sharing, commits to the secret sharing, and then distributes the shares to the relevant parties. The final secret sharing is the sum of these. The combination of multiple parties ensures that the secret is indeed random as long as one participant is honest, while the commitments ensure that the parties distribute the shares properly.[7]

PROTOCOL 2.12 The additive verifiable secret-sharing protocol, which we denote by $\mathsf{AVSS}(n, k, \mathbb{G}, g, p)$, is an interactive protocol between n parties, numbered 1 through n. The secret is in a field $\mathbb{Z}_p$ and there is a corresponding group $\mathbb{G}$, with generator g of the same order, such that the DL problem in $\mathbb{G}$ is hard. Every party has a public signing key that is known to all the other parties. The parties proceed as follows.

1. Every party i generates secret shares $r_{i,A} \in_R \mathbb{Z}_p$ for each $A \subset [n]$ of cardinality $n - (k-1)$. Let $x_i = \sum_{A \subset [n], |A| = n-(k-1)} r_{i,A}$ be this party's random secret.

2. Party i commits to its secret shares. It does this by calculating $F_{i,A} = g^{r_{i,A}}$ for all subsets $A \subset [n], |A| = n - (k-1)$, signing these values and broadcasting them.

3. When every party has broadcast its commitments, every party i sends its secret shares for party j, that is, the shares $r_{i,A}$ such that $j \in A$, and a signature on these, securely to party j for $j = 1, \ldots, n$.

7 Despite repeated searches, I have not yet found a reference that contains this algorithm, or, in fact, any algorithm for the distributed generation of an additive secret sharing.

4. Every party i checks that it received the correct secret shares by checking the shares against the commitments. In particular, for every party j and subset $A \ni i$, it checks that $g^{r_{j,A}} = F_{j,A}$. If any of these checks fail, party i will broadcast the corresponding signature and secret share to identify the misbehaving party. If any misbehaving parties are identified in this step, the protocol aborts.

5. Finally, party i computes its shares $r_A = \sum_{j=1}^{n} r_{j,A}$ for all subsets $A \ni i$, of the shared secret.

When all the parties start out with randomly chosen additive shares, then this protocol creates a k-out-of-n additive secret sharing of the value $\sum_{i=1}^{n} \sum_A r_{i,A}$.

Again, we can extend this protocol to publish the public key corresponding to the shared secret.

PROTOCOL 2.13 The AVSS^+ variant of the AVSS protocol, denoted by $\mathsf{AVSS}^+(n, k, \mathbb{G}, g, p)$, proceeds by first running the original $\mathsf{AVSS}(n, k, \mathbb{G}, g, p)$ protocol. Then the public key h corresponding to the shared secret

$$\sum_{i=1}^{n} \sum_{\substack{A \subset [n] \\ |A| = n-(k-1)}} r_{i,A}$$

can be calculated using solely public information:

$$h = g^{\sum_{i=1}^{n} \sum_A r_{i,A}} = \prod_{i=1}^{n} \prod_A g^{r_{i,A}} = \prod_{i=1}^{n} \prod_A F_{i,A},$$

where A ranges over all subsets of $[n]$ of cardinality $n - (k-1)$.

We prove that this AVSS protocol is a proper verifiable secret-sharing protocol.

THEOREM 2.14 *The additive verifiable secret-sharing protocol is such that as long as one party is honest and the protocol completes, the secret sharing is random, and any subset of k honest parties can recover the same secret.*

Proof. In the protocol, every party acts as a dealer. Hence, if at least one of them is honest, the resulting secret sharing is completely random, i.e., indistinguishable from a sharing generated by a trusted party.

Next, we show that any subset of k honest users recovers the same secret. Since the protocol completed successfully, no honest party aborted in step 4. From this, it follows that all honest parties have the same view of the set of shares $\{r_{i,A}\}_A$ of each party i (as the signed commitments $F_{i,A}$ fix each of these values for all honest users), so, if parties $i_0, i_1 \in A$ then both received the same share $r_{i,A}$. Therefore, any subset k honest parties can combine their shares to recover the same secret. □

2.5.2 *Non-interactively generating pseudorandom secret-sharings*

The downside of Pedersen's vss algorithm is that it requires communication among all the parties. This is undesirable when parties cannot communicate with each other, but also when they are not necessarily online at the same time.

It would be easier if the parties could derive a new Shamir secret sharing from an initial sharing of a secret, without communication with each other. Unfortunately, a Shamir secret sharing itself cannot be used for this. Instead, we create a new Shamir secret sharing without communication using Cramer et al.'s share conversion scheme [52]. This scheme converts an additive secret sharing into a Shamir secret sharing.

To additively share a secret s using a k-out-of-n access structure, we proceed as before and generate a share r_A for each set $A \subset \{1, \ldots, n\}$ of cardinality $n - (k-1)$ such that $s = \sum_A r_A$. The share r_A is given to each party i such that $i \in A$.

The advantage of an additive secret sharing is that when all parties change a share r_A in the same way—for example by hashing it—the result is a new valid secret sharing of a fresh random secret. This is not possible for a Shamir secret sharing.

To convert the additive shares into a Shamir secret sharing, Cramer et al. [52] create a secret sharing polynomial f that party i can evaluate only at point i. Define, for every set $A \subset \{1, \ldots, n\}$ of cardinality $n - (k-1)$, the unique $k-1$ degree polynomial g_A such that:

$$g_A(X) = \begin{cases} 1 & \text{if } X = 0 \\ 0 & \text{if } X \in \{1, \ldots, n\} \setminus A. \end{cases}$$

Then, we construct the secret sharing polynomial $f(X)$ as follows:

$$f(X) = \sum_{\substack{A \subset \{1,\ldots,n\} \\ |A| = n-(k-1)}} r_A \cdot g_A(X).$$

This polynomial is indeed of the desired degree and $f(0) = s$. By construction of the polynomials g_As, party i can evaluate f only at $f(i)$. Furthermore, since $g_A(0) = 1$ we have that $f(0) = s$, as desired.

2.6 ZERO-KNOWLEDGE PROOFS OF KNOWLEDGE

A zero-knowledge proof of knowledge allows a prover to prove validity of a statement, for example that it knows the discrete logarithm x of a public key $A = g^x$, without revealing anything beyond the validity of this statement. In particular, for the example, the prover does not reveal

anything about the value of x beyond that it knows it. These proofs are proofs of knowledge because there exists a knowledge extractor that on input of a successful prover can extract the values of which knowledge is being proven; in the example, the extractor recovers the value x. We refer to, for example, Smart [150, Chapter 21] for a general introduction.

In our schemes we often use zero-knowledge proofs to let untrusted parties prove that they followed the protocol correctly. We use a simplified version of the notation of Camenisch and Stadler [39] to denote these zero-knowledge proofs. More specifically, we write

$$\mathrm{PK}\{(x,y) \ : \ A = g^x \wedge B = g^x h^y\}$$

to denote the interactive zero-knowledge proof of knowledge of the values x and y, such that $A = g^x$ and $B = g^x h^y$. The verifier only learns the validity of this proof and the values of A, B, g, h, but not the values before the colon, x and y, of which knowledge is being proven.

An interactive proof of knowledge as above, can be converted into a non-interactive proof using the Fiat-Shamir heuristic [64]. In this case we write

$$\mathrm{SPK}\{(x,y) \ : \ A = g^x \wedge B = g^x h^y\}(m)$$

to denote the signature proof of knowledge (proving the same relation as above), over the message m.

2.7 ANONYMOUS CREDENTIALS

Anonymous credentials, and attribute-based credentials (ABCs) in particular, are a cryptographic alternative to traditional credentials such as driver's licenses and passports. They allow a user to prove that she has a certain attribute, e.g. membership of a group (or a set of attributes as for ABCs), and sometimes relations and inequalities among attributes, without becoming identifiable or linkable because of this.

While anonymous credentials can sometimes encode only a single boolean attribute, they can often contain multiple attributes. Hence, we prefer the term attribute-based credentials for those that contain multiple attributes.[8] ABCs contain a set of attributes, typically encoded as numbers, that a user can selectively reveal to a verifier. Even when some attributes are hidden, the verifier can still assess the validity of the credential. Again, the identity of the user remains hidden.

There are many methods to construct credentials. One could use an attribute-based credential system such as U-Prove [25] or Idemix [35], or directly construct credentials using an appropriate signature scheme

8 In fact, the ABC4Trust project even uses the term *privacy-enhancing attribute-based credentials* to emphasize that these credentials offer good privacy protection [34].

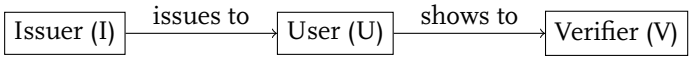


Figure 2.1: Interactions between parties in a credential system.

such as BBS+ signatures [11] (all of these support attributes). If the issuer and the verifier are the same party, you could even use algebraic anonymous credentials [40].

To prevent users from combining credentials from different people, credentials are usually bound to a secret key x. Our schemes are (mostly) agnostic to the choice of credential scheme, so we just write $C(x)$ to denote a credential over the secret key x, and $C(x, a_1, \ldots, a_L)$ to denote a credential over the secret key x and attributes $a_1, \ldots, a_L$.

A typical credential scheme comprises the following three parties, see also Figure 2.1.

ISSUER The issuer issues credentials to users. It ensures that the correct data are stored in the credential, before signing it. A typical credential scheme has multiple issuers.

USER The user holds a set of credentials, obtained from one or more issuers. In attribute-based credential systems, she can disclose a (user defined) selection of attributes from any number of her credentials to a verifier to obtain a service.

VERIFIER The verifier, sometimes called relying party or service provider, checks that the credential is valid, the revealed attributes (if any) are as required, and the credential is not revoked. Based on the outcome, it may provide a service to the user.

The process by which a user obtains a credential is called issuance. While the process of showing possession of a credential is called showing or verification.

Our vote-to-revoke system in Chapter 4 relies heavily on anonymous credentials, because it allows users to use a unique identity and prove properties about it without revealing their particular identities. In Chapter 6 we extend attribute based credential schemes with efficient means of revoking credentials. Both schemes require the following properties:

UNFORGEABILITY It is not possible for any party in the system to forge a credential $C(x)$ over a new key x without the help of the issuer.

UNLINKABILITY The showing of a credential gives no information about the owner to the verifier. In particular, it is not possible for an adversary to distinguish between two users (of its choosing)

when it is shown a credential of one of them. (Provided that the set of disclosed attributes is the same for both users.)

ZERO-KNOWLEDGE PROOFS The showing of a credential $C(x)$ can be combined with a zero-knowledge proof that proves a statement about x.

2.7.1 *An example credential scheme: BBS+ credentials*

A simple credential scheme can be built using BBS+ signatures [11]. We use BBS+ signatures as an example because its setting—it uses a type 3 pairing and relies on a discrete-logarithm based assumption—meshes well with our other schemes that also use cyclic groups and discrete-logarithm based assumptions.

We first describe the signature scheme, and then show how this leads to an anonymous credential scheme. We present a variant of Au et al.'s original BBS+ scheme, due to Camenisch et al. [30], that does not require an efficiently computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$. This allows us to use the more efficient type 3 pairings.

SCHEME 2.15 (BBS+ signatures [11, 30]) The BBS+ signature scheme is given by the algorithms BBS.KeyGen, BBS.Sign, and BBS.Verify.

- $\mathsf{BBS.Setup}(1^\ell)$. On input of a security parameter ℓ this algorithm generates a type 3 bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$, both of prime order p of ℓ bits, generated by g and h respectively. The pairing is given by $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_T$ is generated by $g_T = \hat{e}(g, h)$. It publishes $(\hat{e}(\cdot, \cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, g_T)$.

- $\mathsf{BBS.KeyGen}((\hat{e}(\cdot, \cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, g_T))$. The key generation algorithm takes as input a description of the groups $(\hat{e}(\cdot, \cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, g_T)$ as above. It randomly chooses generators $B_0, \ldots, B_L \in_R \mathbb{G}_1$. Next, the algorithm picks a private key $\gamma \in_R \mathbb{Z}_p$ and computes $w = h^\gamma$. The algorithm publishes a description of the groups (and related functions) and returns private key γ and the public key $\Delta = (w, B_0, \ldots, B_L)$.

- $\mathsf{BBS.Sign}((m_1, \ldots, m_L), \gamma, \Delta)$. To sign a tuple of messages $(m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ using a private key γ and corresponding public key $\Delta = (w, B_0, \ldots, B_L)$, randomly generate $e, s \in_R \mathbb{Z}_p$ and set

$$A = \left(g B_0^s \prod_{i=1}^{L} B_i^{m_i} \right)^{\frac{1}{e+\gamma}} \in \mathbb{G}_1.$$

  The signature σ on $(m_1, \ldots, m_L)$ is then given by (A, e, s).

- BBS.Verify $((m_1, \ldots, m_L), \sigma, \Delta)$. The verification algorithm takes as input a tuple of messages $(m_1, \ldots, m_L) \in \mathbb{Z}_p^L$, a signature $\sigma = (A, e, s)$, and a public key $\Delta = (w, B_0, \ldots, B_L)$. It then checks that
$$\hat{e}(A, wh^e) = \hat{e}(gB_0^s \prod_{i=1}^{L} B_i^{m_i}, h). \tag{2.1}$$
If the equation holds, it returns $\top$ otherwise it returns $\bot$.

Camenisch et al. prove the following theorem.

THEOREM 2.16 (Camenisch et al. [30]) *The BBS+ signature scheme is existentially unforgeable under adaptive chosen message attacks provided the q-SDH assumption holds.*

To use a signature scheme as a credential scheme two things need to change: (1) it should be possible to sign messages without the signer knowing these messages (because in a credential scheme the credential typically contains the user's private key that binds the credential to the user and that key should not become known to the issuer) and (2) it should be possible to prove possession of a signature without being linkable.

The user can run the following BBS.Issue protocol with an issuer to get a signature on attributes $(a_1, \ldots, a_L)$, such that the issuer does not learn the values of the attributes $(a_i)_{i \in \mathcal{H}}$ [11].

- BBS.Issue(). The BBS.Issue protocol is run jointly by a user and a issuer (acting as a signer). Let $(a_1, \ldots, a_L)$ be the user's private input, the message tuple to be signed. The issuer has as input the private key γ. Let $\mathcal{D} \subset [L]$ be the set of indices of attributes determined by the issuer. Then, the attributes with indices in $\mathcal{H} = [L] \setminus \mathcal{D}$ are determined by the user (the issuer does not learn the value of these attributes). First, the user creates a commitment
$$C = B_0^{s'} \prod_{i \in \mathcal{H}} B_i^{a_i}$$
using a random $s' \in \mathbb{Z}_p$, sends it to the issuer, and proves it is well formed:
$$\text{PK}\left\{ ((a_i)_{i \in \mathcal{H}}, s') \ : \ C = B_0^{s'} \prod_{i \in \mathcal{H}} B_i^{a_i} \right\}. \tag{2.2}$$
If the proof verifies, the issuer randomly generates $s'', e \in_R \mathbb{Z}_p$, sets
$$A = \left(gB_0^{s''} C \prod_{i \in \mathcal{D}} B_i^{a_i} \right)^{\frac{1}{e+\gamma}} \in \mathbb{G}_1$$

and returns the transient signature (A, e, s'') to the user. The user calculates $s = s' + s''$ and stores the real signature $\sigma = (A, e, s)$.

It is easy to verify that the resulting signature σ is indeed a correct signature on the attributes $a_1, \ldots, a_L$, and that the issuer does not learn anything about the attributes $(a_i)_{i \in \mathcal{H}}$ due to the information theoretic hiding property of the commitment C. The user can extend the proof of knowledge in equation (2.2) to prove other relations about the attributes (for example that one of the attributes is the private key corresponding to the user's public key).

A user can also show that she has certain attributes by proving possession of a signature over these attributes [11]. She hides the signature σ and hidden attributes $(a_i)_{i \in \mathcal{H}}$ using a zero-knowledge proof. The zero-knowledge proof convinces the verifier that the signature $\sigma = (A, e, s)$ satisfies equation (2.1). This proof is due to Au et al. [11] and is constructed as follows.

Let g_1 and g_2 be two extra generators in $\mathbb{G}_1$, and let $\mathcal{H}$ and $\mathcal{D}$ be the set of hidden and disclosed attributes as before. First, the user creates a commitment $C_1 = A g_2^{r_1}$ to A, where $r_1 \in_R \mathbb{Z}_p$ is a randomizer, and then commits to the randomizer as well using $C_2 = g_1^{r_1} g_2^{r_2}$ where $r_2 \in_R \mathbb{Z}_p$. Then, she sends these commitments to the verifier. These commitments perfectly hide the value of A. Finally, she engages in the following zero-knowledge proof with the verifier:[9]

$$\mathrm{PK}\left\{ (r_1, r_2, \alpha_1, \alpha_2, e, (a_i)_{i \in \mathcal{H}}, s) : C_2 = g_1^{r_1} g_2^{r_2} \wedge C_2^e = g_1^{\alpha_1} g_2^{\alpha_2} \ \wedge \right.$$
$$\hat{e}(C_1, w)\hat{e}(C_1, h)^e = \hat{e}(g, h)\hat{e}(B_0, h)^s$$
$$\left. \prod_{i \in \mathcal{H}} \hat{e}(B_i, h)^{a_i} \prod_{i \in \mathcal{D}} \hat{e}(B_i, h)^{a_i} \hat{e}(g_2, w)^{r_1} \hat{e}(g_2, h)^{\alpha_1} \right\},$$

to prove that she indeed possesses a signature (in the proof $\alpha_1 = e r_1$ and $\alpha_2 = e r_2$).

In the following we will typically just write $\mathrm{PK}\{(a_1, a_2) : C(a_1, a_2)\}$ to denote a proof of knowledge involving a credential C over the attributes a_1 and a_2, even if the credential is actually constructed using BBS+ signatures. If we want to be more explicit, we abuse notation and write

$$\mathrm{PK}\left\{ (A, e, (a_i)_{i \in \mathcal{H}}, s) \ : \ A^{e+\gamma} = g B_0^s \prod_{i \in \mathcal{H}} B_i^{a_i} \prod_{i \in \mathcal{D}} B_i^{a_i} \right\}$$

to denote the proper disclosure proof given above.

9 For clarity we have been slightly liberal in our notation, and did not restrict ourselves to known terms on the left hand side of the equality signs. It is easy to see that all equations can be brought into this form.

3

REVOCABLE PRIVACY: PRINCIPLES AND USE CASES

As we saw in Chapter 1, one way to refute the rhetoric of security versus privacy is to demonstrate systems that have both in equal measures. To this end, Hoepman rekindled the term *revocable privacy* to describe systems that have both security and privacy. The core idea of revocable privacy arises from the realisation that it is not the data themselves that are (or should be) important, but rather the violations of certain rules that manifest themselves in the data. Data related to people who do not violate any rules are irrelevant, and, in fact, these people should remain anonymous, as if no data on their behavior was ever collected. Revocable privacy is a design principle that ensures this property. Informally speaking, a system offers revocable privacy if users of the system are guaranteed to be anonymous except when they violate a predefined rule, or, even more directly: in a system with revocable privacy users are anonymous until they misbehave.

To ensure privacy, the system's anonymity guarantees cannot rely on policy and regulations alone. It is all too easy to ignore policy, to sidestep it, or to change it retroactively. As a result, data that were collected for one specific purpose can easily be reused for another—violating people's privacy. A key aspect of a system implementing revocable privacy is to prevent this type of function creep through technical means: it should not be possible to change the rules retroactively.

In Chapter 1 we already briefly mentioned the example of an anonymous electronic cash system by Chaum, Fiat and Naor [44], which we revisit here. This electronic cash system actually implements revocable privacy (although Chaum et al. did not use this term). Users have electronic coins, which they can spend as if they were physical coins, in effect making an untraceable digital payment system in which the users' privacy is guaranteed. However, to maintain security, this anonymity cannot be unconditional. If it were, it would allow misbehaving users to double-spend the digital coins without consequence. Instead, the revocable privacy aspect of the design guarantees that users are anonymous, as long as they spend the digital coins only once. When they do spend a coin twice, their identity can be recovered from the two transaction records of the two spendings. Any single transaction record, however, gives no information about the identity of the user. This is thus a perfect example of the principle of revocable privacy.

In general, to ensure anonymity for rule-abiding users, data must be collected in a special manner. In Chaum et al.'s electronic cash system, the cut-and-choose paradigm is used to ensure that a single transaction record gives no information, whereas two records for the same coin reveal the identity of the culprit. Distributed encryption [83, 114], see Chapter 5, offers another method for creating threshold based rules. Using distributed encryption, the user's identity is revealed only if she causes an event to happen at sufficiently many *different locations*.

While Chaum et al.'s electronic cash could be seen as such a scheme with a threshold of two, it differs significantly from distributed encryption. In the first, the user actively partakes in the transaction, whereas in the second, the user deliberately does not take part. As a result, these systems have different privacy guarantees and trust assumptions. These different aspects of revocable privacy have not been explored yet.

In all previous work on revocable privacy [82, 83, 110, 114], the focus was on identifying users who violate the rules. However, in some situations, such an approach might be too strong. For example, anonymity is the core property of Tor [59], so it should never be possible to deanonimize users. Yet, Tor can also be abused. In order to stop abuse, some approaches, such as blacklistable anonymous credentials (BLAC), aim to block misbehaving users, rather than to identify them [159].[1]

This chapter's first contribution, in Section 3.1, is to re-examine revocable privacy in a more general setting. We consider the implications of different security models, and explore ramifications of users' actions that are less invasive than simply identifying users, for example, blocking users and linking their actions. In fact, in Chapter 4 we create a vote-to-link system that links a user's actions (within a limited time frame) when she misbehaves.

Next, we explore and classify some use cases for revocable privacy in Section 3.2. We generalize the underlying rules of the use cases into abstract rules. These use cases illustrate that even if a user has violated a rule, she did not necessarily do something wrong. In fact, we will explore some systems where a violation only means that closer examination is necessary.

The abstract rules for the use cases make it possible to link them to specific techniques. Our final contribution, in Section 3.3, is to give a brief non-technical overview of existing techniques that can be used to implement revocable privacy. For each technique, we indicate which abstract rules it can implement. This not only shows which use cases we can already solve, but also highlights which abstract rules we cannot

1 Nymble [160] is a related system that can be used to block misbehaving users. However, it relies on a trusted party that can make users linkable if they misbehave, so we do not consider it further in this chapter.

yet implement. We analyse the latter in Section 3.4 to reveal interesting new research directions. We also discuss some general limitations of revocable privacy. Finally, we conclude this chapter in Section 3.5.

We stress that revocable privacy is not a license for unchecked surveillance. The use cases explored in this chapter come from various sources. Some of them are real, others are purely hypothetical. In many cases the legality and/or morality of the situation described in the use case is debatable. We have included them for the sole purpose of investigating the types of rules a system with revocable privacy might need to implement in the future. Inclusion of a use case in this chapter does not mean that we endorse it in any way.

3.1 REVISITING THE CONCEPT OF REVOCABLE PRIVACY

In this section we will (re)define revocable privacy. We first explore what it means to be anonymous and what levels of anonymity exist.

3.1.1 *Levels of anonymity*

At first sight it may seem that anonymity is an all-or-nothing property: either you have it or you do not. This is false. There are many shades of anonymity, ranging from fully anonymous to fully identified. For example, users might be pseudonymous: their actions are known under a fixed identifier—the pseudonym—but it is not necessarily known which pseudonym belongs to which user.

Some pseudonyms can be easily related to the owner's full identity by at least one party (but not necessarily by everyone). For example, your social security number, bank account number, and passport number, can be related to your full identity by, respectively, the government, the bank and the government again. Other pseudonyms, such as the randomly generated identifier stored in a website's cookie, are often much harder to relate to the owner's full identity, although analysis of transactions related to such pseudonyms generally still allow identification [131]. See, for example, Leenes [102] for a more thorough discussion of different types of identifiability based on pseudonyms.

To improve their anonymity, users can use different pseudonyms with different parties or in different situations. They can even use pseudonyms that change frequently. (We use variants of these temporary pseudonyms in Chapters 4 and 6.) However, it is possible to have even stronger forms of anonymity. When a user's actions are *unlinkable*, it is impossible to determine whether two actions were performed by the same user or by different users. (This linking is trivial in a pseudonymous system with fixed pseudonyms.) When we say that a

system is fully anonymous, we mean that it has this level of unlinkability. See Pfitzmann and Hansen [138] for a more thorough discussion of pseudonymity, anonymity and unlinkability.

Alternatively, we can measure a user's anonymity mathematically [58, 145]. To this end, we calculate for each user the probability that she performed a given action, taking into account the prior knowledge of the adversary. Full anonymity in this setting means that all users are equally likely to have performed an action.

These ranges in anonymity have two consequences when dealing with revocable privacy. First, you can lose anonymity (because you violated a rule) without becoming fully identified. For example, you could become temporarily linkable as in Chapter 4. Second, it is better to see losing anonymity in relation to other participants in the system, as some systems may not offer full anonymity in the first place.

3.1.2 *Improving the definition*

Hoepman [82] originally defined revocable privacy as follows:

> "A system implements revocable privacy if the architecture of the system guarantees that personal data are revealed only if a predefined rule has been violated."

There are some problems with this specific definition. First, the rule explicitly mentions *personal* data. Companies, however, might have an equally big desire for protecting their corporate data (e.g., their business processes). Moreover, as we explored in the previous section, revealing personal data is not always necessary; there are other ways to lose anonymity, for example, by making actions linkable. Sometimes, it even suffices just to block the user.

The definition could also be extended to include cases where revealing a user's personal information could be positive to the user, rather than just negative, as we have discussed so far.[2] One example is privacy-friendly matching on a dating site, where you get each other's contact information only if the profiles match. However, we think that such systems should not be classified as having revocable privacy, as this makes the definition too broad, almost to the point of including all privacy-enhancing technologies.

The second problem we have with this definition is that it is very easy to misread it and assume that if a person were to violate the rule then personal data are revealed. However, it does not say that. It states just that personal data *may* be revealed only if the rule is violated.

2 In fact, we suggested this approach in our technical report [110].

We incorporate these suggestions into the following revised definition of revocable privacy. We focus on anonymity rather than personal data and rephrase the rule to clarify that violating a rule does not necessarily imply the release of personal information.

DEFINITION 3.1 A system implements revocable privacy if the architecture of the system guarantees a predefined level of anonymity for a participant as long as she does not violate a predefined rule.

As required, this definition does not say anything about the consequences when a participant does violate the rule. In practice there will be a consequence. If the system implements revocable privacy this usually means that the participant loses anonymity. However, this definition also allows for revocable privacy systems where the consequences do not reduce anonymity, for example when the violator is blocked from making further actions.

3.1.3 *Systems and rules*

In the above definition, we consider the *system* as the environment with which the user interacts, and within which certain rules should be maintained. For example, in Chaum et al.'s electronic cash scheme, this system is the payment environment.

Rules are part of the system, and we require them and their parameters to be predefined. For example, if the abstract rule is "A participant is allowed to cause an event at most k times", then the threshold k should be defined for every instantiation. This requirement prevents function creep and ensures clarity. If, instead, parameters should be configurable afterwards—e.g., if some criteria are not known in advance—the rule should explicitly state this. Similarly, using a trusted third party to enforce a rule is not allowed. If, instead, it is necessary to rely on the decision of third parties, the rule should explicitly capture that too.

We impose no other restrictions on the rules as this allows us to best capture the notion of 'anonymity until violation of the rules'. In particular, we do not demand the rules to be known to the participants. While it is better that the rules are known to prevent the chilling effect, there might be circumstances where they must be kept secret.

We realize that the freedom in choosing rules (and keeping them secret) makes the rules a very powerful instrument. In fact, a rule might simply require all events to be output, or allow parameters to be set to non privacy-friendly values. Thus, careful scrutiny of the rules is of the utmost importance. The designers of the rules must ensure that the reduction in privacy that results from violating a rule is proportional to the detected behavior.

Ensuring proportionality is particularly important when violation of a rule does not necessarily imply that the participant is misbehaving. It may only be an indication of misbehavior (as in the canvas cutters use case, see Section 3.2.1) or even that the participant might be harmed (as in the detecting child abuse case, see Section 3.2.3).

3.1.4 *Architecture of a system*

What does it mean for the architecture of the system to protect the anonymity of well-behaving users? As we argued before, policies and procedures do not offer sufficient protection against function creep and misuse of the data in the future. We cannot assume that the raw data remain secure forever. Instead, it is much better, as Kapor [91] argues, to rely on the architecture of the system (the manner in which it is built, including the cryptography) to enforce the rules and to guarantee the anonymity of rule-abiding participants.

However, systems implementing revocable privacy cannot always offer unconditional anonymity either. It matters how the user's actions are observed within the system. For example, if the system sees what the user does, but needs to forget it, we have to trust the system to actually forget. In this section, we explore the trust assumptions in a system implementing revocable privacy.

To determine these assumptions we examine how data are collected—is the identity of the user ever known?—and how they are stored. We consider three conceptually different methods. For reference, we first describe the traditional method where the user is known and the data are stored in the clear. In the second method, the user is still identified, but only processed information is stored; the user's identity is forgotten. In the third, the user is never identified.

In all of these situations, data resulting from the user's actions are stored. A final post-processing procedure, that is based on the rule, takes these data as input and outputs data such that a (negative) consequence for the participants can be effected. Usually, these data will reduce the anonymity, but they might also be used just to block further access to the system—as is the case in blacklistable anonymous credentials (BLAC). Both how the data are encoded and how they are post-processed depend on the rule. In a system with revocable privacy, it is not possible to change the rule and then reprocess old data (that were collected using a different rule) according to the new rule.

Plaintext logging

For contrast, we first describe the obvious method for implementing a system with rules. Users are never anonymous with respect to the system. Every relevant action by the user—relevant with respect to the rules that are to be enforced—is stored together with the user's identity.

Violations of the rules are detected by checking the rules against the stored data. Since the user's identity is also stored, any consequence to the user's actions can immediately be enforced.

Any anonymity guarantees offered by such a traditional system rely on the policies and regulations that protect access to the stored data and that govern the data retention policies. Hence, trust lies in the policies. Because this is not an architectural protection, we say that such a system does *not* offer revocable privacy.

One way to bolster the protection is to add one or more trusted third parties that decide if the rule is violated and then carry out the desired consequence. Hoepman [82] says such a system, which he calls of the *spread responsibility* type, does have revocable privacy. However, since the system does not enforce the rule we do not consider that to be the case in this chapter. Instead, we focus solely on systems that, according to Hoepman, have a *self-enforcing architecture*, where the architecture determines if the anonymity guarantee can be weakened.

Similarly, Stadler, in his thesis [155], also proposes the use of trusted third parties (or 'trustees') that can revoke a users anonymity when there is a need to do so. Since these trusted third parties can enforce any rule, we do not consider these systems to have revocable privacy.

Schwartz et al. take a related approach to revocable privacy in their contractual anonymity system RECAP [144]. In our terminology, using RECAP, a user agrees to abide by a set of rules by registering her identity and the set of rules with an accountability server. Thereafter, the user can interact anonymously with the system. If she ever violates the rules, the system can ask the accountability server to reveal the identity of the user after presenting proof of the violation. The key difference with Stadler is that the accountability server is not arbitrarily trusted, instead, users use remote attestation to verify that the software running on the accountability server will reveal a user's identity only if she violates the agreed upon rules. While relying on the physical security provided by trusted computing (or hardware security modules for that matter) is surely better than simply trusting a third party, we focus instead on solutions that rely solely on cryptographic assumptions.

Non-interactive sensors with encoding

The second method uses sensors to process users' actions before storing them in encrypted form. This approach drastically improves the anonymity guarantees, without requiring changes to the users of the system. As with plaintext logging, the actions of the participant and its identity are visible to the system, however, they are never stored directly. Instead, a sensor (there can be many sensors in a system) observes these actions and identities, and then transforms them, based on the rule, into encrypted data. Only these encrypted data are stored. The sensors are non-interactive: they do not communicate with the observed user.

The encryption method is special. There is no key that can be used to decrypt the data. Only when the encrypted data correspond to a violation of the rule, can they be decrypted to produce some useful output.

To guarantee anonymity, we need to trust that the sensors behave as specified. In particular, we trust that sensors do not store their inputs. In addition, many sensors use private keys, in which case we trust them to keep these secret too. These private keys ensure that even if the sensors' outputs are deterministic (i.e., the sensors do not use randomness) an attacker cannot simply confirm a suspected event based on the stored data by simply calculating the same function as the sensor.

Despite these strict trust assumptions on the sensor, these systems can be very useful because they can be used as a drop-in replacement for traditional systems. In particular, they do not require any changes to the user's side. For example, non-interactive sensors can be used to enforce speed limits based directly on cars' license plates (rather than equipping all cars with equipment to communicate directly with road side equipment). Of course, the sensors and the rest of the system still need to be adapted to work with the encrypted data. In some sense, the sensors act on behalf of the user.

In Chapter 5 we construct a distributed encryption scheme that is of this type, i.e., the sensors are non-interactive and do not communicate with the observed parties. It shows, among other things, how to implement speed limiting with revocable privacy.

MITIGATING THE TRUST REQUIRED IN THE SENSORS. In some cases, such as the threshold system described in Section 3.3.1, the sensors are distributed. In this case, it may be possible that some are compromised, while the system as a whole still offers (some) anonymity.

Another approach that is useful in this setting is to make sure that the system is forward secure. Loosely speaking this would imply that if a sensor is compromised, it impacts only future events. Our distributed encryption scheme in Chapter 5 has this property. We will discuss this property further in that chapter.

Interactive sensors

In the third and final method, there is no sensor that simply observes the user; the user herself needs to be actively involved and interact with the sensor. We see this structure in existing systems offering revocable privacy. For example, in Chaum et al.'s electronic cash system the user interacts with the receiver to spend a coin, whereas in BLAC the user constructs a proof that at that point in time she was not blacklisted.

The advantage of this approach is that the user can use this interaction to hide some information, in particular her identity, from the sensor, so that it is never known to the system. As a result, the user's anonymity does not rely on the trustworthiness of parties within the system. The downside is that the user needs to interact with the sensors.

However, the user cannot hide all data. The system still needs to enforce the rule. In systems with interactive sensors, the users provide the encrypted data that are used to enforce the rule themselves—as opposed to systems with non-interactive sensors, where sensors produce these encrypted data on behalf of the user.

Sensors should verify that the data constructed by the user is correct, lest the user avoid the consequences of violating the rule. However, since the user's identity is hidden, the sensor cannot rely on its own inputs to verify correctness. Instead, this burden falls on the participant: she needs to convince the sensor that the supplied data is correct. For example, in BLAC the user uses an anonymous credential to bind the data she produces to her identity, without revealing that identity. To enable these proofs, the user usually keeps track of some secret information.

In Chapter 4 we construct a vote-to-link scheme that is of this type, i.e., the users communicate actively with the sensor. In this scheme, users prove that they supplied the correct values by relating them to anonymous credentials in their possession.

3.2 USE CASES

We now present a number of use cases that could benefit from revocable privacy. These use cases are the result of interviews with security experts, internal discussions and a review of the privacy-enhancing technologies literature. This overview is by no means exhaustive. Instead, it serves as a motivation for revocable privacy and as a source of insight into the abstract rules underlying these cases. We use these abstract rules to determine which cases we can already solve, and for which ones we need to develop new primitives.

We omit some of the use cases from our original analysis [110]. As we discussed in Section 3.1.2, we omit cases where the user would benefit from having its anonymity revealed. Other cases we omit because

they are too vague or not interesting. Finally, we omit cases that simply give too much power to a government agency, even if only suspicious behavior would be detected.

We sort and categorize the use cases based on the type of rule that the system should enforce. The rules are roughly ordered by complexity. We start with three simple classes. The first class is that of threshold rules, where an event should not happen too often. The second class, containing predicate rules, consists of rules that logically combine simpler events. The third class covers cases where the rule encodes a human decision making element—for example, a judge signing a warrant.

Next, we consider two classes of more complicated rules. The first class covers rules that are more complex than any of the aforementioned. For example, rules about flows on graphs (useful in tax situations) and about combining (private) information into the decision making process. The second class covers rules that are fuzzy and would normally, even when no anonymity is required, involve machine learning and data mining techniques.

For each of these classes we present several use cases. For every use case, we describe the case, extract an abstract rule and note the consequence of violating that rule. While the use cases focus on specific scenarios, the abstract rules generalize the rule within these scenarios. It ignores the scenario specific details. This makes it easier to determine which use cases have similar rules, and which techniques might be used to solve a use case using revocable privacy. Table 3.1 presents an overview of all use cases, recording the type of sensor,[3] the source of the use case, and potential solutions.

3.2.1 *Threshold rules*

A threshold rule has the form "a certain action should be performed no more than k times (within a certain time period)". The most common consequence of violating the rule is to reveal the violator's identity, however, it is also possible to block the user. A threshold of one is possible in some of these scenarios. The following use cases work with threshold rules.

Canvas cutters

This case, as well as the following two cases, focusses on detecting bad or suspicious behavior involving cars. As cars are generally not able to communicate with roadside equipment, we focus on the scenario where

3 Since we limit ourselves to revocable privacy solutions, we only consider non-interactive and interactive sensors, and omit plaintext logging.

an automatic number plate recognition system reads the license plates of passing cars. This makes using a non-interactive sensor the only viable option.

Criminals frequently loot trucks parked at rest stops by cutting the canvas that protects the goods (hence the name of the use case). One way to detect these criminals is to look for cars that enter several different rest stops within a couple of hours [158]. These cars are suspicious. While false positives cannot be eliminated—e.g., police cars and road-side assistance vehicles may cause them as well—most hits will correspond to suspicious behavior.

ABSTRACT RULE Given n sensors at different locations, a participant should trigger fewer than k different sensors within a given time period.

CONSEQUENCE OF VIOLATION The system learns the identity of the participant.

Object surveillance

Related to the previous problem is the problem of casing: criminals checking out a location, such as a sensitive piece of infrastructure, multiple times. These criminals can be detected by looking for cars that pass by this location frequently. This case is not the same as the canvas cutters use case. In particular, here all events contribute to the threshold, whereas for the canvas cutters case the number of different locations of the events matters. Again, false positives cannot be eliminated.

ABSTRACT RULE Given one or more sensors, a participant should trigger fewer than k sensors (counting repeats) within a given time period.

CONSEQUENCE OF VIOLATION The system learns the identity of the participant.

Average speed checking

Besides spot checking with a speed camera, some countries have deployed average speed checking systems which measure a car's speed along a stretch of road. For spot checks it is immediately clear whether an observed car is speeding. However, average speed checking requires some form of storage to determine the time it took a car to traverse a stretch of road. Phrased as a revocable privacy problem: the system should output the license plates of cars that pass two measuring station—one in the beginning and one at the end—within a too short time period.

ABSTRACT RULE Given n sensors at different locations, a participant should trigger fewer than k different sensors in any time period of a given length.

CONSEQUENCE OF VIOLATION The system learns the identity of the participant.

Sale of valuable objects

Consider the notarized sale of valuable objects such as houses. Objects that change hands frequently may indicate fraud or money laundering and may therefore be suspicious. To detect these sales—while keeping non-suspicious sales private—authorities may want to inspect those objects that are processed by many different notaries.[4]

In this scenario, keeping sales private protects the privacy of the legitimate owners. However, the system operates on the identity of the object. Since these objects cannot yet act on their own behalf, non-interactive sensors must be used.

ABSTRACT RULE Same as for canvas cutters (but for objects instead of participants).

CONSEQUENCE OF VIOLATION The system learns the identity of the object.

Anomalies in logs

Servers keep activity logs. These logs can be used to detect attacks. One example of such an attack are repeated log-in attempts from the same remote system. These are easy to spot in the logs as they all originate from the same system. However, it is usually not necessary keep the logs for all the authentic users.

By nature of the system (the remote systems are identified by IP address) we can use non-interactive sensors to detect which remote system makes frequent fraudulent login attempts. These systems can then be blocked.

ABSTRACT RULE Same as for object surveillance.

CONSEQUENCE OF VIOLATION The system learns the identity of the participant or the system blocks the participant from further accessing the system.

4 Alternatively, the system may also detect objects that are sold often in general, making the rules the same as in the object surveillance use case.

Sharing anonymous resources

Some systems give people anonymous access to a resource on the basis that they can prove something—e.g., that they have a license to a game, or that they are of a certain age. While this anonymity is good for the user, it also makes it trivial to share the access with any number of people without detection. To limit this sharing, people could be allowed only a limited number of accesses per time period. When this value is exceeded—it should be chosen in such a way that under normal use it is not—the identity of the presumed sharer is revealed or the presumed sharer is blocked from accessing the system.

Since the user and the system already interact, we prefer interactive sensors.

ABSTRACT RULE Participants can perform actions. A participant of the system should perform each action fewer than k times per time period.

CONSEQUENCE OF VIOLATION The system learns the identity of the participant.

No-shows in anonymous reservations

Consider anonymous reservations of resources such as cinema seats, museum access or computing resources based on unlimited access subscriptions. Resources are often scarce, making no-shows undesirable. For example, the (non-anonymous) restaurant reservation app OpenTable allows restaurants to charge fees for no-show reservations and OpenTable deactivates accounts in case of too many no-shows.[5]

If the system is fully anonymous, however, it is not possible to discourage no-shows. Instead, we would like to construct a system that either blocks or deanonymizes a user if she does not use a reservation, or fails to do so too often, but lets honest users be anonymous. Note the difference with the abstract rule for sharing anonymous resources: the event that should not happen too often is negative (i.e., not claiming the reserved resources) rather than positive.

ABSTRACT RULE Participants can reserve resources. Participants may fail to claim a reserved resource fewer than k times per time period.

CONSEQUENCE OF VIOLATION The system learns the identity of the participant or the system blocks the participant from making further reservations.

5 https://community.opentable.com/t5/My-Account/What-is-your-cancellation-and-no-show-policy/ta-p/95 last visited July 3, 2017.

Electronic cash

As we discussed in the introduction, another common case for revocable privacy is electronic cash [43]. Users are given digital coins that they can spend anonymously. However, they are not allowed to spend the same coin twice. This form of electronic cash is a threshold system with a threshold of two.

As before, using a non-interactive sensor is not desirable as the user already interacts with the receiver of the coin when she is spending it.

ABSTRACT RULE Participants can perform actions (e.g., spend a specific coin). Participants may perform each action at most once.

CONSEQUENCE OF VIOLATION The system learns the identity of the participant.

3.2.2 *Predicate rules*

Not all rules are as simple as limiting the occurrence of an event. In this section we consider a class of rules that combine different indicators, similar to logical formulas.

Social welfare fraud

In the Netherlands people can receive social welfare when they are unemployed. The amount received depends on the number of people in the household. In particular, people receive less welfare when they share living expenses. Some people defraud the system by incorrectly reporting that they live alone.

To detect possible cases of fraud, the municipality of Groningen looked for people who received social welfare and who indicated living alone but had higher utility consumptions (water, gas, electricity and waste) than would correspond to a one person household.[6] This search required collecting information from different sources. Using revocable privacy, it would be possible to combine these data, and only recover the suspected violations.

Data can be supplied to the system either using non-interactive sensors (for example, the utility companies and the government) or directly by the cooperating welfare recipients (the system verifies that they behave honestly).

6 The original source, http://gemeente.groningen.nl/algemeen-nieuws/2010-1/sociale-dienst-spoort-bijna-driehonderd-gevallen-van-bijstandsfraude-op (Dutch, last accessed January 29, 2012), is currently unavailable. The same technique is mentioned on http://www.nu.nl/politiek/2670044/aanpak-bijstandsfraude-bestandskoppeling.html (Dutch, last accessed February 17, 2017).

ABSTRACT RULE Let $\mathcal{D}_I$ be a set of data items associated with a participant I, and let $\mathcal{P}$ be a predicate over such data items. Then $\mathcal{P}(\mathcal{D}_I)$ must be false.

CONSEQUENCE OF VIOLATION The system learns the identity of the participant.

Detecting terrorist activity

Contrary to the canvas cutters use case, law-enforcement-like cases may instead depend on a combination of various indicators to find criminals. One rather primitive example works as follows. A person who buys fertilizer, rents a van and scouts a government building in a short period of time may be planning to make and set off a bomb.

Any one of these events might be totally benign. It is only the combination that leads to suspicion. In practice, the rules may be more complicated and involve different data items. Usually, the actual actions and the identity of the person performing them are known, making non-interactive sensors the most obvious choice.

ABSTRACT RULE Same as for social welfare fraud.

CONSEQUENCE OF VIOLATION The system learns the identity of the participant.

3.2.3 *Decision rules*

All previous rules depend only on the inputs they receive. Given these inputs, the outcome is clear. People do not take part in the decision making process. However, sometimes this decision process is essential. For example, we do not know how to codify the rule "posts should not be offensive." Such a rule is better suited for human decision making. In this section, we discuss a few rules that include human decision making.

Detecting child abuse

This first rule is actually a threshold rule, but with human decisions as input. Professionals working with children, e.g., doctors and teachers, may suspect abuse. However, for fear of causing undue panic and because reports would become part of the child's record, they may decide not to report this. These concerns would be alleviated if these reports could be made in such a way that a child's identity becomes available only when a predetermined number of professionals agree that a child might be abused. In this situation using an interactive sensor is truly

undesirable as it would alert the child or its guardians to the suspicion of abuse.

ABSTRACT RULE Given n deciders, fewer than k deciders should mark a participant as suspect.

CONSEQUENCE OF VIOLATION The system learns the identity of the participant.

Blocking anonymous edits

In the previous case it was essential that there was no interaction with the participant (the child). Here, we consider another case where interaction *is* possible: anonymously editing Wikipedia pages. Given the sensitive nature of some Wikipedia pages, it would be beneficial to allow anonymous edits. Yet, this anonymity can also facilitate abuse, and this abuse is usually not easily detected automatically. Yet, people are good at this task, in fact, Wikipedia is based on this principle.

To protect the system, an anonymous user should be blocked from making further edits if one or several of her edits have been classified as abusive. Even though a moderator can classify an edit as abusive, and thus block a user, the moderator should never be able to recover the identity of the editor. Contrary to the detecting child abuse case, the following abstract rule acts on the participant's actions, rather than on the participant herself.

ABSTRACT RULE Participants can perform actions. A participant should perform fewer than k bad (as determined by the system) actions per time period.

CONSEQUENCE OF VIOLATION The system blocks the participant from performing further actions.

Linking anonymous edits

By blocking malicious editors the system limits the damage that these editors can do in the future. However, as we explain in Chapter 4, this might not suffice. A lot of damage might already have been done by the malicious editor before she is eventually blocked.

To help the system recover from abuse, the vote-to-link system from Chapter 4 instead links previous actions of a malicious editor within a limited time window (say, a day). Since this decision is rather invasive, the edits should only be linked if a sufficient number of moderators agree to do so.

Since the user already interacts with the system, an interactive sensor should be preferred (in particular because a non-interactive solution would allow the system to see the participant's identity).

ABSTRACT RULE Given n moderators. Participants can perform actions. Each action should be marked as bad by fewer than k moderators.

CONSEQUENCE OF VIOLATION The system links the actions of the participant within a predetermined time window.

Deanonymizing comments

Like edits on Wikipedia, some posts on an online bulletin board might be made anonymously. Another method of discouraging abusive comments is to hold the authors accountable by actually revealing their identity. As with linking, the identity of the author should only be revealed if a sufficient number of moderators agree to do so.

It is possible to build this system with a non-interactive sensor. However, the user already interacts with the system, so an interactive sensor provides better privacy.

ABSTRACT RULE As for linking anonymous edits.

CONSEQUENCE OF VIOLATION The system learns the identity of the participant.

Wiretapping policy

Typically, law enforcement agencies require permission, for example from a judge or other authority, before they can legally tap phone and internet connections. However, this is enforced only by policy.

To increase privacy, telecom operators could send the requested data to law enforcement agencies in such a way that the agencies can only access this information after the required permission has been obtained. In this case, the telecom operator acts as a non-interactive sensor.

This case is more restrictive than general key-escrow methods like the Clipper chip [126], which would also allow decryption of communication as well as access to prior communications. Instead, the focus here is to enforce the legal requirements that law enforcement must obtain permission from a judge before they may place a wiretap and thus receive future communications (the data received as a result of the tap might still be encrypted by the user).

ABSTRACT RULE Given n trusted parties. Participants can perform actions. None of the trusted parties decides that the participant's behavior is suspicious.

CONSEQUENCE OF VIOLATION The system learns the actions of the participant that occur after the participant has been marked suspicious.

3.2.4 *Complex rules*

We now discuss rules that are more complex, for example because they operate on graphs and labeled data, or because they use auxiliary information that should be protected as well.

In principle, the rules in this class can be described by any deterministic computer program. However, to illustrate how hard some of these tasks can be, we also discuss fuzzy rules, based on for example machine learning, in the next section.

Riot control

In 2009/2010 there were riots between two ethnic groups in Culemborg (a city in the Netherlands). The police knew that the rioters might receive reinforcements from certain parts of the country. To prevent them from arriving, they wanted to detect these groups en route, and block the highway exits into Culemborg at the appropriate times.

To detect these groups, they automatically read license plates. If a group of more than four cars originating from the reinforcement area was detected on the highway, they closed the highway exit.

Two things make this case interesting. One, the goal is not to deanonymize specific cars, but rather to detect a group of cars from a specific area. Two, in order to make this system work auxiliary information is required about where cars are registered.

ABSTRACT RULE Let $\mathcal{D}_I$ be a set of data items associated with a participant I, and let $\mathcal{P}$ be a predicate over such data items. Given a sensor observing participants, then fewer than k participants I such that $\mathcal{P}(I)$ is true should be observed by the sensor within a given time period.

CONSEQUENCE OF VIOLATION The system learns that a match has been found.

Money flow anomalies

Some types of tax fraud manifest themselves in discrepancies in money flows between companies. In particular, whatever company A claims to have sold to B should also be reported as bought from A by B. However, cash flows between companies also reveal strategies and other sensitive economical information that companies would rather not share.

Instead of just sharing this information, the tax office and the companies could build a revocable privacy system where a company's name is revealed only if it incorrectly reported its cash flow.

ABSTRACT RULE Given a graph, with the participants represented by nodes and the edges representing money flows between them. Participants should report the flows over their adjacent edges correctly.

CONSEQUENCE OF VIOLATION The system learns the identity of the participants and their reported flows.

3.2.5 *Fuzzy rules*

Until now we discussed situations where the participants are easy to recognize because they have a unique identifier (e.g., license plate, social security number, name). However, this is not the case, for example, when only video of a person is available. Even if it is possible to recognize people using facial recognition, we may still need to recognize suspicious behavior. This brings us to the realm of fuzzy and probabilistic computation. We consider this class separately because we suspect it is even harder to solve these cases using revocable privacy.

Object surveillance based on people

This first use case is similar to the object surveillance case earlier, but with the twist we described in the previous paragraph: we have only video of the people in the system. We want to know if someone cases a location, but without the convenience of a fixed identifier.

In a system without revocable privacy, we could (maybe) collect facial features of all recorded people and determine how often they show up. To do this in a privacy friendly manner would require a system that can take faces as input, and keep track of how often a specific face has been seen. Furthermore, even if this works on features that are derived from the image, the original(s) are necessary to make future identification possible. This is why we classify this case as fuzzy.

ABSTRACT RULE Given one or more cameras acting as sensors, a participant should be seen by fewer than k sensors (counting repeats) within a given time period.

CONSEQUENCE OF VIOLATION The system obtains a picture or video of the participant.

Table 3.1: An overview of the use cases and their sensor type, source (KLPD is the Dutch abbreviation for the Dutch National Police), and applicable techniques. The sensor type is non-interactive (N-I), interactive (I) or both. The techniques are distributed encryption (DE, Section 3.3.1 and Chapter 5), *k*-times anonymous credentials (*k*-AA, Section 3.3.1), blacklistable anonymous credentials (BLAC, Section 3.3.2), the vote-to-link system (VTL, Chapter 4), group signatures (GS, Section 3.3.2) and secure multi-party computation (MPC, Section 3.4).

Use case	Type	Source	Technique
Canvas cutters	N-I	KLPD	DE
Object surveillance	N-I	KLPD	Unknown
Average speed checking	N-I	KLPD & Lueks et al. [114]	DE
Sale of valuable objects	N-I	Lueks et al. [114]	DE
Anomalies in logs	N-I	Biskup and Flegel [16]	Unknown
Sharing anon. resources	I	Camenisch et al. [31]	*k*-AA
No-show reservation	I	Internal discussion	Unknown
Electronic cash	I	Chaum et al. [44]	*k*-AA
Social welfare fraud	both	Municipality of Groningen	Unknown
Terrorist activity	N-I	Internal discussion	Unknown
Child abuse	N-I	Internal discussion	DE
Blocking anon. edits	I	Tsang et al. [159]	BLAC
Linking anon. edits	I	Lueks et al. [112]	VTL
Deanonym. comments	both	Interview	GS & VTL
Wiretapping policy	N-I	Interview	Unknown
Riot control	N-I	KLPD	Unknown
Money flow anomalies	I	Sharemind application [19]	MPC
Object surveillance 2	N-I	Internal discussion	Unknown
Camera footage	N-I	Internal discussion and Sound Intelligence [153]	Unknown

Retrieving camera footage after a crime

Many cities install cameras to increase safety. One way to use these cameras is as a remote viewing tool, so that it is easier to monitor many locations at once. However, often the camera feeds are also stored in case something untoward happens later on. When nothing bad happens, the data can safely be discarded. The data are only stored to obtain more information after a crime has been detected.

If the system automatically detects bad situations (for example based on sounds [153]), the system could encode the data, and only release past records when a bad situation is detected. Hence, the system guarantees that the privacy sensitive recordings are released only when necessary.

Relying on a human operator to make the decision to reveal footage, would put this use case back in the decision-making class of cases.

ABSTRACT RULE Given video cameras (placed at different locations) observing the behavior of participants, the system (or the operator thereof) should not detect improper behavior.

CONSEQUENCE OF VIOLATION The system obtains footage around the time of the violation.

3.3 TECHNOLOGIES

In this section, we review some technologies from the past twenty years that can be used to implement revocable privacy. Table 3.1 shows which techniques apply to which use cases.[7] This list is by no means an exhaustive literature review. In particular, there exist many variants of the schemes that we mention here. This section primarily serves as a brief overview of techniques that apply to the use cases we collected.

3.3.1 *Threshold primitives*

First, we discuss primitives that can implement threshold rules.

Distributed encryption

The distributed encryption primitive [83, 114] was specifically designed to solve revocable privacy problems with a threshold rule, in particular, the canvas cutters scenario. As such, it describes how non-interactive

7 Biskup and Flegel proposed a system [16] to solve the object surveillance and anomalies in logs cases. However, their solution requires the sensor to store a (partial) record of all events, it thus does not offer the anonymity that our definition of revocable privacy requires. We do not know of a solution for these cases that implements revocable privacy as we defined here.

sensors (automatic number plate recognition (ANPR) stations at the rest stops) can encrypt messages (license plates) in such a way that only if enough encryptions (by different stations) of the same message are available they can be combined to recover the original message.

Obviously, any corrupted sensor can encrypt any message of an attacker's choosing. So the system guarantees security only as long as not too many sensors are corrupted.

The distributed encryption primitive counts only events, while many of the use cases count events per time period. In most cases, it suffices to start a new instance of the distributed encryption scheme for every new time period, thereby enforcing that the counts are reset after each interval ends. If it is instead required that no more than a number of events happen in any time interval of a given length (no matter when that interval commenced), this approach does not work. Chapter 5 describes how to approximate the desired behavior by using overlapping instances of a distributed encryption scheme.

Combining the encrypted messages to recover the messages is not very efficient; it is exponential in the threshold. However, if the number of messages is small and they can be enumerated, like for license plates, then the batching technique in Chapter 5 allows the system to trade space for time, making it reasonably efficient.

k-times anonymous credentials

Whereas the previous primitive uses non-interactive sensors, *k*-times anonymous credentials [31] allow participants to directly interact with the sensors. As a result, the trust assumptions are much weaker. The system gives every user a credential. The user can use this credential to anonymously authenticate *k* times per time period. If the user authenticates more often, her identity becomes known.

Effectively, the user can create *k* different (random) numbers per interval. If the user authenticates more often, she is bound to reuse one of the previous ones. This makes it easy and efficient to detect violations of the rule. Extensions make it also possible for a user to exceed the limit a couple of times (possibly in different time periods) before her identity is revealed. These schemes are a generalization of electronic cash schemes [10, 32, 43, 44] where the threshold is one: every coin may be spend only once.

3.3.2 *Decision primitives*

We now discuss techniques that can be used to implement decision-based revocable privacy rules.

Blacklistable anonymous credentials

Blacklistable anonymous credentials [159] make it possible for a service provider to block users from future authentication if the user misbehaved in an earlier session. To enable this, the user uses his (certified) private key to generate a new, random token for every authentication. These tokens are bound to the user (but, without the user's private key it is impossible to determine to which user they belong). In addition, the user proves that it did not generate any of the tokens that the service provider placed on the blacklist.

If the service provider later detects abuse, it can add that session's token to the blacklist. The corresponding user can then no longer prove that its tokens are not on the blacklist and loses access to the service. Alternatively, the user can prove that it has no more than k tokens on the blacklist, thus the system allows a few bad actions.

The complexity of this protocol is linear in the number of items on the revocation list, making it inefficient. Some techniques can be used to reduce the complexity [80].

Blacklistable anonymous credentials are a specific form of anonymous blacklisting systems—we refer to Henry and Goldberg [79] for an overview—that are particularly privacy friendly. There exist other schemes that are more efficient, but they rely on trusted third parties to enforce the rules.

Group signatures with distributed management

A group signature scheme allows members of a group to digitally sign documents on behalf of the group [45]. The signers are anonymous in the sense that it is known only that they belong to the group, not which specific member signed a specific message. A special party, the tracing agent, can overcome this anonymity and determine who created a specific signature, thereby revealing the identity of the signer.

Already this scheme can be used to implement the simple rule that you lose your anonymity when the tracing agent decides that this should happen. However, in some sense we are then back to having a single, trusted third party. Instead, we can distribute the powers of the tracing agent. In a group signature scheme with a distributed tracing agent, several agents need to cooperate before the identity of the signer can be recovered [70]. As long as at most a predefined number of the tracing agents are malicious, the anonymity of honest users is guaranteed.

3.4 ANALYSIS

In the preceding section, we reviewed some techniques that can be used for revocable privacy. Unfortunately, we do not know of existing primitives for many of the more complex rules. Only the threshold use cases are covered reasonably well by existing techniques. Since the object surveillance and anomalies in logs cases are similar in structure, these cases might admit a simple solution too.

For decision-based rules, there are some existing techniques that help solve some of the use cases. This again suggests that we might be able to develop techniques for the remaining wiretapping policy case.

Nevertheless, there are some very generic techniques that could help implement the remaining rules using the revocable privacy paradigm. First, the field of privacy-preserving data mining [3] might help in solving some of the anomalies like social welfare fraud.

Finally, secure multi-party computation allows multiple parties, each with their own private input, to compute any shared function over the data, see, for example, Smart [150, Chapter 22] for an introduction. All inputs are kept private; only the output is shared. While this technique works, in theory, for any computation, including machine learning algorithms, it is also very computationally intensive.

Moreover, secure multi-party computation cannot solve all rules because of its structure. Firstly, secure multi-party computation assumes that the computing parties can communicate to exchange (potentially many) messages. This is not realistic for all problems. Secondly, the underlying trust assumption for secure multi-party computation—that at least one party participating in the computation is honest—may not match the specifics of the rule.

For example, the Sharemind company successfully used their multi-party computation platform to solve several real-world problems on private data [19, 20], one of which is the money flow problem [19]. However, the rule they enforce is subtly weaker than the rule in the original problem statement: if two out of the three computing parties collude, the anonymity of properly-reporting participants can no longer be guaranteed. The original statement does not allow disclosure of properly-reporting participants under any circumstance.

3.4.1 *Limitations*

We briefly discuss two limitations of revocable privacy. The first is that to obtain better anonymity without losing security, we have to pay in computing power. This is especially the case for the non-interactive sensor techniques that we discussed. However, we think that this cost

is often acceptable. In particular, the systems we present in this thesis confirm this.

The other limitation stems from the fact that most use cases and all solutions describe positive effects. A participant performs an action, and as a result of doing so, can violate a rule. It seems much harder to handle negative events: what if you follow the rules if you do something, rather than not do it? Consider, as an example, the rule that if you observe someone misbehaving, you have to report it.

3.5 CONCLUSIONS

We have argued why revocable privacy is an important construct that can be used to increase the privacy of a system's participants while maintaining security. We have classified systems with revocable privacy into two classes:[8] those with non-interactive sensors and those with interactive sensors. Furthermore, we have clarified the definition and have generalized it to include different types of consequences for violating the rules.

We have also explored use cases that benefit from a revocable privacy approach. This not only illustrates the usefulness of revocable privacy, but also allows us to compile some abstract rules that revocable privacy techniques should be able to implement. We have described some of these techniques and showed which problems they solve.

The comparison between the abstract rules and existing revocable privacy techniques identifies interesting directions of future work in the area of revocable privacy. Based on the fact that many threshold-based rules and decision-based rules already have corresponding primitives, we expect that the remaining ones may be solvable as well. Furthermore, we identify whole classes of more challenging research direction in finding techniques for the other use cases that lack corresponding techniques, most notably social welfare fraud detection, detecting terrorist activity, riot control, and object surveillance based on people.

The first three of these rely on predicates over data associated to the participants: the first two require the implementation of rules where such a predicate must be false, whereas the third case requires the rule that the predicates may not be true for too many observed participants in a time window. Finally, the fourth is a simple threshold rule (a participant may not be observed too often), but rather than the participant's identity being available, only pictures of participants are available.

8 As we explained before, we do not consider the plaintext sensors because they do not provide revocable privacy

4

VOTE TO LINK

Many interactive websites either completely disallow anonymous access or block anonymous users from making changes—for example, Wikipedia does not allow edits by users using the Tor network.[1] While there are benefits to allowing anonymous access, websites apparently feel that the cost in terms of abuse outweighs these benefits.

In this chapter we present the vote-to-link system. It shifts the balance between the costs and benefits of anonymous access. The system allows fully anonymous actions, while, at the same time, enabling linking of a malicious user's actions to make it much easier to recover from abuse. The vote-to-link system is constructed in such a way that, after at least k moderators mark an action as bad, the system can identify all other actions by the same user within a limited time frame (which we call an epoch). Without the cooperation of at least k moderators, users' actions remain anonymous and unlinked.

To further reduce the impact of linking a user's actions—the judgement might have been erroneous, or a user's earlier actions might have been benign—the linking is limited to a predefined time window. Note that even if many moderators are malicious, the worst they can do is link a user's actions within epochs, never across epochs, nor can they directly deanonymize her.

REVOCABLE PRIVACY. The vote-to-link system implements revocable privacy. In particular, it solves the 'linking anonymous edits' use case, as described in Section 3.2.3. As long as actions are not marked as bad by at least k moderators, users retain their full anonymity. If, on the other hand, an action is marked as bad by at least k moderators, that user's anonymity is only reduced: her actions within that time frame become linkable (but her identity is still unknown). Users actively communicate with the system. The system therefore has an interactive sensor—the service provider hosting the website—and the user's anonymity is fully guaranteed even if the service provider is malicious.

1 https://en.wikipedia.org/wiki/Wikipedia:Advice_to_users_using_Tor last accessed February 17, 2017. Naturally, Wikipedia also does not allow registering sock puppet accounts via Tor (that would defeat the purpose of blocking Tor in the first place). While registering accounts outside of Tor is possible, even those accounts are normally not exempt from the Tor block (see https://en.wikipedia.org/wiki/Wikipedia:IPBE, last accessed February 17, 2017).

A LEADING EXAMPLE: WIKIPEDIA. To see why linking an abusive user's edits within a limited time frame is useful, we return to the example of editing Wikipedia. We wish to emphasize that it makes sense for editors to be anonymous—for example when they edit controversial articles. However, this anonymity also enables abuse by anonymous editors that maliciously change pages.

To reduce abuse, Wikipedia could deter users from being abusive. For example, Wikipedia could block abusive users from further accessing the system (for example, BLAC [159] allows service providers to block anonymous users without identifying them). Alternatively, it could use a reputation system that reduces an abusive user's reputation so that it is more difficult for the abusive user to make edits in the future. Or, it could even fully deanonymize abusive users in a name-and-shame fashion. However, when such measures fail to actually deter a user from being malicious, she can still do a lot of damage. Our vote-to-link system helps mitigate this damage.

A combination of four factors ensures that undeterred users can still do a lot of damage. One, people are needed to detect abuse, so, time passes before the first violation is detected and the deterrent is effected. Two, this delay is exacerbated if only a small percentage of actions is bad, requiring moderators to examine many edits before finding one abusive edit. Three, until the first violation is detected, the abusive user is free to continue acting anonymously (and these actions are, by nature of the anonymity, unlinkable). Four, an abusive user can perform many actions, as rate limiting would also adversely affect honest users.[2]

The vote-to-link system reduces the damage by invalidating the second and third factor: after sufficiently many moderators agree that a particular edit is bad, all that user's edits within a time frame (maybe 24 hours is appropriate here) become linkable. Hence, the moderation effort can be focussed on those edits that are already highly suspect.

While the actions of abusive editors can be linked (after enough moderators vote to do so) to make it easy to find all other abusive actions by that user, the anonymity of honest users is guaranteed: actions are unlinkable if there are too few votes. Moreover, the identity of an editor is never revealed. To the best of our knowledge this is a novel approach to combat malicious edits.

2 For example, the Wikipedia editor Giraffedata regularly makes about 80 edits within one hour (see https://backchannel.com/meet-the-ultimate-wikignome-10508842caad, last accessed February 17, 2017), while the most prolific Wikipedia editor Ser Amantio di Nicolao averages 60 edits an hour assuming 8 hours of editing every day of the year (see https://en.wikipedia.org/wiki/Wikipedia:List_of_Wikipedians_by_number_of_edits, last accessed February 17, 2017), while his/her burst rate is most likely much higher.

OUR CONTRIBUTIONS. Our first contribution is the idea of a vote-to-link scheme. At a high level it works as follows. For every transaction a user wants to perform at a service provider (for example, editing a Wikipedia page), she must create a linking token. The linking token is bound to her cryptographic identity. Given this token, it is possible to identify all other transactions by that user within the current epoch. A group manager provides each user with at most one anonymous credential certifying the user's cryptographic identity (we assume Sybil attacks are not easily possible, see Section 4.1.2).

To protect the linking token, the user encrypts it to the moderators' public key using a threshold encryption scheme. Finally, to perform the transaction, she sends the encrypted linking token and a zero-knowledge proof of correct encryption (i.e., she anonymously proves that the encrypted linking token belongs to the identity in her anonymous credential) to the service provider. Moderators can vote to link the user's transaction by partially decrypting the linking token (they can do so without communicating with other moderators).

If, at some point in time, enough moderators have voted, the service provider can decrypt the linking token (using the partial decryptions provided by the moderators) and hence link all the user's actions within that epoch. The anonymity of the user is guaranteed by distributing the voting power invested in the moderators. However, even if all moderators are malicious, their capabilities are limited: they can only link users within epochs, not across epochs. The service provider itself has a purely facilitating role; even if it is malicious, it cannot link a user's actions. We first introduce the high level structure of our approach in Section 4.1, introduce the basic cryptographic ideas in Section 4.2, then describe preliminaries in Section 4.3, and, finally, describe our vote-to-link scheme in full in Section 4.4.

Our second contribution is to allow moderators to vote anonymously (normally, the structure of the threshold encryption scheme identifies voting moderators). While procedural measures can protect the votes to some extent, we prefer to not rely on those to protect the identity of the voting moderators. Hence, we have created two variants of our basic vote-to-link scheme that allows moderators to vote anonymously at the cost of reduced efficiency. We present these schemes in Section 4.5.

Our third and final contribution is an implementation of our protocols, showing the practicality of our approach. We present these results, and other aspects of deploying a vote-to-link system, in Section 4.6.

While the vote-to-link scheme may seem similar to group signatures with a distributed group manager—in fact, we borrow ideas from group signatures—we wish to emphasize that there are significant differences. First, the goal of our scheme is to make a user's actions linkable within

an epoch, rather than identify the signer of a single message. Second, while distributed tracing managers allow a similar voting process, they do not allow anonymous voting. We cover related work more extensively in Section 4.7, before concluding this chapter in Section 4.8.

4.1 SYSTEM DESIGN AND ASSUMPTIONS

In this section we describe the architecture of our vote-to-link system, and its assumptions.

4.1.1 *Architecture*

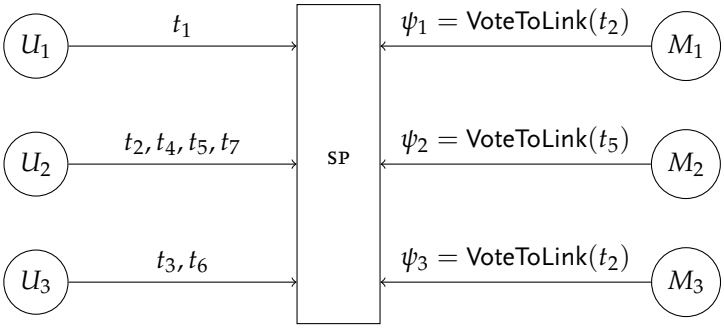
The system consists of one group manager and several users, service providers, and moderators, each with their own role in the system. For clarity, we focus on a system with only one service provider. In practice, it might be desirable to have multiple service providers. Our solution is easily extended in this direction.

GROUP MANAGER (GM) The group manager (GM) sets up the system. It ensures that a user in the system has at most one identity (for example, by requiring that a user has access to a scarce resource, see below). This ensures that the user cannot prevent linking of her actions by creating new identities. To allow the user to use this identity anonymously, the GM provides the user with an anonymous credential.

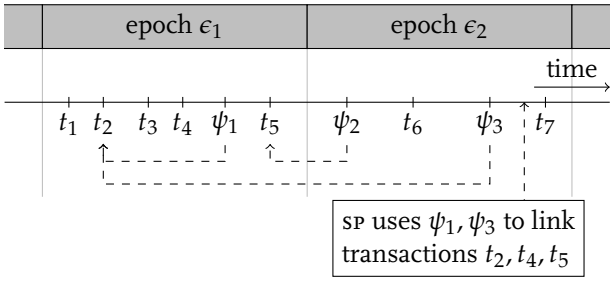
USER (U) Users access services offered by the service provider. To enable access they need an anonymous credential from the GM. The users' actions at a service provider can be linked only when enough moderators vote to enable this; normally, users' actions are anonymous. Users are not assumed to be always online.

SERVICE PROVIDER (SP) The service provider allows users to anonymously perform transactions (provided they supply the correct information). As part of performing the transaction, the user proves to the SP that she has an anonymous credential. The service provider determines the length of an epoch.

MODERATOR (M) The moderators monitor the users' activities. They can vote on an action to indicate their desire to link all other actions of the user who performed this voted-upon action. When sufficient moderators have voted on the same action, all actions by that user within this epoch become linkable. The group of moderators can be specific to each service provider. Moderators are also not assumed to be always online.



(a) Interactions among parties



(b) Timeline of interaction events

Figure 4.1: An example of the vote-to-link system with 3 users (U_1, U_2, U_3), 3 moderators (M_1, M_2, M_3) and a threshold $k = 2$. Figure (a) shows how the users and the moderators interact with the service provider: users send transactions ($t_1, \ldots, t_7$) to the service provider. The SP publishes these transactions in a public record after checking their validity. The moderators monitor these transactions, and, whenever they detect a malicious one, send a vote to link to the service provider (ψ_1, ψ_2, ψ_3). Figure (b) shows a timeline of events at the service provider. After the SP receives $k = 2$ votes on transaction t_2, it can link it to the other transactions t_4, t_5 made by the same user in that epoch (note that t_7 is not linked because it was performed in another epoch).

The groups of users and moderators do not have to be mutually exclusive. In fact, moderators may also be users, or even, all users may be moderators as well.

Because the users' credentials are anonymous, users can use them to prove that they correctly constructed the encrypted linking tokens, without otherwise revealing anything about their identity, nor becoming linkable because of using the credential. To maintain anonymity, users (and moderators) communicate with the SP via an anonymous channel.

Figure 4.1 gives an overview of an example vote-to-link system. The figure shows how a user's actions become linkable as a result of the moderators' votes. For clarity, we omitted the group manager.

Only the service provider and the group manager are assumed to be always online (but note that the only consequence of a group manager being offline is that new users cannot join). The moderators can cast votes completely non-interactively. This ensures that the system can still function even when the vast majority of the moderators are offline.

4.1.2 *Threat model and security goals*

Our system has two security goals (we formalize these in Game 4.9 in Section 4.4.2 and Game 4.12 in Section 4.5.2).

USER ANONYMITY As long as an adversary controls at most $k-1$ moderators, it cannot link an honest member's transactions with non-negligible advantage.

MODERATOR ANONYMITY Users that collude with up to $n-2$ moderators cannot determine the identity of a non-colluding voting moderator with non-negligible advantage.

The user anonymity goal strictly implies that users cannot be identified either. Furthermore, for user anonymity, we make no security assumptions about the users and the service provider. In fact, even if they are completely malicious, the anonymity of the honest users remains fully protected. Clearly, we can allow at most $k-1$ moderators to be malicious. For moderator anonymity, we assume that the user performing the transaction and the SP do not collude.

Finally, we assume that the SP acts in its own interest by honestly following the protocol, so that the actions of misbehaving users can indeed be linked after sufficient votes have been cast.

ABOUT SYBIL ATTACKS. To ensure that all user's actions can be linked after voting, we require, like many other anonymous systems with an accountability feature, that users have only one identity. Our

system by itself is not robust against Sybil attacks [60], and protecting against them is outside the scope of this chapter.

We do acknowledge that protecting against Sybil attacks is difficult. However, some protection can be achieved when the group manager issues credentials based on some scarce resource (for example, IP address, phone number or a national electronic ID card). See Henry and Goldberg for an overview [79].

4.2 THE IDEA OF THE BASIC SCHEME

After presenting the high level structure of our scheme, we now sketch how to construct the linking tokens that can be used to link a user's actions. We follow an idea by Nakanishi and Funabiki [124]. As mathematical setting we use a bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$ of prime order p with $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ the corresponding bilinear map and $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$ the generators. Let x be the user's secret key and $H : \{0,1\}^* \to \mathbb{G}_1$ a cryptographic hash function. We use this hash function to transform a description of the epoch ϵ (for example, ϵ could be the current date to get 24 hour epochs) into a generator $g_\epsilon = H(\epsilon)$.

The user's linking token for epoch ϵ is given by $r = g_\epsilon^x$. With each transaction, the user publishes auxiliary values $t_1 = h^z$ and $t_2 = \hat{e}(g_\epsilon, t_1)^x$ for a random $z \in_R \mathbb{Z}_p$. The user's linking token r—which is normally unknown—can be used to link transactions by this user. The linking token r thus acts as a trapdoor. To check whether a transaction with auxiliary values t_1', t_2' was made by a user with linking token r, the SP can simply check whether $\hat{e}(r, t_1') = t_2'$.

So, given the linking token, the SP can link the actions of the user within an epoch. To enable linking when the moderators vote to do so, the user creates an encrypted linking token $T = \mathsf{TDH.Enc}(w, r, \tau)$ by encrypting the linking token r labeled with the transaction τ with the moderators' public key w using the k-out-of-n threshold encryption scheme that we present in the next section. To ensure that the user cannot cheat, she has to prove in zero-knowledge that T, t_1, and t_2 are correct and correspond to her anonymous credential $C(x)$ certifying her cryptographic identity or secret key x.[3]

If a moderator later feels that a transaction is inappropriate, the moderator can cast a vote to link the user's actions by partially decrypting the encrypted linking token T. When a sufficient number of decryption shares are published, the vote passes, and the SP can use the decryption shares to recover the linking token r, allowing the user's transactions to be linked.

3 Recall from Section 2.7 that we write $C(x)$ to denote an anonymous credential over the private key x.

4.3 PRELIMINARIES

In this section we cover some of the (cryptographic) preliminaries necessary to describe our schemes.

4.3.1 CCA *secure threshold encryption*

To encrypt the linking tokens, we need need an encryption scheme that is both verifiable—to ensure that the encrypted linking tokens are well-formed—and threshold decryptable—to allow the moderators to create decryption shares when necessary. Furthermore, in our user anonymity game (see Game 4.9) the adversary can request decryption shares of any ciphertext except the challenge ciphertext. To ensure that it cannot learn anything about the challenge ciphertext, we require CCA security.

In this section we show a variant of the Shoup and Gennaro's TDH2 threshold encryption scheme [147] that can verifiably encrypt group elements, rather than strings. The core of the original scheme is similar to hashed ElGamal in a cyclic group $\mathbb{G}$ generated by g: the ciphertext of a message m for a public key $w = g^\kappa$ is of the form $(c, u) = (m \oplus H(w^a), g^a)$. In our TDH2' variant we replace c with $m \cdot w^a$.

SCHEME 4.1 (TDH2') The setting of this scheme is in a cyclic group $\mathbb{G}$ of prime order p. Let $H' : \{0,1\}^* \to \mathbb{Z}_p$ be a cryptographic hash function. The encryption function takes a label as additional parameter. This label is not encrypted, but it is bound to the ciphertext. The decryptors use this label to decide whether they want to decrypt the ciphertext. The scheme is given by the following algorithms.

- $\mathsf{TDH.Setup}(1^\ell)$. On input of the security parameter ℓ, the setup algorithm generates a cyclic group $\mathbb{G}$ of prime order p generated by g, such that p is ℓ bits. It publishes $(\mathbb{G}, p, g)$.
- $\mathsf{TDH.KeyGen}(n, k, (\mathbb{G}, p, g))$. On input of the number n of decryptors, the threshold k, and a cyclic group $(\mathbb{G}, p, g)$ as above, the TDH.KeyGen algorithm proceeds as follows. It picks coefficients $f_0, \ldots, f_{k-1} \in_R \mathbb{Z}_p$ and defines the secret sharing polynomial $f(X) = \sum_{i=0}^{k-1} f_i X^i$. Let $\kappa_i = f(i)$ and $w_i = g^{\kappa_i}$ for $0 \leq i \leq n$. Write $\kappa = \kappa_0 = f(0)$ for the private key and $w = g^\kappa$ for the public key. Next, the algorithm generates a random generator $\bar{g} \in_R \mathbb{G}$ and publishes the public parameters $(\mathbb{G}, g, \bar{g}, p)$, the ciphertext space $\mathcal{C} = \mathbb{G} \times \{0,1\}^* \times \mathbb{G}^2 \times \mathbb{Z}_p^2$, the public key w, and the verification key $VK = (w_1, \ldots, w_n)$.[4] Every decryptor i gets the private key $\delta_i = (i, \kappa_i)$.

4 Contrary to the basic vote-to-link scheme, we cannot use the verification keys to verify the decryption shares in the moderator-anonymous schemes, since using them would

- $\mathsf{TDH.Enc}(w, m, L)$. On input of a public key w, a message m and a label L, the TDH.Enc algorithm creates a ciphertext as follows. First, it generates $a, s \in_R \mathbb{Z}_p$ and sets

$$c = m \cdot w^a, \quad u = g^a, \quad \hat{u} = g^s, \quad v = \bar{g}^a, \quad \hat{v} = \bar{g}^s.$$

  Then, it calculates $e = H'(c \parallel L \parallel u \parallel \hat{u} \parallel v \parallel \hat{v})$ and sets $d = s + ae$. The ciphertext is $\psi = (c, L, u, v, e, d)$. Note that the tuple $(\hat{u}, \hat{v}, e, d)$ basically forms a non-interactive signature proof of knowledge of the form

$$\mathrm{SPK}\{(a) \ : \ u = g^a \wedge v = \bar{g}^a\}\,(c \parallel L).$$

- $\mathsf{TDH.Dec}(\delta_i, \psi)$. On input of a ciphertext $\psi = (c, L, u, v, e, d)$ and a decryption key $\delta_i = (i, \kappa_i)$ the decryptor first verifies that the ciphertext is well-formed. To this end it calculates:

$$\hat{u} = g^d u^{-e} \qquad \text{and} \qquad \hat{v} = \bar{g}^d v^{-e},$$

  and checks that $e = H'(c \parallel L \parallel u \parallel \hat{u} \parallel v \parallel \hat{v})$. If this check fails, it returns $\psi_i = (i, \perp)$. Otherwise, it generates a random $s_i \in_R \mathbb{Z}_p$ and computes

$$u_i = u^{\kappa_i}, \qquad \hat{u}_i = u^{s_i}, \qquad \text{and} \qquad \hat{w}_i = g^{s_i},$$

  before calculating $e_i = H'(u_i \parallel \hat{u}_i \parallel \hat{w}_i)$ and the response $d_i = s_i + e_i\kappa_i$. The decryptor returns $\psi_i = (i, u_i, e_i, d_i)$. Note that the tuple $(\hat{u}_i, \hat{w}_i, e_i, d_i)$ basically forms a non-interactive proof of knowledge of the form

$$\mathrm{SPK}\{(\kappa_i) \ : \ u_i = u^{\kappa_i} \wedge w_i = g^{\kappa_i}\}\,().$$

- $\mathsf{TDH.ShareVerify}(\psi_i, VK, \psi)$. On input of a ciphertext $\psi = (c, L, u, v, e, d)$ a decryption share ψ_i and a verification key $VK = (w_1, \ldots, w_n)$ the TDH.ShareVerify algorithm proceeds as follows to check that the decryption share is correct. First, it checks that the ciphertext ψ is well-formed, as before. If ψ is not well-formed, it returns $\top$ if $\psi_i = (i, \perp)$ and $\perp$ otherwise. If the ciphertext is well-formed, then ψ_i should be of the form (i, u_i, e_i, d_i) (the algorithm returns $\perp$ if it is not). Next, it calculates

$$\hat{u}_i = u^{d_i} u_i^{-e_i} \qquad \text{and} \qquad \hat{w}_i = g^{d_i} w_i^{-e_i},$$

  to check the non-interactive proof of knowledge. It returns $\top$ if $e_i = H'(u_i \parallel \hat{u}_i \parallel \hat{w}_i)$ and $\perp$ otherwise.

identify the voting moderators. We do use the verification keys in the full moderator-anonymous scheme to check the validity of the user-generated TDH2' keys, see Section 4.5.3.

- $\mathsf{TDH.Combine}(\psi, \{\psi_{i_1}, \ldots, \psi_{i_k}\})$. Given a ciphertext ψ and a set of k shares the combine algorithm proceeds as follows. It tests the validity of its inputs: letting $\mathcal{I} = \{i_1, \ldots, i_k\}$, the algorithm checks that $|\mathcal{I}| = k$; it checks that ψ is well-formed; and, it checks that the decryption shares are valid by checking that

$$\mathsf{TDH.ShareVerify}(\psi_{i_j}, VK, \psi) = \top$$

for all $j = 1, \ldots, k$. It returns $\perp$ if any test fails. Otherwise, every decryption share ψ_i is of the form (i, u_i, e_i, d_i) for $i \in \mathcal{I}$, so

$$m = c \prod_{i \in \mathcal{I}} u_i^{-\lambda_i^{\mathcal{I}}}$$

is the plaintext. It returns m.

It is easy to see that the scheme is correct: for all messages m and ciphertexts $\psi = \mathsf{TDH.Enc}(w, m, L)$ and for all sets of k decryption shares $\psi_{i_1}, \ldots, \psi_{i_k}$ we have that $\mathsf{TDH.Combine}(\psi, \{\psi_{i_1}, \ldots, \psi_{i_k}\}, VK) = m$.

The proof of security for the original TDH2 scheme strictly relies on the hash-function used to create the ciphertext element $c = m \oplus H(w^a)$. Since we do not have such a hash function, we give a new proof of the CCA security. But, before we can do so, we need to define the CCA security game for threshold encryption schemes. We specify the game directly for our TDH2' scheme.

GAME 4.2 (Threshold CCA security [147]) The threshold CCA game between an adversary $\mathcal{A}$ and the challenger proceeds as follows.

SETUP PHASE The adversary chooses the number n of decryptors and the threshold k. It also chooses $k - 1$ decryptors it wants to corrupt.[5] The challenger first runs $\mathsf{TDH.Setup}(1^\ell)$ to set up the group $(\mathbb{G}, p, g)$, and then runs the $\mathsf{TDH.KeyGen}(n, k, (\mathbb{G}, p, g))$ algorithm. It gives the private decryption keys of the $k - 1$ corrupted decryptors to the adversary. The challenger keeps the other keys for its own use.

QUERY PHASE During the query phase, the challenger can make any $\mathsf{Dec}(i, \psi)$ queries of any non-corrupted host i with ciphertext ψ of its choosing.

CHALLENGE PHASE At some point the adversary chooses messages $m_0, m_1 \in \mathbb{G}$ and a label L and sends them to the challenger. The challenger randomly picks a bit $b \in_R \{0, 1\}$ and sends $\psi = \mathsf{TDH.Enc}(w, m_b, L)$ to the adversary.

5 Note this is the static model where the to be corrupted servers have to be announced up front.

RESTRICTED QUERY PHASE The adversary can make Dec queries as before, except at ψ.

OUTPUT PHASE Finally, the adversary outputs a bit b' as its guess for b. The adversary wins if $b' = b$.

The advantage of an adversary $\mathcal{A}$ is given by $\mathsf{Adv}^{\text{CCA}}_{\mathcal{A}} = 2\left|\Pr[b = b'] - \frac{1}{2}\right|$, where the probability is over the random bits of the challenger and the adversary. The TDH2' encryption scheme is threshold CCA secure if $\mathsf{Adv}^{\text{CCA}}_{\mathcal{A}}$ is negligible for every PPT algorithm $\mathcal{A}$.

We now give a small lemma that we will use in our proof. (We don't strictly need this lemma, but it makes the proof clearer.)

LEMMA 4.3 *The proofs of knowledge in the TDH.Enc and TDH.Dec algorithms can be simulated in the random oracle model for H'.*

Proof. We start with the zero-knowledge proof in TDH.Enc. Given any $u, v \in \mathbb{G}$ proceed as follows. Generate a random $e, d \in \mathbb{Z}_p$ and set

$$\hat{u} = g^d u^{-e} \qquad \text{and} \qquad \hat{v} = \bar{g}^d v^{-e}.$$

Then, using the fact that we operate in the random oracle model, back patch H' such that $e = H'(c \parallel L \parallel u \parallel \hat{u} \parallel v \parallel \hat{v})$. It is easy to see that this proof verifies. Also, since $\hat{u}, \hat{v}$ are random and generated by us, the back patching fails (i.e., because the adversary already queried the hash function on this precise input) with negligible probability.

Similarly, we can simulate the proof of knowledge in the TDH.Dec function. Given any u_i and w_i we proceed as follows. Pick a random $e_i, d_i \in_R \mathbb{Z}_p$ and set

$$\hat{u}_i = u^{d_i} u_i^{-e_i} \qquad \text{and} \qquad \hat{w}_i = g^{d_i} w_i^{-e_i}.$$

Then back patch H' such that $e = H'(u_i \parallel \hat{u}_i \parallel \hat{w}_i)$. It is easy to see that this proof verifies. As before, the back patch fails with negligible probability. □

THEOREM 4.4 *The TDH2' scheme is CCA secure in the random oracle model for H', assuming that the DDH assumption holds in $\mathbb{G}$.*

Proof. In this proof we show that we can replace the proper challenge ciphertext with a random ciphertext without the adversary detecting this. Clearly, in the latter case, the adversary has no chance of winning the CCA game.

Recall, the ciphertext is given by $\psi = (c = m \cdot w^a, L, u = g^a, v = \bar{g}^a, e, d)$, where (c, u) forms the actual encoding part, and (v, e, d) the proof of correctness. This proof proceeds in two steps. First, we show

that an adversary cannot detect that we replace the correct v component in the challenge ciphertext with a random element from $\mathbb{G}$, provided the DDH assumption in $\mathbb{G}$ holds. Second, we show that we can also replace the correct c component with a random element from $\mathbb{G}$ without the adversary detecting this, again, provided the DDH assumption in $\mathbb{G}$ holds. Clearly in the latter case the ciphertext is essentially random, so the adversary does not have an advantage. This proves CCA security. We now give the details.

We first prove that we can replace v in the challenge ciphertext with a random element from $\mathbb{G}$. Assume that an adversary exists that can detect whether v has been replaced by a random element. We show how we can use such an adversary to solve a DDH instance. We simulate the entire game honestly, except for the challenge query. Let $(g, X, Y, Z) = (g, g^x, g^y, g^z)$ be a DDH instance in $\mathbb{G}$. Our goal is to decide whether $z = xy$ or not. To do so, we will encode this problem into v. If $z = xy$ then v is correctly formed, otherwise it is random. Thus, any algorithm that can decide on the well-formedness of v can be used to solve the DDH-problem.

We set up the system as in the $\mathsf{TDH.KeyGen}(n, k, (\mathbb{G}, p, g))$ algorithm, with one exception. Instead of generating $\bar{g}$ randomly we pick $\beta \in_R \mathbb{Z}_p$ and set $\bar{g} = Y^\beta$. Clearly, $\bar{g}$ is a random generator from $\mathbb{G}$ as required. Let κ be the private key. Obviously, we can answer all decryption queries honestly, as the challenger knows all the required keys.

Now we show how to answer the challenge query for the encryption of message m_0 or m_1 with label L. The challenger first picks a random bit $b \in \{0, 1\}$. Then, it picks a random element $\alpha \in_R \mathbb{Z}_p$ and sets $u = X^\alpha$, so $a = x\alpha$. Then, $c = m_b \cdot u^\kappa$ is correctly formed. We let $v = Z^{\alpha\beta}$. Now, if $z = xy$ then $v = Z^{\alpha\beta} = (g^{y\beta})^{x\alpha} = \bar{g}^{x\alpha} = \bar{g}^a$ as required. Otherwise, v is a random element in $\mathbb{G}$. The proof of knowledge we simulate as per Lemma 4.3. So the simulation is perfect, and hence any adversary that can distinguish between a correctly formed v and a random element can be used to solve the DDH-problem. In particular, if the adversary indicates that v is correctly formed, output 1 to indicate that $z = xy$, and 0 otherwise.

Now that we have seen that we can replace the v element with a random element under the DDH assumption, we proceed by replacing the c component with a random element. Again, we assume that there exist an adversary that can detect whether c is correctly formed, given that v has already been replaced by a random element.

This time, the setup is more complicated. Again, we will use a DDH instance (g, X, Y, Z) in $\mathbb{G}$. Let g be the generator. Now, we let the public key $w = X$, so $\kappa = x$ (but, we do not know κ). Assume, w.l.o.g., that the corrupted servers are numbered $1, \ldots, k-1$. Choose their key-shares

randomly $\kappa_1, \ldots, \kappa_{k-1} \in_R \mathbb{Z}_p$, and let $w_i = g^{\kappa_i}$ be their corresponding verification keys. Let $\mathcal{I} = \{0, 1, \ldots, k-1\}$, then for $k \leq j \leq n$ we have

$$w_j = w^{\lambda_0^{\mathcal{I}}(j)} \prod_{i=1}^{k-1} w_i^{\lambda_i^{\mathcal{I}}(j)},$$

by Lagrange interpolation. Furthermore, we pick a random element $\alpha \in_R \mathbb{Z}_p$ and set $\bar{g} = w^{\alpha}$. This allows us to answer decryption queries, because the proof of knowledge essentially forces the adversary to give us $\bar{g}^a = (w^a)^{\alpha}$. This completes the setup.

We will now show how to answer decryption queries for server j, with $k \leq j \leq n$. Let $\psi = (c, L, u, v, e, d)$ be the ciphertext. If the ciphertext is not valid, simply return $(j, \perp)$. If the ciphertext is valid, then with overwhelming probability $v = \bar{g}^a$, and hence $v^{1/\alpha} = w^a = u^{\kappa}$. We now use Lagrange interpolation on the set $\mathcal{I} = \{0, 1, \ldots, k-1\}$ to find u_j:

$$u_j = v^{\lambda_0^{\mathcal{I}}(j)/\alpha} \prod_{i=1}^{k-1} u^{\kappa_i \lambda_i^{\mathcal{I}}(j)}.$$

Once again, we simulate the proof of knowledge, as per Lemma 4.3.

Finally, we show how to deal with a challenge query of two messages $m_0, m_1 \in \mathbb{G}$ and label L. First, pick a bit $b \in_R \{0, 1\}$ and set $u = Y$, and $v \in_R \mathbb{G}$ (by our first step, this is as the adversary now expects). Finally, set $c = m_b \cdot Z$. Now, if $z = xy$ then c is correctly formed as before, otherwise c is random. Any adversary that distinguishes between a well-formed c and a random c breaks the DDH assumption. In particular, if the adversary indicates that c is correctly formed, output 1 to indicate that $z = xy$, and 0 otherwise.

We have seen how, in two steps, we can modify the protocol in such a way that the ciphertext gives no information about the plaintext. Hence, the adversary cannot win. □

In our security proof for our vote-to-link scheme, we require that the TDH2' scheme is secure even if the challenger can make many challenge queries, in the sense of the following game.

GAME 4.5 (Real-or-random threshold CCA security) The real-or-random threshold CCA game between an adversary and the challenger proceeds as follows.

SETUP PHASE As in the threshold CCA security game Game 4.2. Additionally, the challenger picks a bit $b \in_R \{0, 1\}$.

QUERY PHASE In the query phase, the adversary can make the following two queries.

- $\mathsf{Dec}(i, \psi)$. The adversary can make decryption queries for ciphertext ψ of any non-corrupted host i, as long as ψ was not the result of an Enc query.
- $\mathsf{Enc}(m, L)$. The adversary can also make challenge encryption queries for message m and label L. Depending on b, the challenger answers as follows. If $b = 0$, the challenger returns $\psi = \mathsf{TDH.Enc}(w, m, L)$ and otherwise returns $\psi = \mathsf{TDH.Enc}(w, \overline{m}, L)$ for $\overline{m} \in_R \mathbb{G}$.

OUTPUT PHASE Finally, the adversary outputs a bit b' as its guess for b. The adversary wins if $b' = b$.

The advantage of an adversary $\mathcal{A}$ is given by $\mathsf{Adv}^{\text{ROR-CCA}}_{\mathcal{A}} = 2\left|\Pr[b = b'] - \frac{1}{2}\right|$, where the probability is over the random bits of the challenger and the adversary. The TDH2' encryption scheme is threshold CCA secure in the real-or-random sense if $\mathsf{Adv}^{\text{ROR-CCA}}_{\mathcal{A}}$ is negligible for every PPT algorithm $\mathcal{A}$.

A standard hybrid argument shows that if a scheme is secure according to Game 4.2 then it is also secure in the sense of Game 4.5 albeit with a loss of tightness proportional to the number of encryption queries made. See Bellare et al. [13] for an example of such a reduction.

COROLLARY 4.6 *The TDH2' scheme is CCA secure in the sense of the real-or-random threshold CCA game (see Game 4.5) in the random oracle model for H', assuming that the DDH assumption holds in $\mathbb{G}$.*

4.3.2 *ElGamal encryption*

In our full moderator-anonymous scheme, we use use ElGamal encryption [63] to encrypt the moderators' decryption shares.

SCHEME 4.7 (ElGamal encryption) The ElGamal encryption scheme in a cyclic group $\mathbb{G}$ with generator g of prime order p is defined as follows.

- $\mathsf{Setup}(1^\ell)$. On input of the security parameter ℓ, the setup algorithm generates a cyclic group $\mathbb{G}$ of prime order p generated by g, such that the size of p is ℓ bits. It publishes $(\mathbb{G}, p, g)$.
- $\mathsf{KeyGen}((\mathbb{G}, g, p))$. Given a cyclic group $\mathbb{G}$ of order p generated by g as above, choose a private key $y \in_R \mathbb{Z}_p$ and set the corresponding public key $Y = g^y$. Return (y, Y).
- $\mathsf{Enc}(Y, m)$. To encrypt a message $m \in \mathbb{G}$ with a public key Y pick a randomizer $a \in_R \mathbb{Z}_p$ and create the ciphertext $c = (c_1, c_2) = (m \cdot Y^a, g^a)$.

- $\mathsf{Dec}(y, c)$. To decrypt a ciphertext $c = (c_1, c_2)$ using the private key y compute $m = c_1 / c_2^y$.

ElGamal encryption is multiplicatively homomorphic. Defining the product of two ElGamal ciphertexts to be the result of component wise multiplication, we have

$$\mathsf{Enc}(Y, m_1) \cdot \mathsf{Enc}(Y, m_2) = \mathsf{Enc}(Y, m_1 \cdot m_2).$$

In Section 4.5 we use this homomorphic property to obtain a fully moderator-anonymous vote-to-link scheme. In that section we also use ElGamal's simple structure to randomize a public key and to transform ciphertexts for this randomized public key into ciphertexts for the original public key.

- $\mathsf{Randomize}(Y, \alpha)$. To randomize a public key $Y \in \mathbb{G}$ using a randomizer $\alpha \in \mathbb{Z}_p$ compute $\overline{Y} = Y \cdot g^\alpha$.
- $\mathsf{Derandomize}(\bar{c}, \alpha)$. To derandomize a ciphertext $\bar{c} = (\bar{c}_1, \bar{c}_2)$ that is encrypted with a randomized public key $\overline{Y} = \mathsf{Randomize}(Y, \alpha)$ with randomizer α, calculate $c = (\bar{c}_1 / \bar{c}_2^\alpha, \bar{c}_2)$.

It is easy to check that for $\overline{Y} = \mathsf{Randomize}(Y, \alpha)$ and $\bar{c} = \mathsf{Enc}(\overline{Y}, m)$ we have $c = \mathsf{Derandomize}(\bar{c}, \alpha) = \mathsf{Enc}(Y, m)$.

4.4 A VOTE-TO-LINK SCHEME

In this section we introduce our basic vote-to-link scheme. We first present the full scheme that expands upon the main ideas sketched at the start of this chapter and Section 4.2, and then prove user anonymity.

4.4.1 *Our scheme*

The following scheme formalizes the ideas presented above.

SCHEME 4.8 (Vote-to-link) Our *vote-to-link* scheme with threshold k and n moderators is given by the following algorithms.

- $\mathsf{Setup}(1^\ell, n, k)$. To set up the system, the group manager first runs $\mathsf{SetupGM}(1^\ell)$ and then $\mathsf{SetupModerators}(1^\ell, n, k)$. Finally, it generates a hash function $H : \{0,1\}^* \to \mathbb{G}_1$.
- $\mathsf{SetupGM}(1^\ell)$. The SetupGM algorithm is run by the group manager responsible for adding users. The group manager first sets up an anonymous credential scheme with security level ℓ in which the GM is an issuer. Next, it generates a bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$, both of prime order p such that the size of p is ℓ bits, with

generators g and h respectively, and a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that the DDH problem is hard in $\mathbb{G}_1$. It publishes the public information for the anonymous credential scheme, and a description of the groups, generators and bilinear map.

- SetupModerators$(1^\ell, n, k)$. Run the TDH.KeyGen$(n, k, \mathbb{G}_1, g, p)$ algorithm. This algorithm gives each moderator a voting key δ_i, and publishes the moderator public key w and the verification key VK.

- UserJoin$()$. The UserJoin protocol is run by the group manager and a user. The GM authenticates the user and confirms that she is eligible to join the system. If so, the GM issues a credential $C(x)$ over the user's secret key x to the user. The group manager stores additional information to ensure that the user only receives one credential.

- PerformTransaction(ϵ, τ). The PerformTransaction protocol is run between the user and a service provider. Let x be the user's secret key, ϵ the current epoch, and τ the transaction that the user wants to perform. The user calculates the current epoch generator $g_\epsilon = H(\epsilon) \in \mathbb{G}_1$, generates $z \in_R \mathbb{Z}_p$, and calculates the linking token $r = g_\epsilon^x$ and auxiliary information $t_1 = h^z$ and $t_2 = \hat{e}(g_\epsilon, t_1)^x$. Next, she creates the encrypted[6] linking token $T = \mathsf{TDH.Enc}(w, r, \tau)$ and generates a signature proof of knowledge that she generated all these values correctly:[7]

$$\pi = \mathrm{SPK}\{(C, x, z, \alpha) : C(x) \wedge t_1 = h^z \wedge t_1^x = h^\alpha \wedge T = \mathsf{TDH.Enc}(w, g_\epsilon^x, \tau) \wedge t_2 = \hat{e}(g_\epsilon, h)^\alpha\}(\tau). \quad (4.1)$$

  In this proof $\alpha = xz$. The user sends the transaction record $t = (\tau, T, t_1, t_2, \epsilon, \pi)$ to the SP. The SP executes the transaction if the proof π is correct and T is well-formed (see TDH.Dec in Scheme 4.1). The SP stores the transaction record t.

- VoteToLink(δ_i, t). On input of a transaction record $t = (\tau, T, t_1, t_2, \epsilon, \pi)$ and voting key δ_i, the moderator checks that the label in ciphertext T matches the transaction τ. If the label is correct, it calculates the decryption share $\psi_i = \mathsf{TDH.Dec}(\delta_i, T)$ and sends ψ_i to the SP. The SP calls TDH.ShareVerify(ψ_i, VK, T) to verify the vote before accepting it.

6 It is essential that the user verifies that the moderators' public key w indeed belongs to the moderators.

7 The part of the proof involving t_1 and t_2 has been adapted from Nakanishi and Funabiki [124]. For the $T = \mathsf{TDH.Enc}(w, g_\epsilon^x, \tau)$ conjunct the prover proves that T indeed encrypts g_ϵ^x, i.e., that it knows a such that $c = g_\epsilon^x w^a$ and $u = g^a$, where c and u are components of T.

- $\mathsf{Link}(t, \{\psi_{i_1}, \ldots, \psi_{i_k}\})$. On input of k decryption shares $\{\psi_{i_1}, \ldots, \psi_{i_k}\}$ and a transaction record $t = (\tau, T, t_1, t_2, \epsilon, \pi)$ the service provider recovers the linking token $r = \mathsf{TDH.Combine}(T, \psi_{i_1}, \ldots, \psi_{i_k})$ (or aborts if the decryption fails). The SP now uses this linking token to find all other transactions by the same user in epoch ϵ: for each transaction $t' = (\tau', T', t'_1, t'_2, \epsilon, \pi')$ in epoch ϵ the SP tests whether
$$\hat{e}(r, t'_1) = t'_2.$$
If the equation holds, the SP adds transaction τ' to the list of transactions by the same user.

The vote-to-link mechanism is effected as follows. When a moderator detects a bad transaction, it calls VoteToLink and sends the resulting decryption share to the SP (the decryption shares are stored with the transaction). When the SP has collected enough decryption shares it runs Link to find all other transactions by the same user in that epoch.[8] In addition, if a user's linking token is recovered within the current epoch, the SP can use the test in Link to block transactions from that user for the remainder of this epoch by testing the submitted t_1, t_2 values.

Correctness of this scheme is easy to verify. The service provider verifies the proof π which ensures that the user has a valid credential, that T contains an encrypted linking token, and that the auxiliary values t_1, t_2 are well-formed. Since credentials are unforgeable, a user is forced to create linking tokens belonging to her own identity. Because of the security of the revocation scheme by Nakanishi and Funabiki [124], the linking tokens can be used to link all the user's transactions within this epoch.

The $\mathsf{TDH.KeyGen}(n, k, \mathbb{G}_1, g, p)$ algorithm used as part of the SetupModerators scheme is non-interactive and relies on a trusted party to create the initial secret sharing. Alternatively, the moderators could jointly run the $\mathsf{VSS}^+(n, k, \mathbb{G}_1, g, p)$ protocol (see Protocol 2.11 in Section 2.5.1) as an interactive variant that does not rely on a trusted party.

4.4.2 *User anonymity*

We now show that well-behaving users remain anonymous. More precisely, we show that unlinked users in epoch ϵ, i.e., users for which the linking token for epoch ϵ has not been recovered, are anonymous. We first define the full user anonymity game, see also Section 4.1.2.

8 The SP should guide the voting process to prevent the situation where there are many bad transactions with only a few votes. Instead, the SP should rank suspicious transactions by the number of votes so that if a transaction is indeed bad, the threshold is reached quickly. In fact, even non-moderators could report bad actions to bring them to the moderator's attention more quickly.

GAME 4.9 (User anonymity) The *user anonymity* (UA) game is a game between an adversary $\mathcal{A}$ and a challenger. The game proceeds in five phases. Each user in the system is identified by a user identifier *id* of the adversary's choosing. The challenger keeps track of a set of honest users $\mathcal{U}_H$ and a set of users $\mathcal{U}_C$ that are under the adversary's control.

SETUP PHASE At the start of the game the adversary informs the challenger about the number n of moderators and the threshold k it wants to use. In addition, the adversary indicates a set $\mathcal{C} \subset \{1, \ldots, n\}$ of cardinality $k-1$ of moderators it wants to corrupt. The challenger runs Setup to set up the group manager and moderators. The adversary receives the keys of the corrupted moderators at the end of the SetupModerators routine. The challenger manages the other moderator keys. Finally, the challenger sets $\mathcal{U}_H = \emptyset$ and $\mathcal{U}_C = \emptyset$.

QUERY PHASE In the query phase the adversary can make the following queries:

- AddU(id). The adversary can make an AddU(id) query to request that a user with identifier *id* is added to the system. The challenger creates this user and runs UserJoin on behalf of this user with the GM. The challenger stores the user's private information and the credential. It adds *id* to $\mathcal{U}_H$.
- JoinU(id). The adversary makes a JoinU(id) query to request that a user it constructed, i.e., the adversary chooses the keys, joins the system. To this end, the adversary runs the UserJoin protocol—on behalf of the user—with the GM—controlled by the challenger. The new user will have identifier *id* and the challenger adds *id* to $\mathcal{U}_C$.
- CorruptU(id). The adversary can request to corrupt user with identifier $id \in \mathcal{U}_H$. The challenger looks up the user's private information and credential and gives them to the adversary. It also adds *id* to $\mathcal{U}_C$ and removes *id* from $\mathcal{U}_H$.
- TransactSP(id, ϵ, τ). The adversary can make TransactSP(id, ϵ, τ) queries to request that a user with $id \in \mathcal{U}_H$ (the adversary can simulate this query for corrupted users) runs the PerformTransaction(τ) protocol for epoch ϵ where the adversary acts as SP.[9] The adversary receives all the information that the SP would normally receive, including the transaction record t.

9 Allowing the adversary to select the ϵ gives the adversary slightly more power, in an actual system time does not run backwards.

- VoteToLink(j, t). The adversary makes a VoteToLink(j, t) query to request the decryption share ψ_j from moderator $j \notin \mathcal{C}$ on transaction t. In response, the challenger runs VoteToLink(δ_j, t) on behalf of moderator j (which has decryption key share δ_j) and returns the result to the adversary.

CHALLENGE PHASE Eventually the adversary will select two users with identifiers $id_0, id_1 \in \mathcal{U}_H$, a transaction τ and an epoch ϵ such that the users id_0 and id_1 are unlinked in epoch ϵ. More precisely, the adversary should not have made any VoteToLink queries on transaction records t produced by either user id_0 or id_1 in epoch ϵ. The challenger picks a bit $b \in_R \{0, 1\}$ and acts as if the adversary called TransactSP(id_b, ϵ, τ). Let t^* be the corresponding transaction record.

RESTRICTED QUERY PHASE After the challenge query the adversary can continue to make queries as before, with the following restrictions. It is not allowed to call CorruptU on the users id_0, id_1, nor is it allowed to make VoteToLink queries involving any transaction record t produced by either id_0 or id_1 in epoch ϵ. (In particular, the adversary is not allowed to call VoteToLink on the challenge transaction t^*. The adversary can already create $k - 1$ decryption shares because of the corrupted moderators it controls.)

OUTPUT PHASE Eventually the adversary will output a guess b' of bit b. The adversary wins if $b = b'$.

At any point in time the adversary can run Link (because anyone can run this algorithm). The advantage of adversary $\mathcal{A}$ in this user anonymity game is given by $\mathsf{Adv}^{\mathrm{UA}}_{\mathcal{A}}(1^\ell) = 2\left|\Pr[b = b'] - \frac{1}{2}\right|$, where the probability is over the random bits of the challenger and the adversary. We say the vote-to-link scheme has *user anonymity* if $\mathsf{Adv}^{\mathrm{UA}}_{\mathcal{A}}(1^\ell)$ is negligible for every PPT adversary $\mathcal{A}$.

THEOREM 4.10 *The vote-to-link scheme has user anonymity in the random oracle model for H provided that the TDH2' is CCA secure and the DBDH-3b (see Definition 2.7) assumption holds.*

Proof. This proof consists of two steps. One, because of the CCA security of the TDH2' encryption scheme, the adversary does not learn anything about the plaintext of the encrypted linking token T. Hence, it does not learn anything about the linking token. In fact, we can replace the real encrypted linking tokens for the challenge users by random ones without the adversary noticing. We use the random oracle to simulate the proof π. Two, because the linking token is unknown, the DBDH-3b assumption and a proof similar to the one by Nakanishi and Funabiki [124]

ensures that the adversary cannot link users through the auxiliary information either. We will now give the full details.

We first prove security for a simpler version of the user anonymity game in which the adversary outputs the challenge users id_0, id_1 and the challenge epoch ϵ^* during the setup phase. Let this user anonymity game be G_0. Define the game G_1 as the game similar to G_0 in which the encrypted linking token T in the response to the transaction queries $\mathsf{TransactSP}(id_i, \epsilon^*, \tau)$ for users id_0 and id_1 in epoch ϵ^*, and in the challenge query have been replaced by the random ciphertext $T = \mathsf{TDH.Enc}(w, m, \tau)$, where $m \in_R \mathbb{G}_1$. Because of the CCA security of the TDH2' scheme (we use the real-or-random variant) games G_0 and G_1 cannot be distinguished.

More formally, suppose we have an adversary $\mathcal{A}_0$ that can distinguish game G_0 from G_1. We now show how to construct an adversary $\mathcal{B}_0$ that breaks the CCA security of the TDH2' scheme. Adversary $\mathcal{B}_0$ controls $\mathcal{A}_0$.

During the *setup phase* adversary $\mathcal{A}_0$ outputs the threshold k, the number n of moderators, a set $\mathcal{C}$ of $k-1$ adversaries it wants to corrupt and the challenge users id_0, id_1 and challenge epoch ϵ^*. Adversary $\mathcal{B}_0$ runs $\mathsf{SetupGM}$ to set up the group manager. It controls all the keys. Next, instead of running $\mathsf{SetupModerators}$, it relays $\mathcal{A}_0$'s choice of k, n and $\mathcal{C}$ to its challenger during the setup phase of the real-or-random CCA security game. Adversary $\mathcal{B}_0$ obtains the decryption keys δ_i for moderators $i \in \mathcal{C}$, which it relays to $\mathcal{A}_0$. This completes the setup phase.

During the *query phase* adversary $\mathcal{B}_0$ answers AddU, JoinU, and $\mathsf{CorruptU}$, queries honestly (it has all the information required to do so). On a query $\mathsf{TransactSP}(id, \epsilon, \tau)$ it proceeds as follows. If $id = id_i, i \in \{0,1\}$ and $\epsilon = \epsilon^*$, it replaces the encrypted linking token T by calling its encryption oracle on the linking token r and label τ. It simulates the proof of knowledge π. In all other cases, it answers the $\mathsf{TransactSP}$ query honestly.

To answer a $\mathsf{VoteToLink}$ query for moderator j and transaction record $t = (\tau, T, t_1, t_2, \epsilon, \pi)$ it runs $\mathsf{VoteToLink}$ as the moderator would, except for the $\mathsf{TDH.Dec}$ call. It uses its oracle to request the decryption share ψ_j of T from moderator j and returns ψ_j. If $\mathcal{A}_0$ does make a query for a transaction record t resulting from a $\mathsf{TransactSP}$ query on user id_0 or id_1 in epoch ϵ^* then $\mathcal{A}_0$ loses, and $\mathcal{B}_0$ aborts.

During the *challenge phase* adversary $\mathcal{A}_0$ outputs a transaction τ that it wants to be performed by either id_0 or id_1, in epoch ϵ^*. Adversary $\mathcal{B}_0$ picks a bit $\beta \in_R \{0,1\}$ and answers the challenge query as if the adversary made a $\mathsf{TransactSP}(id_\beta, \epsilon^*, \tau)$ query (that is, it uses its encryption oracle to create the encrypted linking token T).

During the *restricted query phase* adversary $\mathcal{A}_0$ continues to make queries as in the *query phase*. Adversary $\mathcal{B}_0$ answers them as before.

Finally, $\mathcal{A}_0$ outputs a guess $b' \in \{0,1\}$ for the game $G_{b'}$ that it believes it just played. If $\mathcal{B}_0$ had access to an encryption oracle producing real ciphertexts, then it perfectly simulated game G_0. If it had access to a random oracle, it perfectly simulated game G_1. So, if $\mathcal{A}_0$ outputs $b' = 0$, adversary $\mathcal{B}_0$ outputs 1 (as a guess that its oracle outputs real ciphertexts) and otherwise outputs 0. Any advantage that $\mathcal{A}_0$ has in distinguishing G_0 from G_1 results in the same advantage for $\mathcal{B}_0$ in distinguishing the real oracle from the random oracle in the TDH2' CCA security game. Since TDH2' is CCA secure, adversary $\mathcal{A}_0$ cannot exist.

We now continue from the situation in G_1, i.e., all the encrypted linking tokens of the challenge users id_0 and id_1 contain random messages. As such, the adversary never learns anything about the true value of the linking tokens. Define game G_2 as the game where we replace t_1 and t_2 in the transaction record t by random values in the challenge query. Any adversary $\mathcal{A}_1$ that can distinguish game G_1 from G_2 breaks the DBDH assumption.

More formally, suppose that we have an adversary $\mathcal{A}_1$ that can distinguish game G_1 from G_2. We now show how to construct an adversary $\mathcal{B}_1$ that can solve the DBDH-3b problem, i.e., $\mathcal{B}_1$ gets as input an instance $(g, A_1 = g^a, B_1 = g^b, h, A_2 = h^a, C_2 = h^c, Z = g_T^z) \in \mathbb{G}_1^3 \times \mathbb{G}_2^3 \times \mathbb{G}_T$ and it needs to decide whether is real, i.e., $z = abc$, or random, i.e., $z \in_R \mathbb{Z}_p$. This part of the proof is a straightforward translation of Nakanishi and Funabiki's proof of anonymity [124] to the Type 3 setting.

We model $H : \{0,1\}^* \to \mathbb{G}_1$ as a random oracle. Adversary $\mathcal{B}_1$ answers hash queries consistently. On input of a new hash query for epoch ϵ it picks an exponent $\delta_\epsilon \in_R \mathbb{Z}_p$. If $\epsilon = \epsilon^*$ it returns $g_{\epsilon^*} = B_1^{\delta_\epsilon} = g^{b\delta_\epsilon}$, otherwise, it returns $g_\epsilon = g^{\delta_\epsilon}$. These generators are still randomly distributed, as desired.

The *setup phase* proceeds as in the game. During the *query phase*, adversary $\mathcal{B}_1$ answers $\mathsf{AddU}(id)$, $\mathsf{JoinU}(id)$, and $\mathsf{CorruptU}(id)$ queries honestly as long as $id \neq id_0, id_1$. (Note that $\mathcal{A}_1$ never makes JoinU or CorruptU queries on id_0 and id_1 by assumption.) On a query $\mathsf{AddU}(id_i)$, $i \in \{0,1\}$, adversary $\mathcal{B}_1$ picks a randomizer $\xi_i \in_R \mathbb{Z}_p$ and proceeds as if user id_i's private key is $a\xi_i$. Since it does not know these user's private keys, it will simulate the proof of knowledge of knowing a credential over this key.

Adversary $\mathcal{B}_1$ answers VoteToLink queries honestly. On input of a query $\mathsf{TransactSP}(id, \epsilon, \tau)$ adversary $\mathcal{B}_1$ proceeds as follows. If $id \neq id_0, id_1$ then $\mathcal{B}_1$ uses that user's secret key to answer the query as usual. For user $id = id_i, i \in \{0,1\}$ it proceeds as follows. It picks $z \in_R \mathbb{Z}_p$

and sets:

$$t_1 = h^z$$
$$t_2 = \hat{e}(g_\epsilon, A_2)^{\xi_i z}$$

Now, $t_2 = \hat{e}(g_\epsilon, h^a)^{\xi_i z} = \hat{e}(g_\epsilon, h^z)^{a\xi_i}$ and hence is exactly as it should be. If $\epsilon = \epsilon^*$, $\mathcal{B}_1$ replaces the encrypted linking token with a random ciphertext $T = \mathsf{TDH.Enc}(w, m, \tau)$ for $m \in_R \mathbb{G}_1$. If $\epsilon \neq \epsilon^*$, the linking token r is given by $g_\epsilon^{a\xi_i} = (g^{\delta_\epsilon})^{a\xi_i} = (g^a)^{\delta_\epsilon \xi_i} = A_1^{\delta_\epsilon \xi_i}$, which it encrypts as normal. In both cases, $\mathcal{B}_1$ simulates the proof π.

During the *challenge phase* adversary $\mathcal{A}_0$ outputs a transaction τ that it wants to be performed by either id_0 or id_1, in epoch ϵ^*. Adversary $\mathcal{B}_0$ proceeds as follows. It picks a bit $b \in_R \{0,1\}$ and it sets $t_1 = C_2^z$ and $t_2 = Z^{z\delta_{\epsilon^*}\xi_b}$. If $Z = \hat{e}(g, h)^{abc}$ then

$$t_2 = Z^{z\delta_{\epsilon^*}\xi_b} = (\hat{e}(g, h)^{abc})^{z\delta_{\epsilon^*}\xi_b} = \hat{e}(g^{b\delta_{\epsilon^*}}, h^{cz})^{a\xi_b} = \hat{e}(g_{\epsilon^*}, t_1)^{a\xi_b},$$

as in game G_1. However, when $Z \in_R \mathbb{G}_T$ then t_1 and t_2 are completely random, as in G_2. Adversary $\mathcal{B}_1$ completes the transaction as before by simulating the proof and using a random encrypted linking token.

During the *restricted query phase* adversary $\mathcal{A}_0$ continues to make queries as in the *query phase*. Adversary $\mathcal{B}_1$ answers them as before.

Finally, $\mathcal{A}_1$ outputs a guess $b'' \in \{1,2\}$ for the game $G_{b''}$ that it believes it just played. Now, if $Z = \hat{e}(g, h)^{abc}$ then $\mathcal{B}_1$ precisely simulated G_1. Otherwise, it precisely simulated G_2. Hence, if $b'' = 1$, then $\mathcal{B}_1$ outputs 1 (to indicate that $z = abc$), and 0 otherwise. So, if $\mathcal{A}_1$ distuinguishes G_1 from G_2 it can be used to solve DBDH-3b instances. Hence, such a distinguisher cannot exist.

We have now reduced the game G_0 to game G_2 in which both the encrypted linking tokens T and the values t_1 and t_2 have been replaced by random values for the challenge user. All that remains is the anonymous credential. Clearly, if an adversary can distinguish the users id_0 and id_1 based on this information, than it breaks the anonymity of the anonymous credential scheme, so such an adversary also cannot exist. This proves that no adversary can win G_0.

We started out by making a slight simplification to the user anonymity game by asking the adversary to output the challenge users and the challenge epoch to obtain game G_0. In our final reduction we simply guess these values to simulate an adversary against G_0. If at any point this guess turns out to be wrong (for example because the real adversary makes VoteToLink queries on transactions from our challenge users corresponding to the challenge epochs, or if it outputs different values during the challenge phase) we simply abort and try again. This proves the final result. □

4.4.3 *A variant: identifying misbehaving users*

Above we presented a scheme that allowed the system to recover from bad actions by linking all the actions from a malicious user within an epoch. An anonymous blacklisting system would simultaneously ensure that a user cannot abuse the system again. In some scenarios, the threat of blacklisting might not be a sufficient deterrent, especially when the damage done by malicious actions cannot easily be repaired.

One area where such recovery from damage might be harder is the area of online review sites of commodities or services. On the one hand, it is in the reviewers' interest to remain anonymous so they can honestly evaluate the product without risk of retaliation by the manufacturer or the service provider. Especially in the hotel industry there have been several reports[10] of customers being fined after leaving critical reviews. On the other hand, as is also pointed out in these reports, the anonymity can also be abused, and once slanderous or false comments have been made, they tend to stick around.

For these scenarios we propose an extra deterrent: bad behavior leads to revealing the user's identity. We follow the same ideas as before, but now we construct a scheme where moderators can act on complaints by voting to deanonymize users, instead of just linking a users actions. This requires only a few changes to the protocol we presented above.

First, the group manager records the user's public key g^x and other information necessary to identify a user when the user joins. When the user performs a transaction she simply uses her public key as the linking token (i.e., $r = g^x$) and proves correctness as before. She omits the auxiliary values t_1 and t_2 because they are not necessary.

In the combination step the SP directly recovers the user's public key that was stored in the encrypted linking token, and can use this to identify the user, possibly with the help of additional information stored by the group manager.

Security and anonymity

The security and anonymity guarantees we derived in the previous section still hold for this variant. This variant does have a different anonymity guarantee: revoking a user identifies her—rather than link transactions—but does so only for a single transaction: no linking occurs.

10 http://edition.cnn.com/2014/08/04/travel/bad-hotel-review-fine-backlash/, last accessed February 17, 2017, and http://www.bbc.com/news/technology-30100973, last accessed February 17, 2017.

4.5 A VOTE-TO-LINK SCHEME WITH MODERATOR ANONYMITY

In this section, we present two schemes that allow the moderators to vote anonymously. We first present the gist of these schemes, then define moderator anonymity, and, finally, present both schemes in full.

4.5.1 *The idea*

In the scheme described in the previous section, moderators cannot vote anonymously. This is because Shamir's secret sharing scheme is used (as part of the threshold encryption scheme) to share the moderators' private key κ. That is, we create a polynomial f of degree $k-1$ such that $f(0) = \kappa$. Each moderator is then given a share $(i, f(i))$ as its decryption key. While partially decrypting typically hides $f(i)$, the index i (which identifies the moderator) is essential to recover the plaintext. To protect the moderators from retaliation we present two modified schemes that allow the moderators to vote anonymously.

The first scheme assumes that moderators trust each other, but do not trust other users of the system. The moderators generate new secret shares (of the same secret) for every transaction—since the secret stays the same, so does the moderators' public key. These shares are Shamir secret shares as before, but the moderator's index depends on the transaction. As a result, the shares look random for outsiders, however, the relation between indices and transactions is known to each of the moderators.

More precisely, the moderators non-interactively derive a secret injective function $\sigma : \{1, \ldots, n\} \to \mathbb{Z}_p$ (the injective function is transaction specific) and a secret-sharing polynomial f^σ sharing the fixed secret κ for each transaction. However, while every moderator knows the injective function σ, moderator i can evaluate the polynomial f^σ at only one point: $\sigma(i)$. Each moderator uses its single secret-share $(\sigma(i), f^\sigma(\sigma(i)))$ to partially decrypt the ciphertext corresponding to this transaction. This perfectly hides the identity of the voting moderators from the users. Since the transaction-specific function σ is known to all moderators, all moderators can identify the voting moderators.

In the case that moderators also do not trust each other we propose a second scheme where the user and the service provider interact. To achieve full anonymity for the moderators, the user and the service provider run a simple three step protocol. In essence, the user and the SP run a small mix network, ensuring that as long as the service provider and the user do not collude, no party can determine the identity of the voting moderators. To enable this protocol, each moderator has a public encryption key. The idea is as follows, see also Figure 4.2:

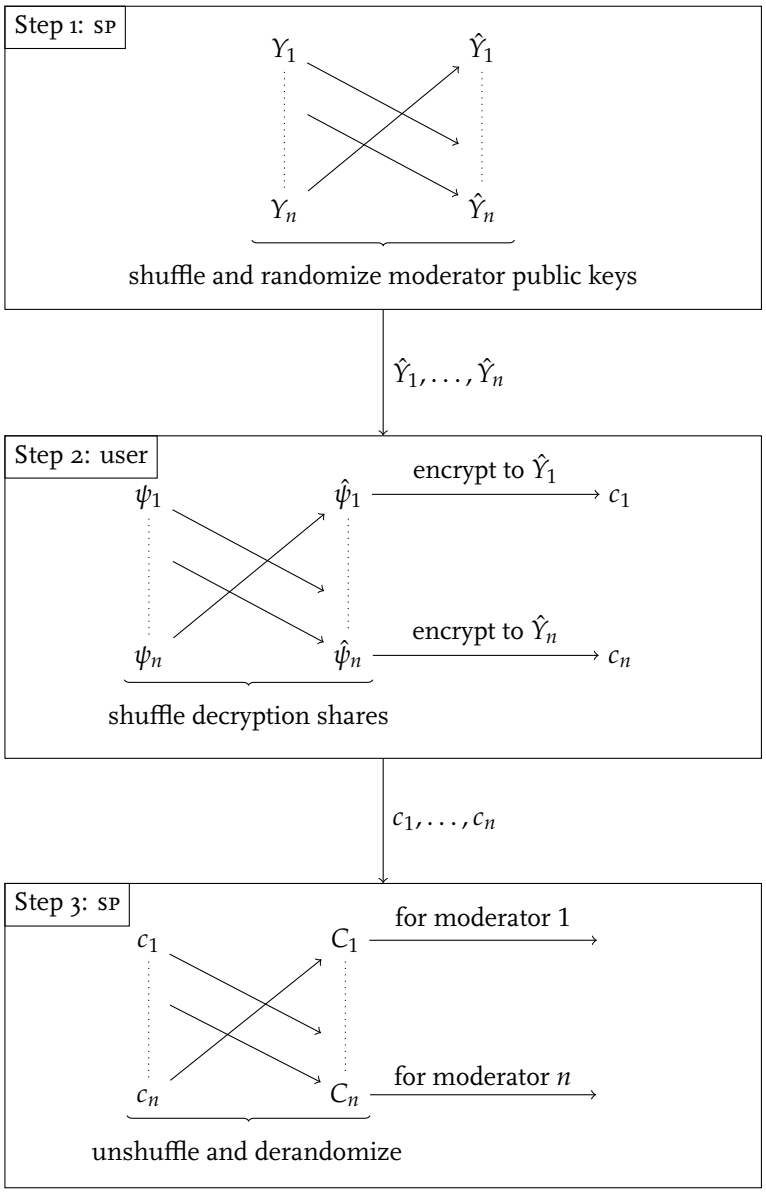


Figure 4.2: Conceptual representation of the mix network constructed by the user and the service provider to enable full moderator anonymous voting. The SP shuffles and randomizes the moderator public keys Y_i and sends them to the user. The user encrypts shuffled decryption shares ψ_i with these shuffled and randomized keys and sends the resulting ciphertexts back to the SP. The SP, finally, undoes its shuffling and randomization to obtain moderator-specific encrypted decryption shares.

1. The service provider shuffles and randomizes the moderators' public keys and sends them to the user.
2. The user creates a new threshold encryption key and encrypts its linking token with this new key. To facilitate decryption, she encrypts a shuffled decryption share for each of the shuffled and randomized public keys she received from the service provider.
3. The service provider unshuffles and derandomizes the ciphertexts containing the encrypted decryption shares and publishes them with the transaction record. Moderators will simply decrypt their ciphertexts to recover their decryption shares and cast their votes.

Assuming, for the moment, that both the user and the service provider are honest, it is easy to see that this gives anonymity for the voting moderators. First, because the service provider shuffles and randomizes the moderators' public keys, the user does not know to which moderator it gave which share. So, if a moderator reveals or uses a share, the user does not learn anything about the identity of the moderator. Second, because the user assigns random shares to each moderator, the service provider does not learn which moderator received which share. Hence, the service provider cannot recognize the voting moderators either.

In practice, we cannot assume that the user is honest—she might want to avoid consequences of bad behavior. Therefore, the user has to prove that she acted honestly. Similarly, we also cannot assume that the service provider is honest. In particular, the SP can deanonymize the user if it generated new keys for the moderators, hence the service provider too needs to prove that it shuffled and randomized the original public keys correctly.

4.5.2 *Outsider anonymity*

In the first vote-to-link system with anonymity for moderators, the identities of the voting moderators are only hidden from entities that are not part of the moderator group. Hence, we call this type of anonymity outsider anonymity.

To obtain outsider anonymity, we proceed in two steps to create a fresh secret-sharing polynomial for the same shared secret key κ. Each moderator is assigned a new index based on the transaction record (the secret, or in our case the decryption key, needs to remain the same to ensure that the corresponding public key is known when the user constructs the transaction record). The steps are as follows.

1. We modify Cramer et al.'s share conversion scheme (see Section 2.5.2) to derive a polynomial $f^{\sigma}(X)$ such that moderator

i can evaluate it at $\sigma(i)$, for some injective function σ—rather than a polynomial that moderator i evaluates at point i. In other words, $\sigma(i)$ is now moderator i's index corresponding to the share $f^\sigma(\sigma(i))$. The derived polynomial f^σ still shares the secret key κ.

2. The polynomial we derive in the previous step is not completely random (to achieve that directly we would also have to change the underlying additive secret shares, but we cannot easily do this without changing the moderators' public key). The reason is that σ determines how f^σ changes, and there are fewer injective functions than there are secret-sharing polynomials for a given secret. To correct this, we add a fully random zero-sharing polynomial $z(X)$ to $f^\sigma(X)$.

Recall that in the share conversion scheme by Cramer et al., see Section 2.5.2, the secret-sharing polynomial $f(X)$ is given by

$$f(X) = \sum_{\substack{A \subset \{1,\ldots,n\} \\ |A| = n-(k-1)}} r_A \cdot g_A(X),$$

where the degree $k-1$ polynomial $g_A(X)$ has zeros at points not in A, and $g_A(0) = 1$. The value r_A is known to all parties $i \in A$, so party i can evaluate $f(i)$.

In the following scheme, we change the zeros of the polynomials g_A based on an injective function σ. As a result, party i can evaluate f at $\sigma(i)$. However, to evaluate f, it needs to know the permuted g_As and hence the injective function σ. This limits the scheme to outsider anonymity.

Since the secret-sharing polynomial changes for each transaction, we cannot use the verification keys of the TDH2' scheme. We simply omit them. (The CCA security of the TDH2' scheme is not affected, as the ciphertext itself can still be validated.)

SCHEME 4.11 (Outsider-anonymous scheme) To obtain a vote-to-link scheme with outsider anonymity we start with our regular vote-to-link scheme (see Scheme 4.8) and modify it to allow outsider-anonymous voting.

- $\mathsf{SetupModerators}(1^\ell, n, k)$. To set up the moderators, generate additive shares $r_A \in_R \mathbb{Z}_p$ for each $A \subset [n]$ of cardinality $n-(k-1)$, set $\kappa = \sum_A r_A$ and generate $w = g^\kappa$. Next, generate a base coefficient $\zeta \in_R \mathbb{Z}_p$ for the zero-sharing polynomial and an injective function key $\theta \in_R \{0,1\}^\ell$, and choose a family of injective functions $\mathsf{inj} : \{0,1\}^\ell \to ([n] \to \mathbb{Z}_p)$ and a cryptographic hash function $H'' : \{0,1\}^* \to \{0,1\}^\ell$ (recall, the hash function

$H' : \{0,1\}^* \to \mathbb{Z}_p$ was already defined for the TDH2' scheme). Furthermore, as in TDH.KeyGen, generate a random generator $\bar{g} \in_R \mathbb{G}_1$ and publish the public parameters $(\mathbb{G}_1, g, \bar{g}, p)$ and the ciphertext space $\mathcal{C} = \mathbb{G}_1 \times \{0,1\}^* \times \mathbb{G}_1^2 \times \mathbb{Z}_p^2$. Finally, each moderator is given a voting key $\delta_i = (i, (r_A)_{A \ni i}, \zeta, \theta)$.

- VoteToLink(δ_i, t). Given a voting key $\delta_i = (i, (r_A)_{A \ni i}, \zeta, \theta)$ and a transaction record t the moderator proceeds as follows. First, let the injective function $\sigma = \mathsf{inj}(H''(\theta \parallel t)) : [n] \to \mathbb{Z}_p$ be based on the transaction record t. Then, it calculates:

$$f^\sigma(\sigma(i)) = \sum_{\substack{A \subset \{1,\ldots,n\} \\ |A| = n-(k-1)}} r_A \cdot g_A^\sigma(\sigma(i)),$$

where

$$g_A^\sigma(X) = \begin{cases} 1 & \text{if } X = 0 \\ 0 & \text{if } X \in \sigma(\{1,\ldots,n\} \setminus A). \end{cases}$$

Furthermore, moderator i calculates the zero-sharing polynomial

$$z(X) = \sum_{j=1}^{k-1} H'(\zeta \parallel j \parallel t) X^j.$$

and creates the decryption key $\delta_i' = (\sigma(i), f^\sigma(\sigma(i)) + z(\sigma(i))$. Finally, it runs VoteToLink(δ_i', t) of the original scheme.

The SetupModerators algorithm uses a trusted party to generate the initial additive secret shares. Alternatively, the moderators can run the AVSS$^+(n, k, \mathbb{G}_1, g, p)$ protocol (see Protocol 2.13 in Section 2.5.1) to jointly generate the required additive shares r_As and the public key w, without relying on a trusted party. Next, they jointly compute the coefficient ζ by adding n coefficients, each randomly chosen by a moderator.

ANONYMITY OF THE SCHEME. All we changed with respect to the original scheme is how the decryption keys are generated. Since the underlying additive shares still have the same threshold k, user anonymity is still guaranteed.

Next, we prove outsider moderator anonymity. We first define the full moderator anonymity game.

GAME 4.12 (Moderator anonymity) The *moderator anonymity* (MA) game is a modification of the user anonymity game (see Game 4.9), again between an adversary and a challenger. It proceeds in five phases:

SETUP PHASE At the start of the game the adversary informs the challenger about the number n of moderators and the threshold k it wants to use. The challenger runs SetupGM to set up the group manager and SetupModerators to set up the moderators. Finally, the challenger sets the set of corrupted moderators $\mathcal{C} = \emptyset$.

QUERY PHASE The adversary controls all users (hence, it cannot make AddU and CorruptU queries). It can make JoinU, TransactSP and VoteToLink queries as in the user anonymity game (note that the VoteToLink queries are restricted to uncorrupted moderators). Additionally, it can make the following query:

- CorruptM(i). The adversary request the corruption of a moderator i. The challenger gives all keys of moderator i to the adversary and adds i to $\mathcal{C}$.

CHALLENGE PHASE Eventually, the adversary requests votes on a valid transaction record t of its choosing. To do so, it sends t and two sets of moderators M_0 and M_1 to the challenger. The challenger verifies that the transaction record t is valid and new (i.e., it was not used earlier with a VoteToLink query), that the sets are of equal size, i.e., $|M_0| = |M_1|$, and that there are no corrupted moderators in the query sets, i.e., $(M_0 \cup M_1) \cap \mathcal{C} = \emptyset$; the challenger aborts otherwise. Finally, the challenger picks a bit $b \in \{0,1\}$ and then returns VoteToLink(i,t) for each moderator $i \in M_b$.

RESTRICTED QUERY PHASE The adversary can make JoinU as well as TransactSP queries as before. The VoteToLink(i,t) query can only be made on non-challenge transaction records.

OUTPUT PHASE Finally, the adversary outputs a guess b' for bit b. The adversary wins if $b' = b$.

The advantage of adversary $\mathcal{A}$ is given by $\mathsf{Adv}^{\mathrm{MA}}_{\mathcal{A}}(1^\ell) = 2\left|\Pr[b = b'] - \frac{1}{2}\right|$, where the probability is over the random bits of the challenger and the adversary.

DEFINITION 4.13 We say the voting scheme has *full moderator anonymity* if the advantage $\mathsf{Adv}^{\mathrm{MA}}_{\mathcal{A}}(1^\ell)$ is negligible for every PPT adversary $\mathcal{A}$.

DEFINITION 4.14 We say the voting scheme has *outsider moderator anonymity* if the advantage $\mathsf{Adv}^{\mathrm{MA}}_{\mathcal{A}}(1^\ell)$ is negligible for every PPT adversary $\mathcal{A}$ that makes no corruption queries CorruptM.

THEOREM 4.15 *The outsider-anonymous vote-to-link scheme offers outsider anonymity for moderators in the random oracle model for H'.*

Proof. We prove that the adversary cannot have an advantage in the outsider moderator-anonymity game. Instead of using the actual VoteToLink function, we define the stronger function where VoteToLink always returns the derived decryption key $\delta'_i = (\sigma(i), h(\sigma(i)))$, where $h = f^\sigma + z$ (recall that f^σ and z depend on the transaction τ).

It is easy to see that the challenge query provides no information to the adversary about the true value of b, the challenge bit used in the moderator anonymity game. We show this in two steps.

First, note that the zero-sharing polynomial z is uniquely determined by the hash function H'. Since the adversary does not know the secret input θ, we can patch H' in the random oracle model to produce any coefficients, and hence any polynomial z, we want. We can thus replace $h = f^\sigma + z$ by a truly randomly chosen secret-sharing polynomial $\tilde{h}$ of degree $k-1$ such that $\tilde{h}(0) = \kappa$ without the adversary detecting this.

Second, the final set of indices $\{\sigma(i) \mid i \in M_b\}$ does not leak any information about b either, because of the family of injective functions inj. Hence, we replace these by truly random indices and return a set of randomly chosen index-value pairs $(j, \tilde{h}(j))$ to the adversary. Clearly, these do not give the adversary any advantage in winning the moderator anonymity game. □

4.5.3 *Full anonymity for moderators*

We now present a mechanism that offers full anonymity to the moderators. The price is a reduction in efficiency for the user. In the outsider anonymity scheme, adding anonymity to the moderators does not cost the user anything. In the scheme we propose here, the user incurs an extra cost linear in the number of moderators.

The SP and the user follow the three steps that we outlined above and that are shown in Figure 4.2. However, since the user needs to prove that she shuffled the shares correctly, step 2 is somewhat more complicated. Recall that the user creates decryption shares, shuffles them, and encrypts them to the shuffled moderators' keys. To facilitate the proof of correctness, the user first encrypts the new decryption shares with a known ElGamal key, then shuffles these ciphertexts, and finally reencrypts these shuffled ciphertexts to the actual moderator keys. For each of these sub steps the user creates a proof that she did so correctly.

SCHEME 4.16 (Fully-anonymous scheme) To construct a vote-to-link scheme with full anonymity for moderators, we start with our regular vote-to-link scheme (see Scheme 4.8) and modify it to allow fully anonymous voting.

- SetupModerators$(1^\ell, n, k)$. Each moderator i generates an ElGa-

mal key-pair $(y_i, Y_i) = \mathsf{KeyGen}(\mathbb{G}, g, p)$ and publishes its public key Y_i. It privately stores its voting key $\delta_i = y_i$.

- $\mathsf{PerformTransaction}(\epsilon, \tau)$. The PerformTransaction protocol is run between a user and a service provider on input of a transaction τ. It proceeds in three steps.

  Step 1. The user registers the transaction τ, the service provider replies by sending a list of randomized and shuffled moderator public keys $\hat{Y}_1, \ldots, \hat{Y}_n$, i.e., the SP picks a permutation $\sigma_{SP} : [n] \to [n]$ and randomizers $\alpha_i \in_R \mathbb{Z}_p$, and uses these to randomize the ElGamal public keys accordingly (see Section 4.3.2):

  $$\hat{Y}_i = \mathsf{Randomize}(Y_{\sigma_{SP}(i)}, \alpha_i).$$

  The service provider then sends $\hat{Y}_1, \ldots, \hat{Y}_n$ to the user together with a proof

  $$\pi_{SP} = \mathrm{SPK}\{((\alpha_i)_{i\in[n]}, \sigma_{SP}) \ : \hat{Y}_i = \mathsf{Randomize}(Y_{\sigma_{SP}(i)}, \alpha_i)\}(\tau) \tag{4.2}$$

  that it randomized and shuffled actual moderator keys, see Section 4.5.4 for how to construct π_{SP}.

  Step 2. The user checks the proof π_{SP} and aborts if it is incorrect.[11] Then, she runs $\mathsf{TDH.KeyGen}(n, k, (\mathbb{G}_1, p, g))$ to create a fresh TDH2' private key κ and public key w with verification keys $w_i = g^{\kappa_i}$ corresponding to the decryption key shares (i, κ_i). Next, she creates her encrypted linking token $T = \mathsf{TDH.Enc}(r, w, \tau) = (c, L, u, v, e, d)$ and auxiliary information t_1, t_2, and proves that she did so correctly by calculating proof π as in Equation 4.1 in the original protocol, except for the fact that she uses a fresh key for the moderators.

  Finally, she encrypts the decryption key shares for each of the randomized public keys $\hat{Y}_1, \ldots, \hat{Y}_n$:

  1. Let $\psi_i = (i, u_i) = \mathsf{TDH.Dec}((i, \kappa_i), T)$ be the decryption shares. The user generates an ElGamal key-pair $(y', Y') = \mathsf{KeyGen}(\mathbb{G}, g, p)$ and encrypts the decryption share's components:[12]

     $$\hat{c}_i = \mathsf{Enc}(Y'; g^i, u_i)$$

     We encrypt g^i instead of the actual index i, since our ElGamal encryption scheme and the proofs we build upon them

11 In addition, as with the previous scheme, the user needs to verify the authenticity of the moderators' public keys.

12 Throughout the remainder of this scheme, when we operate on tuples of messages, we simply write $\mathsf{Enc}(Y; m_0, m_1)$ instead of the longer $(\mathsf{Enc}(Y, m_0), \mathsf{Enc}(Y, m_1))$.

require group elements. The user then proves that these are generated correctly, i.e., that $u_i = u^{\kappa_i}$, using the proof:

$$\pi_a = \mathrm{SPK}\Big\{(y', (\kappa_i)_{i\in[n]}) \;:\; Y' = g^{y'} \wedge \\ \forall i \in [n]\big[w_i = g^{\kappa_i} \wedge \hat{c}_i = \mathsf{Enc}(Y'; g^i, u^{\kappa_i})\big]\Big\}(\tau).$$

2. She chooses a random permutation $\sigma_U : [n] \to [n]$ and permutes the ciphertexts $\hat{c}_i$ according to σ_U:

$$\tilde{c}_i = \hat{c}_{\sigma_U(i)}\mathsf{Enc}(Y'; 1, 1),$$

and proves that she shuffled correctly:

$$\pi_b = \mathrm{SPK}\Big\{(\sigma_U) \;:\; \forall i \in [n]\big[\tilde{c}_i = \hat{c}_{\sigma_U(i)}\mathsf{Enc}(Y'; 1, 1)\big]\Big\}(\tau)$$

using, for example, Groth's verifiable shuffle protocol [76].

3. Finally, the user reencrypts the shuffled ciphertexts $\tilde{c}_i$ to the randomized and shuffled moderators' public keys $\hat{Y}_i$ to get ciphertexts c_i and proves that she did so correctly:

$$\pi_c = \mathrm{SPK}\Big\{((j_i, \kappa_i')_{i\in[n]}) \;:\; \forall i \in [n]\big[\\ \tilde{c}_i = \mathsf{Enc}(Y'; g^{j_i}, u^{\kappa_i'}) \wedge c_i = \mathsf{Enc}(\hat{Y}_i; g^{j_i}, u^{\kappa_i'})\ \big]\Big\}(\tau).$$

Here the user uses that she knows the content of all the ciphertexts.

The user, finally, sends the tuple $t_U = (T, t_1, t_2, w, Y', \pi, \pi_a, \pi_b, \pi_c, (\hat{c}_i, \tilde{c}_i, c_i, w_i)_{i\in[n]})$ to the SP.

Step 3. The service provider receives t_U and:

- it checks that the verification keys $w_1, \ldots, w_n$ and $w_0 := w$ are consistent using Lagrange interpolation. In particular, it sets $\mathcal{I} = \{0, \ldots, k-1\}$ and checks that, for all $i \in \{k, \ldots, n\}$,

$$w_i = \prod_{j\in\mathcal{I}} w_j^{\lambda_j^{\mathcal{I}}(i)};$$

- it verifies the correctness of the proofs π, π_a, π_b, and π_c; and
- it recovers ciphertext C_i for moderator i:

$$C_i = \mathsf{Derandomize}(c_{\sigma_{SP}^{-1}(i)}, \alpha_{\sigma_{SP}^{-1}(i)}), \tag{4.3}$$

and publishes these together with t_U as the transaction record.

- $\mathsf{VoteToLink}(\delta_i, \tau)$. Moderator i decrypts C_i using its voting key δ_i to recover the pair (g^j, u^{κ_j}) for some j. The moderator uses an algorithm such as baby-step giant-step to recover j (this only takes $O(\sqrt{n})$ time, see for example Katz and Lindell [93, Section 9.2.2] for an introduction). Then, it publishes (j, u^{κ_j}).

ANONYMITY OF THE SCHEME. First, we show anonymity for the user. The user sets up a completely new system for every transaction, but while t_U contains many values, these are either already present in the original protocol (like the public key w and verification keys w_i) or zero-knowledge proofs. So, we only need to concern ourselves with the ciphertexts. Of these, only moderator i can decrypt C_i. This ensures that every moderator receives at most one share, and hence guarantees user anonymity.

We already argued that the shuffling by both the user and the service provider ensures anonymity for the moderators as long as the user and the SP do not collude.

THEOREM 4.17 *The fully anonymous vote-to-link scheme (Scheme 4.16) offers full anonymity for moderators.*

4.5.4 *Shuffling randomized keys*

In the first step of the fully anonymous vote-to-link protocol, the SP shuffles randomized moderator keys. Just as for the user's proof of shuffling, the SP uses Groth's verifiable shuffle protocol [76] to construct the proof π_{SP}, however, the randomization of the moderators' public keys using $\mathsf{Randomize}$ complicates this proof slightly.

The trick to seeing why we can apply Groth's protocol is to reinterpret the $\mathsf{Randomize}$ function in a special ElGamal encryption scheme. In particular, let $\hat{g} \in_R \mathbb{G}_1$ be a random generator, and $g \in \mathbb{G}_1$ the corresponding public key. Then, the encryption of $m \in \mathbb{G}_1$ is $\mathsf{Enc}'(g, m) = (m \cdot g^\beta, \hat{g}^\beta)$, where $\beta \in_R \mathbb{Z}_p$. Note that when $\hat{g}$ is truly random the ciphertexts cannot be decrypted.

Now, we reinterpret the original public keys as ciphertexts in this scheme, i.e., Y_i becomes $H_i = \mathsf{Enc}'(g, Y_i) = (Y_i, 1)$ (i.e., we use $\beta = 0$). Similarly, we can reinterpret $\hat{Y}_i = \mathsf{Randomize}(Y_{\sigma_{SP}(i)}, \alpha_i)$ as

$$\hat{H}_i = H_{\sigma_{SP}(i)} \cdot \mathsf{Enc}'(g, 1) = (Y_{\sigma_{SP}(i)} \cdot g^{\alpha_i}, \hat{g}^{\alpha_i}),$$

where encryption $\mathsf{Enc}'(g, 1)$ uses the ephemeral key α_i. Hence, $\hat{H}_1, \ldots, \hat{H}_n$ are simply shuffled and randomized versions of the original ciphertexts $H_1, \ldots, H_n$. And this is exactly what we can prove using Groth's verifiable shuffle protocol.

4.5.5 *Probabilistic checking of moderator keys*

In step 3 of the full moderator anonymous vote-to-link scheme the service provider checks the validity of the verification keys $w_0, \ldots, w_n$. The naive version of this verification is slow: it requires $O(kn)$ group operations, whereas all other protocol steps are $O(n)$. In our implementation we replaced this check with a probabilistic check which only requires $O(n)$ group operations, thereby bringing it in line with the other operations.

The probabilistic checking algorithm uses the fact that if two polynomials are different, their evaluation at a random point is different as well with overwelming probability. This idea is not new. It is, for example, used to create short polynomial commitment schemes [92].

1. Create a cover of $\{0, \ldots, n\}$ consisting of sets of size k. For example, letting $r = \lfloor \frac{n}{k} \rfloor$, the $r+1$ sets $\mathcal{I}_0, \ldots, \mathcal{I}_r$ given by

$$\mathcal{I}_i = \{0 + ik, 1 + ik, \ldots, k - 1 + ik\} \text{ for } 0 \leq i < r$$
$$\mathcal{I}_r = \{n - (k-1), \ldots, n\}$$

   form such a cover. (We always use the same cover.)

2. Each set of verification keys $(w_j)_{j \in \mathcal{I}_i}$ defines its own degree $k-1$ polynomial f_i in the exponent, i.e., $w_j = g^{f_i(j)}$. (Such a polynomial always exists, since a degree $k-1$ polynomial has k degrees of freedom.) Pick a random element $\alpha \in_R \mathbb{Z}_p$ and evaluate the polynomials at this point

$$g^{f_i(\alpha)} = \prod_{j \in \mathcal{I}_i} w_j^{\lambda_j^{\mathcal{I}_i}(\alpha)}.$$

   To check that all f_i are equal and hence that the verification keys are correctly formed, test if $g^{f_0(\alpha)} = g^{f_i(\alpha)}$ for all $0 < i \leq r$.

Using this algorithm, we only do k group operations per covering subsets. Since we have $\lfloor \frac{n}{k} \rfloor + 1$ of these subsets, the complexity is indeed $O(n)$, as desired.

THEOREM 4.18 *The probabilistic verification key checking algorithm presented above is correct.*

Proof. It is easy to see that if the verification keys are generated honestly, the algorithm accepts. We now show that if the verification keys were not generated correctly, the algorithm detects this with overwhelming probability.

If the verification keys were not correctly generated, there exists an $i \in \{1, \ldots, r\}$ such that $f_0 \neq f_i$. Since $f_0 \neq f_i$, they agree at at most $k-1$ points (otherwise they would be equal). We choose α randomly, and after the user has fixed the verification keys. Hence, the probability that f_0 and f_i agree on α is $(k-1)/p$ which is negligible as required. □

4.6 VOTE-TO-LINK IN PRACTICE

In this section we explain how to choose parameters, and analyse the efficiency of our vote-to-link schemes.

4.6.1 *Choosing parameters*

The length of an epoch determines the utility of linking a user's action. Choosing a longer epoch ensures that a malicious user's actions become linkable over a longer time span. Hence, it is easier to locate all other bad actions by that same user. On the other hand, choosing longer epochs also increases the invasiveness of linking actions. Special care has to be taken when the decision to vote is very subjective. Since a vote to link effectively reduces a user's anonymity to an epoch-dependent pseudonym, the length of an epoch should be smaller in this case.

For our leading example, editing Wikipedia, we believe that an epoch of 24 hours strikes a good balance: the system can easily recover from bursty misbehavior, while inadvertent linking is not too damaging.

To reduce inadvertent linking to a minimum, moderators and the voting threshold must be carefully selected. Ideally, a system has only a few trusted moderators, in which case the threshold can be small as well. If the number of moderators cannot be kept low, the threshold should be set to a sizable percentage of the number of moderators to ensure that the 'bad apples' cannot influence the vote too much.

4.6.2 *Prototype implementation*

To evaluate the performance of our two schemes, we built and tested a proof-of-concept implementation[13] in C using the RELIC cryptographic library [8].[14] We implemented only the protocols' cryptographic parts, but the communication parts are easily added. We ran all experiments on a single core of an Intel i5-6200U running at 2.30 GHz. Most code can be optimized further, and is easily parallelizable.

We used BBS+ signatures [11] as anonymous credential scheme which

13 https://github.com/wouterl/vote-to-link

14 We set up RELIC to use an optimized 254-bit BN curve.

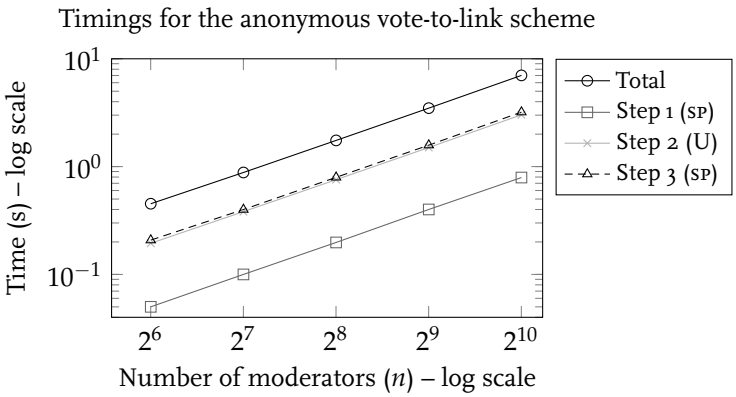


Figure 4.3: Running time of the total PerformTransaction protocol of the anonymous vote-to-revoke scheme for a threshold $k = 32$, as well as running times for the individual steps. As expected, the running time increases linearly in the size of n. (Communication cost is not taken into account.)

work in the type 3 setting which we also use for the linking tokens, see Section 2.7.

The non-anonymous vote-to-link scheme is very fast. The PerformTransaction protocol takes about 5 ms for both the user and the service provider. The size of a single transaction record is about 1 KiB. VoteToLink takes about 0.5 ms. After the SP receives k votes, it recovers the linking token in about 20 ms (for $k = 32$) and can then check about 2800 transactions per second against the recovered linking token.

The outsider anonymous version of the vote-to-link scheme is identical to the non-anonymous vote-to-link scheme as far as users and service providers are concerned. The performance for those parties is therefore identical. While we did not implement the outsider anonymous version, we expect a serious performance impact for the moderators. To cast a vote, the moderators must evaluate the polynomial f^σ, which requires $O(\binom{n}{k-1}k)$ field operations (the extra factor k is because the moderators also need to evaluate each of the polynomials g_A^σ). Since voting hopefully does not occur too frequently, a longer running time for VoteToLink might be acceptable. If the cost becomes too high, it is better to switch to the full moderator anonymous protocol.

Figure 4.3 shows the running times of the PerformTransaction protocol for the full moderator anonymous protocol. The $O(n)$ complexity of the proofs[15] causes a significant slow-down. However, a running time of 7 seconds for a large number of moderators like 1024 is still practi-

15 We replaced the verification of the w_i's in step 3 by the $O(n)$ probabilistic variant described in Section 4.5.5.

cal. The size of the transaction record increases linearly from 47 KiB for $n = 64$ to 732 KiB for $n = 1024$. Moderators can run VoteToLink in 5 ms for $n = 1024$.

4.7 RELATED WORK

Many systems have been developed that cover methods for dealing with misbehaving users. Perhaps one of the best known approaches are group signatures [45], which allow group members to anonymously sign messages on behalf of the group. We refer to, for example, Manulis et al. [117] for an overview. An essential aspect of group signatures is the ability to open or trace a signature to determine which group member created it. Typically, this power rests with either the group manager or a separate tracing manager.

Recent work has looked into creating group signatures where the tracing manager is distributed among many parties [67, 70, 116, 165]. Only when a specific number of parties agree can the signer of a signature be traced, i.e., identified. Another extension to group signatures is that of traceable signatures [94], which allows the group manager to produce tracing tokens that can be used to recognize signatures by the same user—similar to our vote-to-link scheme. The novelty of our scheme is the combination of the idea of a distributed tracing manager—the moderators in our case—and the linking of a user's actions within a *limited* time window (in the traceable signatures scheme [94] linking is global).

Many recent group signature schemes also offer revocation: giving a revocation manager the ability to block misbehaving users. In particular, we wish to highlight Nakanishi and Funabiki's scheme [124]. It offers backwards unlinkable revocation by using revocation tokens that are different for every epoch. Upon revocation the tokens for all future epochs are published. We employ this mechanism too. However, we instead use the revocation token for the current epoch to temporarily *link* a user's actions.

Also outside the realm of group signatures researchers looked into methods for deterring misbehavior. For example, the Nymble system seeks to block misbehaving Tor users, but uses a trusted party to do so [160]. The blacklistable anonymous credentials (BLAC) [159] instead blocks users without using a trusted party. In both cases the goal is to simply block the misbehaving user. But, as we have indicated, often other methods are required to recover from abuse. If this is the case, our scheme can be used on top of such an anonymous blocking scheme.

Anonymous reputation systems such as RepCoin [7] and AnonRep [164] offer another method to hold users accountable for their actions. The users' behavior is reflected in their reputation. While

RepCoin only supports positive feedback, AnonRep also allows actions to be downvoted. In both cases, the change in reputation only affects future actions. Hence, past misbehavior remains unidentified, contrary to our system.

Desmedt and Frankel were among the first to mention threshold cryptosystems and threshold encryptions [55]. In this chapter we created a verifiable variant of the more recent Shoup and Gennaro [147] scheme which is CCA secure. Of more recent interest is the scheme by Delerablée and Pointcheval [54] which allows encryptors to precisely select the threshold under which the message is to be encrypted. To make this possible, a trusted party—similar to the group manager in group signatures—assigns decryption keys to decryptors. This makes this scheme not applicable to our scenario because we do not want such an all-powerful party to exist. Another threshold encryption scheme is the one by Libert and Yung [106], which is secure against adaptive adversaries and is secure in the standard model. The downside is that they have to rely on a less common setting—a composite order bilinear group. Furthermore, the message is embedded in the target group, making it impossible to apply the linking mechanism which itself relies on pairings.

Either by design or as a result of the specific construction, these threshold cryptosystems identify the decryptors—the moderators in our scheme. In most schemes, this is a direct result of using Shamir's secret sharing scheme. There has been some research into anonymous secret sharing schemes, but none of these are applicable to our scenario. Blundo and Stinson's anonymity of secret sharing schemes [18] merely deals with the theoretical optimization of secret share sizes when you have to include the identity of the share-holder in the share. The shares themselves may still identify the share-holder, in fact, the Shamir share $(i, f(i))$ (rather than just $f(i)$) is 'anonymous' by their definition. Guillermo et al. present some truly anonymous schemes [77]—they prove that given the shares you cannot determine the shareholders from which these shares originated. However, the schemes they present are theoretical and lack efficient secret recovery algorithms. This makes it impossible to use them to construct an anonymous versions of threshold encryption schemes such as TDH2.

4.8 CONCLUSIONS

In this chapter we have introduced a new efficient vote-to-link scheme in which a group of moderators can decide to link a user's actions within an epoch when they detect abuse. We think that this scheme is especially useful in combination with an anonymous blocking scheme such

as BLAC: BLAC ensures that the user cannot continue her abuse, while the vote-to-link system ensures that abuse prior to the block can be revealed if the moderators decide this is necessary.

The concept of linking a user's actions within epochs also sets our solution apart from the open mechanism of group signatures, that always work on a single signature—and future ones in case of revocation—rather than identifying a set of signatures belonging to the same user, and traceable signatures, that do not limit the tracing capabilities.

In addition to this, we introduced two less efficient, but still practical variants that offers anonymity for the moderators in addition to anonymity for the users at the cost of efficiency. We think that this enables interesting scenarios, in particular if the moderators could otherwise experience retaliation because of their actions, or could be coerced into acting.

We do wonder if it is possible to build a non-interactive fully anonymous version of our scheme. In effect, this would give an anonymous threshold encryption scheme. Alternatively, it would be interesting to explore if pratical anonymous secret-sharing schemes can be built, and to use that to construct a vote-to-link scheme.

5 DISTRIBUTED ENCRYPTION

In this chapter, we revisit the distributed encryption scheme proposed by Hoepman and Galindo [83]. Distributed encryption is the dual of the usual threshold encryption schemes. In a threshold encryption scheme, decryptors can *decrypt* a ciphertext to produce decryption shares; the plaintext can only be recovered given at least k decryption shares (produced by different decryptors) corresponding to same ciphertext. In a distributed encryption scheme on the other hand, senders can *encrypt* plaintexts producing ciphertext shares; the plaintext can only be recovered given at least k ciphertext shares (produced by different senders) corresponding to the same plaintext.

As we saw in Chapter 3, distributed encryption is a primitive that can be used to implement revocable privacy. In particular, it can be used to implement the rule "if a person or an object generates more than k events at different locations, its identity should be revealed." To do so, sensors that register such events are senders in a distributed encryption scheme that encrypt the corresponding identity for every event that occurs. The scheme guarantees that the recipient of these ciphertexts can recover the identity only if it can combine k ciphertext shares, i.e., encryptions, of the same identity created by different senders.

Chapter 3 describes several use cases that can be solved using distributed encryption: the canvas cutters case, the average speed checking case, the sale of valuable objects case, and the child abuse case. In this chapter we will focus on the first two: the canvas cutters case and the average speed checking case.

Recall that the goal in the canvas cutters case is to detect cars that visit many different rest stops along highways, as these might correspond to the cars of criminals that rob the trucks that are parked at these rest stops. To detect these cars using distributed encryption, place an automatic number plate recognition (ANPR) system at each rest stop. For each car that visits the rest stop, the ANPR system creates a ciphertext share corresponding to the car's license plate. Only the ciphertexts corresponding to cars that visited at least k can be combined to reveal the car's license plate.

A distributed encryption scheme can also be used to enforce speed limits in a privacy-friendly manner as follows. Place an ANPR system at the start and end of a stretch of highway, like in the SPECS system [154]. Suppose that a car is speeding if it takes at most t seconds to traverse

this stretch. The ANPR systems generate distributed encryption ciphertext shares for every passing car. The system restarts, with fresh keys, after time t, so only when a car is speeding can it generate two shares before the keys are reset. To detect the speeding cars, the distributed encryption system uses a threshold of two. To reliably detect speeding cars, the detection system needs to run multiple, staggered instances of the distributed encryption scheme. The higher the number of parallel instances, the better the accuracy. See Section 5.6 for the details.

The second application is especially challenging: the time frames are short and the number of observations is high—during rush hours, sensors may see 30 cars per minute per lane. In this chapter we propose two new schemes that are much more efficiently in these situations than the distributed encryption scheme by Hoepman and Galindo [83].

In both applications, interaction between the cars and ANPR systems is infeasible: adding communication facilities to cars is costly. Therefore, distributed encryption schemes should be non-interactive to offer privacy in these situations. Techniques based on k-times anonymous credentials [31] and threshold encryption schemes—see Hoepman and Galindo [83] for a detailed discussion—are thus not appropriate. Also, when using a distributed encryption scheme, the senders can immediately encrypt their observations. Storing a plaintext copy, as would be needed for a secure multi-party computation between the senders, is therefore not necessary. We see this as an advantage. The non-interactivity does imply, however, that the senders have to be trusted to not store plaintext copies and to not frame users. See Section 3.1.4 for a broader discussion of interactive versus non-interactive sensors.

The first contribution of this chapter is a simpler distributed encryption scheme that does not use pairings, and satisfies stronger security requirements than the original scheme by Hoepman and Galindo [83]. We present this scheme in Section 5.3. We extend it with a non-trivial key-evolution method [66] to forward-securely [89] generate as many keys as necessary while keeping the key-size constant, see Section 5.4. The ability to restart the system with fresh keys is important in almost all applications, including the speed-limiting example, because it ensures that only shares generated within the same time frame can be combined. Hoepman and Galindo's original solution requires that the keys for every future time frame are generated in advance, and therefore scales poorly.

The second contribution is a batched distributed encryption scheme. It addresses the issue of inefficiency in traditional distributed encryption schemes in practice. The cost of recovering all encrypted messages from a set of ciphertexts shares is exponential: the only option is to try all possible combinations of shares. Our batched solution, which we

present in Section 5.5, is much more efficient, at the cost of increased storage requirements. The amount of storage is linear in the number of possible messages. Hence this solution is feasible only if the domain is small, as is the case for license plates. Nevertheless, we believe this to be a worthwhile trade-off.

Our third and final contribution is a prototype implementation of our schemes. In Section 5.6 we analyze the performance of our schemes, suggest additions to our schemes that may be useful in practice and present conclusions.

5.1 THE IDEA

The structure of our new distributed encryption scheme is as follows. Senders are given Shamir secret shares of the value 1. To encrypt a message into a ciphertext share, senders first encode this message into a group element, and then raise this message-specific group element to the power of their secret share. Given enough of these ciphertext shares, the combiner can use Lagrange interpolation to remove the secret shares from the exponent thus recovering recover the message-specific group element, and thereby the message itself.

In more detail, let $\mathbb{G}$ be a cyclic group of prime order p, such that the DDH problem is hard in $\mathbb{G}$. The protocol uses an injection $\chi : \{0,1\}^{\ell_m} \to \mathbb{G}$ to encode messages into group elements. The function χ^{-1} is the inverse of χ, i.e., $\chi^{-1}(\chi(m)) = m$ for all $m \in \{0,1\}^{\ell_m}$. This mapping is redundant, i.e., with high probability a random group element $g \in \mathbb{G}$ has no inverse under χ^{-1}. We construct this map in the next section.

Every sender i is given a secret share $s_i \in \mathbb{Z}_p$, corresponding to a k-out-of-n Shamir secret sharing of a *publicly known* value, 1. Let f be the corresponding degree $k-1$ secret-sharing polynomial, i.e., $f(0) = 1$ and $s_i = f(i)$. Sender i produces a ciphertext share for message m as follows. First, it encodes the message into a generator $\chi(m) \in \mathbb{G}$. Then, it uses its secret share to produce the ciphertext share $c_i = \chi(m)^{s_i}$. Given enough of these shares for the same message, the exponents can be removed, and the original encoded message can be recovered. More precisely, consider a set $\{c_{i_1}, \ldots, c_{i_k}\}$ of shares with $\mathcal{I} = \{i_1, \ldots, i_k\}$ the set of indices, then there exist Lagrange coefficients $\lambda_{i_1}^{\mathcal{I}}, \ldots, \lambda_{i_k}^{\mathcal{I}}$ such that $\sum_{i \in \mathcal{I}} \lambda_i^{\mathcal{I}} s_i = f(0) = 1$. So, we can calculate

$$\alpha = \prod_{i \in \mathcal{I}} c_i^{\lambda_i^{\mathcal{I}}} = \chi(m)^{\sum_{i \in \mathcal{I}} s_i \lambda_i^{\mathcal{I}}} = \chi(m)^{f(0)} = \chi(m).$$

As a result, $m = \chi^{-1}(\alpha)$. If the shares do not belong to the same message, the resulting encoding will, with high probability, not be an en-

coding of a message, and therefore fail to decode using χ^{-1}.

If χ is not redundant, the scheme is insecure. Let $m = \chi^{-1}(g)$ and $\overline{m} = \chi^{-1}(g^2)$. Then, given a share $c_i = \chi(m)^{s_i} = g^{s_i}$, it is trivial to make a share $c_i' = \chi(\overline{m})^{s_i} = g^{2s_i} = (g^{s_i})^2$ for $\overline{m}$ without help of the sender. Effectively, this allows the attacker to generate ciphertext shares for messages that it never queried. Hence, if the attacker is given fewer than k shares of a message $\bar{m}$ it can still decrypt it using the extra shares it constructed. This breaks the security of the scheme (see Section 5.3.2 for the full definition of security).

5.2 PRELIMINARIES

To prove security of our schemes we have to assume the hardness of the DDH problem, see Section 2.4 for its definition. Additionally, we model most hash functions in the random oracle model.

5.2.1 *A redundant injective map*

We now describe how to construct the map χ, which maps messages onto group elements in a redundant manner, as described in the previous section. First, we formalize what we mean by such a map.

DEFINITION 5.1 We call a map $\psi : A \to B$, with partial inverse $\psi^{-1} : B \to A \cup \{\perp\}$ a *redundant injective map* with security parameter ℓ_H if it satisfies the following properties:

COMPUTABLE The functions ψ and ψ^{-1} are efficiently computable. The function ψ may fail with negligible probability.

REVERSIBLE For all $a \in A$ we have $\psi^{-1}(\psi(a)) = a$.

REDUNDANT The probability $\Pr[\psi^{-1}(b) \neq \perp : b \in_R B]$ is negligible.

The redundancy ensures that when we combine ciphertext shares belonging to different messages we can detect this because ψ^{-1} returns $\perp$ (note, this requires $|B|$ to be at least $2^{\ell_H}|A|$). The redundancy requirement only guarantees that randomly chosen elements in B will not have an inverse with high probability. It is easy to adversarially choose $b \in B$ that does have an inverse (simply mapping an element from A first will guarantee this).

To prevent the attack at the end of Section 5.1, it should not be possible for an adversary to construct special group elements and find the messages that map to these group elements. In our security proof later on, we use the following stronger programmability requirement, which basically states that in the random oracle model we can change the redundant injective map to send messages queried by the adversary to

arbitrary group elements. This property implies that the attack at the end of Section 5.1 cannot work, as mapped group elements are essentially randomly chosen group elements.

DEFINITION 5.2 We say a redundant injective map is *programmable* in the random oracle model if we can adaptively ensure that $\psi(a) = b$ for any queried $a \in A$ and $b \in_R B$ with overwhelming probability, provided that $\psi(a)$ was not queried before. In particular, we can make these changes on the fly, as queries for a come in.

In our scheme, B must be a group. We give an example of a redundant injective map that redundantly maps strings to elements of an elliptic curve (sub)group $\mathbb{G}$ of prime order.[1] The map uses the same rejection sampling principle that is used to hash strings to group elements: we use the string to determine a candidate x-coordinate of a point on the curve. If this is a valid x, i.e., there is a corresponding y-coordinate such that (x, y) is a point in the group $\mathbb{G}$ then we accept, otherwise, we slightly change the message and try again.[2]

SCHEME 5.3 Our *redundant injective map* is defined by the algorithms RIM.gen, RIM.map and RIM.unmap—the last two match χ and χ^{-1}.

- RIM.gen$(1^{\ell_m}, 1^{\ell_H}, E, (\mathbb{G}, p))$. This method takes as input a message size ℓ_m, a security parameter ℓ_H, a description of an elliptic curve E, and the desired subgroup $\mathbb{G} \subset E$ of prime order p. Let $\mathbb{F}$ be the field over which E is defined, $y^2 = f(x)$ be the defining equation of the elliptic curve E, and h be the cofactor of $\mathbb{G}$ (that is, E has hp points). Let $\ell'_H = \ell_H + 2 + \lfloor \lg h \rfloor$ and let

$$\ell_c = \left\lfloor \lg \left(\frac{\ell_H}{\lg(2h) - \lg(2h-1)} \right) \right\rfloor + 1$$

  be the length of the counter. The algorithm aborts if it cannot encode strings of bit length $\ell_m + \ell_c + \ell'_H$ onto elements of $\mathbb{F}$.[3] The algorithm publishes two cryptographic hash functions $H_1 : \{0,1\}^{\ell_m + \ell_c} \to \{0,1\}^{\ell'_H}$ and $H_2 : \{0,1\}^{\ell'_H} \to \{0,1\}^{\ell_m + \ell_c}$, and

1 While this redundant injective map maps strings to group elements on an elliptic curve, a very similar method works to map strings to elements of the quadratic residues (simply check if the candidate point is a square).

2 In the paper on which this chapter is based, we used the Elligator map [15] to map strings (with redundancy already added) to group elements. Unfortunately, Elligator maps strings to a subset of the entire curve, rather than the prime-order subgroup that we work in throughout this thesis (moreover, in this chapter it is essential that this group is of prime order so that our secret sharing works). While we could solve this by rejection sampling too, we opted to use the rejection sampling method directly. This has the added benefit that implementing this map is easier than implementing the Elligator map.

3 If $\mathbb{F} = \mathbb{Z}_q$ with q a prime, then this means that q needs to be at least $\ell_m + \ell_c + \ell'_H$ bits.

the sizes ℓ'_H and ℓ_c. Finally, the mapping maps strings $\{0,1\}^{\ell_m}$ into $E' \subset \mathbb{G}$ such that

$$E' = \{(x,y) \mid x \text{ is } \ell_m + \ell_c + \ell'_H \text{ bits and } y \text{ is 'positive'}\} \cap \mathbb{G}.^4$$

- RIM.map(m). This function takes as input a message $m \in \{0,1\}^{\ell_m}$. It initializes counter ctr to zero, the counter is represented by a fixed-width string of ℓ_c bits, and proceeds by incrementing ctr until the following steps do not reject the counter:
  1. First, it creates an augmented message $m' = m \parallel \text{ctr} \in \{0,1\}^{\ell_m+\ell_c}$.
  2. It creates a candidate x coordinate $x = (m' \oplus H_2(r)) \parallel r \in \mathbb{F}$ where $r = H_1(m')$.[5]
  3. It calculates $y = \sqrt{f(x)}$, or rejects this counter if no such y exists. The algorithm always chooses the same 'positive' square root.
  4. Finally, it rejects this counter if $(x,y) \notin \mathbb{G}$.

  The algorithm returns the point $(x,y) \in \mathbb{G}$.
- RIM.unmap(c). Given a group element $c = (x,y) \in \mathbb{G}$ this function first parses x as $b_1 \parallel b_2$ such that $b_1 \in \{0,1\}^{\ell_m+\ell_c}$ and $b_2 \in \{0,1\}^{\ell'_H}$, and returns $\perp$ otherwise. Else, it sets $m' = b_1 \oplus H_2(b_2)$. If $H_1(m') = b_2$ it returns the leading ℓ_m bits of m (chopping off the counter), else it returns $\perp$.

THEOREM 5.4 *The map from Definition 5.1 is a programmable redundant injective map in the random oracle model for H_1 and H_2, where ψ = RIM.map, ψ^{-1} = RIM.unmap, $A = \{0,1\}^{\ell_m}$, and $B = E'$*

Proof. We first show that computability is satisfied. To do so, we need to show that the procedure for RIM.map runs in (expected) polynomial time. First, note that by construction of the sample point x using the hash function, it is uniformly random for each new counter ctr. Second, step 3 succeeds with probability negligibly close to ½ due to the Hasse-Weil bound (see Silverman and Tate [148] for an introduction). Third, the cofactor h is generally small—1, 2, 4 and 8 are common values—hence step 4 succeeds with probability $1/h$. Therefore, the algorithm needs on average $2h$ steps to find a successful mapping, which is polynomial (at least for elliptic curves, where h is small). Therefore, the map is computable. Reversibility is clearly satisfied.

4 We require that y is positive to make square roots unique. In particular, if $\mathbb{F} = \mathbb{Z}_q$ then it suffices to take square roots from $0, \ldots, (q-1)/2$.

5 Strictly speaking, we need to encode the string $(m' \oplus H_2(r)) \parallel r$ into an element in $\mathbb{F}$, but for clarity we omit this as such encoding is generally straightforward.

The construction of ℓ_c ensures that the counter can be large enough so that we almost always find a mapping. The probability that a counter value is not accepted is $P_F = (2h-1)/(2h)$. By construction, the algorithm can try at least

$$t = \frac{\ell_H}{\lg(2h) - \lg(2h-1)} = \frac{\ell_H}{\lg \frac{2h}{2h-1}} = \frac{\ell_H}{\lg P_F^{-1}}$$

different points. Then, the probability that all t possible counters fail is negligible, in particular,

$$P_F^t = P_F^{\frac{\ell_H}{\lg P_F^{-1}}} = \left(P_F^{\frac{1}{\lg P_F}}\right)^{-\ell_H} = 2^{-\ell_H} = \mathsf{negl}(\ell_H).$$

As an example, with a cofactor 8 we need to try, on average, 16 times before finding a mapping. After making 1375 attempts, the probability of all attempts failing is less than 2^{-128}.

Next, we show that the map is redundant. For any $c = (x, y) \in_R \mathbb{G}$ such that $x = b_1 \parallel b_2$ (with $b_1 \in \{0,1\}^{\ell_m + \ell_c}$ and $b_2 \in \{0,1\}^{\ell'_H}$), x is drawn uniformly at random from a subset of $\{0,1\}^{\ell_m+\ell_c+\ell'_H}$ that is $2h$ times smaller. Therefore, since H_1 is a hash function, $H_1(b_1 \oplus H_2(b_2)) = b_2$ with probability at most $2h \cdot 2^{-\ell'_H} \leq 2^{-\ell_H}$, so the map is redundant.

Finally, we show that the map is programmable. To do so, we show that we can ensure that $\mathsf{RIM.map}(m) = g$ for any queried message $m \in \{0,1\}^{\ell_m}$ and $g \in_R E'$ with overwhelming probability, provided that $\mathsf{RIM.map}$ was not queried with m, i.e., that with H_1 was not queried with $m \parallel \mathsf{ctr}$ for any ctr.

Let $g = b_1 \parallel b_2$. In the following, we use the random oracles for H_1 and H_2 to ensure that $\mathsf{RIM.map}(m) = g$ for message m. The difficulty is to ensure that the distribution of ctr matches that of an actual mapping. We ensure this by first sampling ctr. To do so, run $\mathsf{RIM.map}(m)$ as normal until it finds a value of ctr for which the mapped point (x, y) is accepted. The first ctr iterations of this procedure (recall that ctr starts at zero), the point is rejected. For each rejection we have to make one query for H_1 and one for H_2. Act as if the user made these queries, and make these part of the oracles' records.

The final, successful mappings of H_1 and H_2 we do not include in the record. Instead, we set $H_1(m \parallel \mathsf{ctr}) = b_2$. Then, since b_2 is random, with overwhelming probability it was not queried before and we can set $H_2(b_2) = (m \parallel \mathsf{ctr}) \oplus b_1$. Since b_1 and b_2 are random, so is $(m \parallel \mathsf{ctr}) \oplus b_1$, therefore, the outputs are set to random values as required. □

The programmability in the random oracle model guarantees that the attack at the end of Section 5.1 does not work. Alternatively, we can

also directly use the pre-image resistance of H_1 to show that the attack cannot work. To effect the attack, the attacker constructs a special element $\overline{g} \in \mathbb{G}$ (this element is not random, hence the redundancy does not help) and now needs to find a message $\overline{m}$ such that $\mathsf{RIM.map}(\overline{m}) = \overline{g} = (\overline{x}, \overline{y})$. To do so, letting $\overline{x} = b_1 \parallel b_2$, the attack at first needs to find a message $m' \in \{0,1\}^{\ell_m + \ell_c}$ such that $m' \oplus H_2(H_1(m')) \parallel H_1(m') = b_1 \parallel b_2$. Hence, the attacker needs to find a pre-image m' of H_1 such that $H_1(m') = b_2$ (this is a necessary but not sufficient condition, since b_1 will likely still be incorrect, and moreover, m' needs to include a counter such that no smaller counters yield a successful mapping). Since H_1 is a cryptographic hash function, it is pre-image resistant, and the attack cannot work.

5.3 A NEW DISTRIBUTED ENCRYPTION SCHEME

In this section we present our new distributed encryption (DE) scheme, which we sketched in Section 5.1. Our new DE scheme directly creates shares of the message instead of shares of an identity-based decryption key that decrypts the message, as in Hoepman and Galindo's scheme [83].

The new distributed encryption scheme is simpler than Hoepman and Galindo's and no longer requires pairings. Furthermore, the new structure allows us to define a non-trivial key-evolution method, which seems impossible for the original scheme without compromising forward security. To better illustrate these differences, we recall Hoepman and Galindo's scheme before formally presenting our own.

The structure of this section is as follows. We first recall the syntax of a key-evolving distributed encryption scheme as defined by Hoepman and Galindo [83]; we then present their security definition, which we generalize, and their distributed encryption scheme. Finally, we describe our own distributed encryption scheme, and prove its security.

5.3.1 *Syntax*

The syntax of a key-evolving distributed encryption scheme, as defined by Hoepman and Galindo [83], is as follows (with the exception that we have made the safety requirement explicit).

SYNTAX 5.5 (Key-evolving distributed encryption [83]) A k-out-of-n *key-evolving distributed encryption* scheme with lifetime divided into s epochs, or (k, n, s)-KDE scheme, consists of four algorithms:

- $\mathsf{KDE.Gen}(1^\ell, k, n, s, 1^{\ell_m})$. The key generation algorithm takes as input a security parameter ℓ, a threshold k, the number n of

senders, the number s of epochs and a message size ℓ_m (in bits).[6] It generates an initial encryption key $S_{1,i}$ for each sender $i \in [n]$. It returns the initial encryption keys, the system parameters, and the message space $\mathcal{M}$.

- $\mathsf{KDE.UpdKey}(S_{\epsilon,i})$. The key update function KDE.UpdKey takes an encryption key $S_{\epsilon,i}$ as input and outputs the encryption key $S_{\epsilon+1,i}$ for the next epoch. This function aborts if $\epsilon + 1 > s$.
- $\mathsf{KDE.Enc}(S_{\epsilon,i}, m)$. Given an encryption key $S_{\epsilon,i}$ and a message m, this function returns a ciphertext share c.
- $\mathsf{KDE.Comb}(C)$. Given a set $C = \{c_1, \ldots, c_k\}$ consisting of k ciphertext shares, the function $\mathsf{KDE.Comb}(C)$ either returns a message m or $\perp$.

Every key-evolving distributed encryption scheme must satisfy the following correctness and safety requirements.

CORRECTNESS Create the encryption keys $S_{\epsilon,1}, \ldots, S_{\epsilon,n}$ by first running KDE.Gen and then repeatedly updating them using KDE.UpdKey to reach the required epoch ϵ. For all messages m and pairwise disjoint senders i_j we recover m, i.e., $\mathsf{KDE.Comb}(C) = m$, if $C = \{\mathsf{KDE.Enc}(S_{\epsilon,i_1}, m), \ldots, \mathsf{KDE.Enc}(S_{\epsilon,i_k}, m)\}$.

SAFETY Generate encryption keys $S_{\epsilon,1}, \ldots, S_{\epsilon,n}$ as for correctness. If $C = \{\mathsf{KDE.Enc}(S_{\epsilon,i_1}, m_{i_1}), \ldots, \mathsf{KDE.Enc}(S_{\epsilon,i_k}, m_{i_k})\}$ with not all messages $m_{i_1}, \ldots, m_{i_k}$ equal, then with overwhelming probability $\mathsf{KDE.Comb}(C) = \perp$.

To make the system secure in practice, senders need to get their keys in a secure manner, and, to ensure forward security, senders have to destroy the old keys after updating them.

A distributed encryption scheme is a special case of a key-evolving distributed encryption scheme.

DEFINITION 5.6 (Distributed encryption) A k-out-of-n *distributed encryption* scheme, or (k, n)-DE scheme, is a $(k, n, 1)$-KDE scheme given by the algorithms DE.Gen, DE.Enc, and DE.Comb such that

$$\begin{aligned} \mathsf{DE.Gen}(1^\ell, k, n, 1^{\ell_m}) &= \mathsf{KDE.Gen}(1^\ell, k, n, 1, 1^{\ell_m}), \\ \mathsf{DE.Enc}(S_i, m) &= \mathsf{KDE.Enc}(S_{1,i}, m), \text{and} \\ \mathsf{DE.Comb}(C) &= \mathsf{KDE.Comb}(C). \end{aligned}$$

6 In the original description the message size was implicit.

5.3.2 *Security definition*

We define the forward security of a key-evolving distributed encryption scheme by recalling its security game. We present the security game by Hoepman and Galindo [83] in a slightly more general setting: messages that have been queried before may be used in the challenge phase, provided this does not lead to a trivial win for the adversary.

GAME 5.7 (KDE forward-security game) Consider a (k, n, s)-KDE key-evolving distributed encryption scheme with security parameter 1^ℓ given by the four algorithms KDE.Gen, KDE.UpdKey, KDE.Enc and KDE.Comb. Define the following forward-security game between a challenger and an adversary $\mathcal{A}$.

SETUP PHASE The challenger runs $\mathsf{KDE.Gen}(1^\ell, k, n, s, 1^{\ell_m})$ to obtain $(S_{1,1,}, \ldots, S_{1,n})$ and sends a description of the message space $\mathcal{M}$ and system parameters to the adversary.

FIND PHASE The challenger initializes the current epoch ϵ to 1, and the set of corrupted senders $I_{1,c}$ to the empty set. The adversary can issue the following three types of queries:

- $\mathsf{corrupt}(i)$. The adversary requests the corruption of sender i. This query is only allowed before any encryption query $\mathsf{enc}(i, m)$ has been made for the current epoch. If the query is allowed, the challenger sends $S_{\epsilon,i}$ to the adversary and it adds i to $I_{\epsilon,c}$.
- $\mathsf{enc}(i, m)$. The adversary can request the ciphertext share on $m \in \mathcal{M}$ produced by sender $i \in [n]$, $i \notin I_{\epsilon,c}$. The challenger sends $\mathsf{KDE.Enc}(S_{\epsilon,i}, m)$ to the adversary.
- $\mathsf{next}()$. On next-epoch queries $\mathsf{next}()$, the challenger updates the encryption keys of senders $i \in \{1, \ldots, n\} \setminus I_{\epsilon,c}$ by setting $S_{\epsilon+1,i} = \mathsf{KDE.UpdKey}(S_{\epsilon,i})$. The adversary is responsible for updating the keys of the other senders in $I_{\epsilon,c}$. Finally, the challenger sets $I_{\epsilon+1,c} = I_{\epsilon,c}$ and $\epsilon = \epsilon + 1$.

CHALLENGE PHASE The adversary $\mathcal{A}$ outputs a challenge epoch number $\epsilon^* < s$, indices $I_{nc} = \{i_1, \ldots, i_t\}$ corresponding to senders from which it wants to receive challenge ciphertexts and two equal length messages $m_0, m_1 \in \mathcal{M}$. Let r denote the cardinality of $I_{\epsilon^*,c}$ and Q_0 and Q_1 denote the senders at which messages m_0 and m_1 were queried respectively in epoch ϵ^*. The challenger aborts if the challenge is not *valid*, i.e., if one of the following conditions holds

- m_0 or m_1 was queried at a challenge sender, i.e., if $(Q_0 \cup Q_1) \cap I_{nc} \neq \emptyset$;

- a challenge sender was already corrupted, i.e., if $I_{nc} \cap I_{\epsilon^*,c} \neq \emptyset$; or
- too many shares are known to the adversary for either m_0 or m_1. This is the case if $\max(|Q_0 \cup I_{\epsilon^*,c}|, |Q_1 \cup I_{\epsilon^*,c}|) + |I_{nc}| \geq k$.

Finally, the challenger chooses $b \in_R \{0,1\}$ and returns a challenge ciphertext share $c_{\epsilon^*,i} = \mathsf{KDE.Enc}(S_{\epsilon^*,i}, m_b)$ for each $i \in I_{nc}$.

OUTPUT PHASE The adversary $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$. The adversary wins if $b = b'$.

The advantage of adversary $\mathcal{A}$ is given by $\mathsf{Adv}^{\text{KDE}}_{\mathcal{A}}(1^\ell) = 2\left|\Pr[b' = b] - \frac{1}{2}\right|$, where the probability is over the random bits of the challenger and the adversary. A KDE scheme is called *forward secure* if $\mathsf{Adv}^{\text{KDE}}_{\mathcal{A}}(1^\ell)$ is negligible for every PPT adversary $\mathcal{A}$.

DEFINITION 5.8 We say a (k,n)-DE scheme is *secure* if the corresponding $(k,n,1)$-KDE scheme is *forward secure*.

Hoepman and Galindo prove the security of their scheme in a more restricted model: they only consider static adversaries and the challenge messages must not have been queried in the challenge epoch.

DEFINITION 5.9 A *static and restricted* adversary announces, before the setup phase, which senders it will corrupt in what epoch, what the challenge epoch ϵ^* is, and which senders it will query in this challenge phase. In addition, the adversary is not allowed to request challenge messages m_0, m_1 that were queried in the challenge epoch.[7]

We prove that our scheme is secure in the unrestricted model of Game 5.7.

5.3.3 *Hoepman and Galindo's DE scheme*

We first recall the scheme proposed by Hoepman and Galindo [83]. Every ciphertext share for a message m contains an identity-based encryption (IBE) of the message corresponding to the Boneh-Franklin IBE scheme [22]. The message m itself is used as the identity. Normally, in an IBE scheme, there exist a private key generator that can generate the decryption key for a given identity. Instead of using a central authority, in Hoepman and Galindo's scheme, each sender immediately produces

7 There is no trivial reduction between the forward-security game with a static and restricted adversary and the forward-security game with a general adversary. While it is possible to guess the challenge epoch (and only incur a polynomial reduction in the success probability), there are $\binom{n}{r}$ possible sets of corrupted senders, which can be exponential in the inputs (since k is an upper bound for r).

a share of the IBE decryption key corresponding to the identity m. The private key generator is, as it were, distributed among the senders.

A distributed encryption ciphertext now consists of two parts: (1) the IBE encryption of the message m under the identity m, and (2) a share of the IBE decryption key for identity m. To combine k ciphertexts corresponding to the same message m, the combiner first recovers the IBE decryption key corresponding to the identity m and then uses this key to decrypt one of the IBE ciphertexts to recover the message.

To implement the IBE scheme, Hoepman and Galindo use a symmetric authenticated encryption (AE) primitive that uses $\mathbb{G}_T$, the target group of a bilinear map, as the key space.[8] Such a scheme is given by two algorithms (we omit the key-generation algorithm as we use $\mathbb{G}_T$ as the key space): AE.Enc and AE.Dec. The ciphertext c of a message m encrypted with a key $K \in \mathbb{G}_T$ is given by $c = \mathsf{AE.Enc}(K, m)$. The message is recovered by calling $\mathsf{AE.Dec}(K, c)$. If the ciphertext c does not correspond to the encryption of a message, AE.Dec outputs $\perp$.

More formally, Hoepman and Galindo's DE scheme works as follows.

SCHEME 5.10 (DE scheme [83]) Hoepman and Galindo's DE scheme uses an authenticated encryption scheme given by the algorithms AE.Enc and AE.Dec. The DE scheme is given by the following algorithms.

- $\mathsf{DE.Gen}(1^\ell, k, n, 1^{\ell_m})$. The DE.Gen algorithm takes as input a security parameter ℓ, a threshold k, the number n of senders and the message size ℓ_m. First, it generates a bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$, both of prime order p such that p is ℓ-bits, with generators g and h respectively, and a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that the DDH problem is hard in $\mathbb{G}_1$. Let $H' : \{0,1\}^* \to \mathbb{G}_1$ be a hash-function mapping into $\mathbb{G}_1$. Finally, it picks a authenticated encryption scheme with keys in $\mathbb{G}_T$ given by the algorithms AE.Enc, AE.Dec such that the message space is at least ℓ_m bits. It publishes a description of the groups, the bilinear map $\hat{e}$, the authenticated encryption scheme and the hash function H'.

  Next, it generates a master secret $e \in_R \mathbb{Z}_p$ and corresponding public key $E = h^e \in \mathbb{G}_2$. It creates a Shamir k-out-of-n secret sharing of the master secret e as follows. It picks $f_1, \ldots, f_{k-1} \in_R \mathbb{Z}_p$ and defines the $k-1$ degree polynomial $f(X) = e + \sum_{i=1}^{k-1} f_i X^i$, then $f(0) = e$. For sender i it creates a share $S_i = (i, s_i)$ where $s_i = f(i)$. It outputs S_i for each sender, and publishes the public key E.

8 This is a bit more subtle than Hoepman and Galindo led us to believe, as AE primitives typically use two keys of ℓ bits to obtain a security level of ℓ bits. Fortunately, the order of $\mathbb{G}_T$ is also ℓ bits, so we can compute a cryptographic hash of the element from $\mathbb{G}_T$ to obtain the keys for the AE scheme.

- $\mathsf{DE.Enc}(S_i, m)$. Given an encryption key $S_i = (i, s_i)$ and a message m, generate a randomizer $r \in_R \mathbb{Z}_p$, calculate the symmetric key $K = \hat{e}(H'(m), E)^r$ and create the ciphertext components

$$\sigma_i = H'(m)^{s_i}, \qquad \rho_i = h^r, \qquad \gamma_i = \mathsf{AE.Enc}(K, m),$$

  where (ρ_i, γ_i) forms the IBE encryption of m under the identity m, while σ_i is the decryption key share. Return the ciphertext share $c_i = (i, \sigma_i, \rho_i, \gamma_i)$.

- $\mathsf{DE.Comb}(C)$. Let $C = \{c_{i_1}, \ldots, c_{i_k}\}$. Parse each share c_{i_j} as $(i_j, \sigma_{i_j}, \rho_{i_j}, \gamma_{i_j})$. Construct[9] $\mathcal{I} = \{i_1, \ldots, i_k\}$, let $\Psi = \prod_{i \in \mathcal{I}} (\sigma_i)^{\lambda_i^{\mathcal{I}}}$ and recover the key $K = \hat{e}(\Psi, \rho_{i_1})$. Finally, return $\mathsf{AE.Dec}(K, \gamma_{i_1})$.

It is easy to see that this scheme is correct. Let $\mathcal{I} \subset [n]$ be a set of k senders and $c_i = \mathsf{DE.Enc}(S_i, m)$ their ciphertext shares for a message m. Then,

$$\Psi = \prod_{i \in \mathcal{I}} (\sigma_i)^{\lambda_i^{\mathcal{I}}} = \prod_{i \in \mathcal{I}} H'(m)^{s_i \lambda_i^{\mathcal{I}}} = H'(m)^{\sum_{i \in \mathcal{I}} s_i \lambda_i^{\mathcal{I}}} = H'(m)^e,$$

so

$$K = \hat{e}(\Psi, \rho_i) = \hat{e}(H'(m)^e, h^r) = \hat{e}(H'(m), h^e)^r = \hat{e}(H'(m), E)^r$$

is indeed the symmetric key used by sender $i \in \mathcal{I}$, and hence $m = \mathsf{AE.Dec}(K, \gamma_i)$ as desired. Safety follows form to observation that if not all ciphertext shares belong to the same message m, then $\Psi \in_R \mathbb{G}_1$. Hence, by the security of the AE scheme, $\mathsf{AE.Dec}(K, \gamma_i)$ outputs $\perp$ with overwhelming probability. Otherwise, it would be too easy to generate valid ciphertexts for the random key Ψ.

Note, though, that this scheme does not offer protection against active adversaries. It is trivial to create a set of ciphertexts that decrypt to any message (simply generate a known sharing of a random secret and run the normal combine protocol). Furthermore, if an adversary knows $k - 1$ ciphertext shares by other senders, it can craft a ciphertext share c_{i_1} such that combining it with the known shares decrypts to any message. For the latter attack to work, it is essential that the adversarially constructed share is used as share i_1 in the combine step.

Hoepman and Galindo proved the following theorem about the security of their scheme.

THEOREM 5.11 *The Hoepman and Galindo DE scheme is forward-secure against static and restricted adversaries in the random oracle model for H',*

9 The index i_j is included in c_j to be able to explicitly reconstruct $\mathcal{I}$ and thus compute the Lagrange coefficients $\lambda_{i_j}^{\mathcal{I}}$ given a set C.

provided that the decisional bilinear Diffie-Hellman (DBDH) assumption and the DDH assumption in $\mathbb{G}_1$ hold, and provided that the authenticated encryption scheme is IND-CCA secure.

Hoepman and Galindo make their DE scheme key-evolving by pre-generating Shamir secret shares and corresponding public keys for each of the epochs. To transition to the next epoch, the senders securely delete the current key-share and load the key-share for next epoch.

5.3.4 *A new distributed encryption scheme*

In Section 5.1 we gave a sketch of our new distributed encryption scheme. Here we fill out the details. Correctness and safety follow from the earlier discussion.

SCHEME 5.12 (DE scheme) The new distributed encryption (DE) scheme is given by the following algorithms, where RIM.gen, RIM.map and RIM.unmap are as in Scheme 5.3.

- $\mathsf{DE.Gen}(1^\ell, k, n, 1^{\ell_m})$. Create a group $\mathbb{G}$ of prime order p, where $p \approx 2^\ell$, such that DDH is hard in $\mathbb{G}$. Let E be the elliptic curve[10] in which $\mathbb{G}$ is embedded. Call $\mathsf{RIM.gen}(1^{\ell_m}, 1^{\ell/2}, E, (\mathbb{G}, p))$ to set up the redundant injective map. Note that this publishes the hash functions H_1 and H_2, and the sizes ℓ'_H and ℓ_c. Let $\mathcal{M} = \{0,1\}^{\ell_m}$ be the message space. Share the public value 1 using Shamir's k-out-of-n secret sharing as follows. Choose coefficients $f_1, \ldots, f_{k-1} \in_R \mathbb{Z}_p$ and use them to define the $k-1$ degree polynomial $f(X) = 1 + \sum_{i=1}^{k-1} f_i X^i$, then $f(0) = 1$. Every sender i is given a share $S_i = (i, s_i)$ where $s_i = f(i)$. Output S_i for each sender, and publish the group description $(\mathbb{G}, p)$.
- $\mathsf{DE.Enc}(S_i, m)$. Given an encryption key $S_i = (i, s_i)$ let $\alpha_i = \mathsf{RIM.map}(m)^{s_i}$. Return $c_i = (i, \alpha_i)$.
- $\mathsf{DE.Comb}(C)$. Let $C = \{c_{i_1}, \ldots, c_{i_k}\}$. Parse each share c_{i_j} as $\left(i_j, \alpha_{i_j}\right)$. Construct$\mathcal{I} = \{i_1, \ldots, i_k\}$, let $c = \prod_{i \in \mathcal{I}} (\alpha_i)^{\lambda_i^{\mathcal{I}}}$ and return $\mathsf{RIM.unmap}(c)$.

In Section 5.6.1 we sketch how to handle arbitrary-length messages.

5.3.5 *Security of the DE scheme*

In this section we sketch the proof of the following theorem.

10 As explained in Section 5.2 using an elliptic curve is but one option. It is, for example, also possible to construct a redundant injective map onto the quadratic residues.

THEOREM 5.13 *In the random oracle model for H_1 and H_2 the new distributed encryption scheme from Scheme 5.12 is forward-secure assuming the* DDH *assumption holds in the group $\mathbb{G}$.*

We first give an ideal model for this scheme, and show that in this model the DE scheme is secure. We then prove that the ideal model and the actual scheme are indistinguishable, hence proving the security of the actual scheme as well.

The ideal scheme

In the ideal scheme the secret sharing is made specific to the message. So, sender i uses a message-specific secret share $s_{m,i}$ to construct a ciphertext share $(i, \alpha_i) = (i, \mathsf{RIM.map}(m)^{s_{m,i}})$ for message m. For each m, the secret shares $s_{m,1}, \ldots, s_{m,n}$ form a random k-out-of-n sharing of the secret 1.

The reader may wonder at this point, if it still suffices to use a degree $k-1$ polynomial. Traditional uses of a secret sharing scheme suggest that the degree should be k instead, since one secret share is already known. This is not the case, for the following two reasons. First, knowing the secret itself does not help in the recovery as the generator, i.e., the encoding of the message, is actually unknown. Second, while it is possible to guess the generator, thus giving k shares in total, an extra share is needed to verify that guess.

LEMMA 5.14 *The ideal* DE *scheme is secure.*

Proof. By construction of the secret shares we only need to consider the shares for the challenge messages m_0 and m_1; all others are completely independent. After the challenge the attacker knows at most $k-1$ ciphertext shares. Hence, no information is leaked as with only $k-1$ shares, both sets of shares are equally likely to combine to $\mathsf{RIM.map}(m_0)$ as they are to $\mathsf{RIM.map}(m_1)$. In fact, for each set of shares there exists a kth share that reconstructs the desired value. □

Indistinguishability of ideal and real scheme

Suppose an attacker can solve DDH problems, i.e., given $(g, X = g^x, Y = g^y, Z = g^z)$ it can decide whether $z = xy$. Then it can break our scheme as follows. It picks three different messages m, m_0, m_1, and calculates $g = \mathsf{RIM.map}(m)$ and $X = \mathsf{RIM.map}(m)^{s_i}$, for the latter it uses one query. Then it sets $Y = \mathsf{RIM.map}(m_0)$ and obtains $Z = \mathsf{RIM.map}(m_b)^{s_i}$ as a response to its challenge query on m_0 and m_1. Now, $b = 0$ if and only if $z = xy$ in the DDH problem, thus breaking the DE security as well.

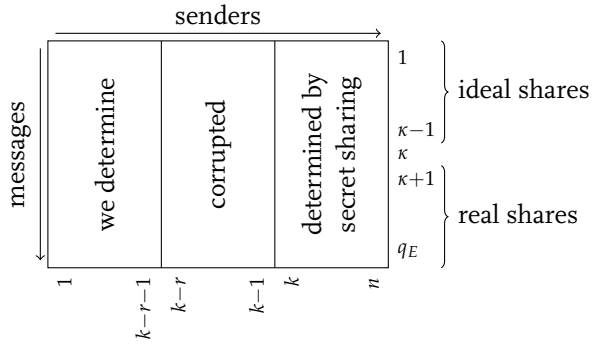


Figure 5.1: This diagram shows which type of secret shares are used in the proof of Lemma 5.15 to answer the encryption queries for the messages. The messages $m_1, \ldots, m_{\kappa-1}$ are answered using ideal shares, the messages $m_{\kappa+1}, \ldots, m_{q_E}$ are answered using real shares. The shares for m_κ are real or random depending on the DDH instance. The diagram also shows how we construct these shares. The first $k-r-1$ we determine, the next r shares are fixed because the adversary has corrupted these senders. The remaining shares follow from the secret sharing scheme.

The indistinguishability proof that we present below shows that any attacker has to solve a DDH problem. The proof is in the hybrid model, see Figure 5.1. Queries for the first $\kappa - 1$ messages will be answered using ideal shares, then the κth message will get either ideal or real shares depending on whether $z = xy$ in the DDH instance, and the remaining message will have real shares. We use the programmability of the redundant injective map to ensure that the generators we choose are used for the messages where we have to use real shares. Induction on κ shows that any distinguisher between ideal and real shares thus solves the DDH instance.

To construct the shares corresponding to the different senders while still ensuring proper secret-sharing we duplicate the DDH instance, combine it with the corrupted shares, and derive the remaining shares based on the underlying secret sharing scheme.

LEMMA 5.15 *The ideal and the real DE schemes are indistinguishable provided the DDH assumption holds in $\mathbb{G}$ and the redundant injective map is programmable.*

Proof. For this proof we use a hybrid scheme that is parametrized by κ. Let $(g, X = g^x, Y = g^y, Z = g^z)$ be a DDH instance for $\mathbb{G}$, our task is to decide whether $z = xy$.

Let $\mathcal{M}_Q = (m_1, \ldots, m_{q_E})$ be the message queries made by the adversary to the redundant injective map. (We do not know the value of these messages yet, but by the programmability of the redundant injective map we can change what they map to as these queries come in.)

In the hybrid scheme, the ciphertext shares corresponding to messages in the set $\mathcal{M}_I = \{m_1, \ldots, m_{\kappa-1}\}$ will be created using ideal shares, whereas the ciphertext shares for messages in $\mathcal{M}_R = \{m_{\kappa+1}, \ldots, m_{q_E}\}$ will be created using the real shares, see Figure 5.1. If $z = xy$ (of the DDH instance) then the hybrid scheme uses real shares for m_κ and ideal shares otherwise. Any distinguisher for the two variants will thus solve the DDH instance. Induction over κ completes the proof.

We now show how to play the security game. Initially we generate $\gamma_m \in_R \mathbb{Z}_p$ for all $m \in \mathcal{M}_R$ such that $g^{\gamma_m} \in E'$ and we generate $\gamma_{m_\kappa} \in_R \mathbb{Z}_p$ such that $X^{\gamma_{m_\kappa}} \in E'$. This is possible since $|\mathbb{G}| / |E'| < 4$ and can be done without knowing the queries in advance.[11] As queries for m come in to the redundant injective map (i.e., for this instantiation of the redundant injective map, as queries for $m' = m \parallel \mathsf{ctr}$ come in to the H_1 oracle), use the programmability of the redundant injective map to set $\mathsf{RIM.map}(m) = g^{\gamma_m}$ for $m \in \mathcal{M}_R$ and to set $\mathsf{RIM.map}(m_\kappa) = X^{\gamma_{m_\kappa}}$.

Then we play the game as follows. Initially, set the set of corrupted senders, I_c, to the empty set. The attacker only makes corruption queries at the start of the game. For every $\mathsf{corrupt}(i)$ query, add i to I_c and generate an arbitrary secret-share $s_i \in_R \mathbb{Z}_p$ and send $S_i = (i, s_i)$ to the challenger. The challenger aborts and the adversary loses if $|I_c| \geq k$.

We now consider three cases of $\mathsf{enc}(i, m)$ queries. The first, where $m \in \mathcal{M}_I$, for which the answers will be using ideal shares, the second when $m = m_\kappa$ and the third when $m \in \mathcal{M}_R$, for which the answers will be using real shares.

Without loss of generality, we assume that the r corrupted senders are numbered $k - r, \ldots, k - 1$, see Figure 5.1. As we cannot play with the corrupted senders, the corresponding shares are always given by $\mathsf{enc}(i, m) = \mathsf{RIM.map}(m)^{s_i}$ for $k - r \leq i \leq k - 1$ and all messages $m \in \mathcal{M}$.[12] This determines r shares. Furthermore, $\mathsf{RIM.map}(m)^1$ is also a valid share, giving $r + 1$ determined shares. In the following we show how to answer the $\mathsf{enc}(i, m)$ queries with $1 \leq i \leq k - (r + 1)$ for all three cases.

Case 1. For messages $m \in \mathcal{M}_I$ generate ideal secret shares $s_{m,i}$ for senders $1 \leq i \leq k - (r + 1)$. The ciphertext shares are given by $\mathsf{enc}(i, m) = \mathsf{RIM.map}(m)^{s_{m,i}}$.

11 Recall that E' consists of all points on the curve whose x coordinate is not too big, and whose y coordinate is 'positive'. We can easily increase ℓ_m or ℓ_c so that the constraint on x loses at most half the points. The constraint on y removes half of the points again, explaining the bound of 4.

12 In this proof we omit the sender's index and just write $\mathsf{enc}(i, m) = \mathsf{RIM.map}(m)^{s_i}$.

For the remaining messages we use the DDH instance $(g, X = g^x, Y = g^y, Z = g^z)$ to compute the answers to the $\mathsf{enc}(i, m)$ queries. First, create an extension as follows. Generate $d_i, e_i \in_R \mathbb{Z}_p$ for $1 \leq i \leq k - (r+1)$ and set $Y_i = Y^{d_i} g^{e_i}$ and $Z_i = Z^{d_i} X^{e_i}$. It can be shown that $(g, X, Y_i = g^{y_i}, Z_i = g^{z_i})$ are DDH tuples such that $z_i = xy_i$ when $z = xy$ in the original problem, and $z_i \in_R \mathbb{Z}_p$ otherwise [125]. We then act as if $s_i = y_i$ for $1 \leq i \leq k - (r+1)$.

Case 2. For m_κ the ciphertext shares for $1 \leq i \leq k - (r+1)$ are given by $\mathsf{enc}(i, m_\kappa) = Z_i^{\gamma_{m_\kappa}}$. If $z_i = xy_i$ then we have $\mathsf{enc}(i, m_\kappa) = (g^{x\gamma_{m_\kappa}})^{y_i} = \mathsf{RIM.map}(m_\kappa)^{s_i}$, making the shares real. Otherwise, the z_i's are random, thus the shares are ideal.

Case 3. For all other messages $m \in \mathcal{M}_R$ the ciphertexts for $1 \leq i \leq k - (r+1)$ are given by

$$\mathsf{enc}(i, m) = Y_i^{\gamma_m} = (g^{\gamma_m})^{y_i} = \mathsf{RIM.map}(m)^{s_i},$$

as desired.

We now determined k shares for every message m, the responses for senders $k, \ldots, n$, see Figure 5.1, are calculated from these by interpolating the exponents

$$\mathsf{enc}(i, m) = \mathsf{RIM.map}(m)^{1\lambda_0^{\mathcal{I}}(i)} \mathsf{enc}(1, m)^{\lambda_1^{\mathcal{I}}(i)} \cdots \mathsf{enc}(k-1, m)^{\lambda_{k-1}^{\mathcal{I}}(i)},$$

where $\mathcal{I} = \{0, \ldots, k-1\}$. We have now described how to answer encryption queries in the find phase. In the challenge phase we create shares in exactly the same manner after first picking a bit $b \in_R \{0, 1\}$ and then constructing the shares for message m_b.

Since the DDH problem is hard, two subsequent hybrid schemes are indistinguishable. Thus, the ideal scheme and the real scheme are indistinguishable as well. □

5.4 FORWARD-SECURE DE SCHEME

The keys of our DE scheme consist of Shamir secret shares. To create a forward secure scheme we need to forward-securely evolve these shares. The difficulty in evolving these shares is twofold. One, our use cases do not allow communication among senders, hence distributed share generation techniques likes those in Section 2.5.1 cannot be used. Two, when non-interactively evolving secrets the traditional approach of using hash chains destroys the threshold structure of the secret shares.

In this section we present a key-evolution scheme that uses an underlying additive secret sharing. This additive sharing can be securely evolved using hash chains, while we can, at the same time, use share conversion on each additive sharing to obtain new Shamir secret shares with the correct threshold structure.

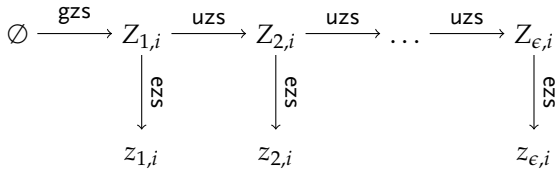


Figure 5.2: Graphical representation of an evolving zero-sharing scheme.

When we combine this key-evolving scheme with our new distributed encryption schemes we obtain a forward-secure distributed encryption scheme. We prove the forward security of the combination in Section 5.4.2. In the next section we prove the forward security of the combination with the batched scheme.

5.4.1 *A key-evolution scheme*

The scheme we present in this section forward securely generates Shamir secret shares of the value 0, which we call an evolving zero-sharing scheme. Simply adding 1 to each share produces a secret sharing of the value 1, as required for our DE scheme. The evolving zero-sharing scheme combines ideas from Cramer et al. [52], see also Section 2.5.2, for the conversion of additive shares to Shamir shares; and ideas from Ohkubo et al. [130] to construct the hash chains.

We take the following approach, see Figure 5.2. Time is split into epochs. Every sender i has an internal state $Z_{\epsilon,i}$ for the current epoch ϵ. The states of all senders combined implicitly define a zero-sharing polynomial z_ϵ. The states are constructed in such a way that every sender can, without interacting with other senders, derive its zero-share $z_{\epsilon,i} = z_\epsilon(i)$ for that epoch. To move to the next epoch, every sender can individually update its internal state. After destroying the previous internal state it is not possible to retrieve any information on it from the current internal state.

Syntax of an evolving zero-sharing scheme

The informal description of the scheme captured in the previous section can be formalized as follows.

SYNTAX 5.16 A k-out-of-n *evolving zero-sharing* scheme for s epochs and a field $\mathbb{F}$ is given by three algorithms: gzs, uzs and ezs which respectively generate an initial sharing, update a share to the next epoch and evaluate a zero-share for the current epoch. In more detail these algorithms work as follows, see also Figure 5.2.

- $\mathsf{gzs}(1^\ell, k, n, s, \mathbb{F})$ takes as input a security parameter ℓ, the threshold k, the number n of senders, the number s of epochs, and a secret sharing field $\mathbb{F}$. It outputs initial states $Z_{1,1}, \ldots, Z_{1,n}$.
- $\mathsf{uzs}(Z_{\epsilon,i})$ is a non-interactive protocol that takes as input the current state $Z_{\epsilon,i}$ and outputs a new state $Z_{\epsilon+1,i}$ or aborts.
- $\mathsf{ezs}(Z_{\epsilon,i})$ takes as input the current state $Z_{\epsilon,i}$ and outputs a zero-share $z_{\epsilon,i}$.

This definition describes a non-interactive scheme because our use cases require this. Interactive schemes are much easier to build. For example, Pedersen's vss protocol, see Protocol 2.10 in Section 2.5.1, could easily function as an evolving zero-sharing scheme. Instead of calculating new shares based on the previous epoch, the senders just run the vss protocol to obtain a fresh zero-sharing. This has the additional benefit of being self-healing.

Forward security of the evolving zero-sharing scheme

Intuitively, forward security requires that no matter what an adversary learns in later epochs (it might, for example, corrupt all senders), it cannot use this knowledge to obtain additional information on the current epoch (for example, to distinguish between the two challenge messages). The key-evolving DE scheme we construct in this section derives its forward security from the keys generated by its evolving zero-sharing scheme. We now define the notion of forward-security of an evolving zero-sharing scheme, which essentially states that even if you obtain the entire internal state of all senders in a later epoch, you do not learn any extra information about the resulting zero-shares (which form the keys for the underlying DE scheme) for the current state. This is precisely what we need to prove security of the combined key-evolving DE scheme.

Consider a epoch ϵ. Clearly, an adversary has the biggest advantage in learning more about epoch ϵ, if it gets the complete state of the system in epoch $\epsilon + 1$. The following definition formalizes the notion that the adversary cannot obtain any extra information on the zero-shares in epoch ϵ in terms of a distinguishability game. The adversary's goal is to distinguish two situations based on the zero-shares it receives for the challenge epoch ϵ. In the first, the challenger follows the protocol honestly. In the second, the challenger replaces the zero-shares with random zero-shares, subject to the constraint that the resulting zero-shares match the view the adversary already had obtained through corruptions—note that this fixes the polynomial if the adversary has corrupted $k - 1$ senders. In both cases the adversary additionally receives

the internal state of all senders for epoch $\epsilon + 1$. We note that it is very liberal to give the adversary all zero-shares in epoch ϵ, as in the actual combination with the DE schemes, the zero-shares will be kept secret.

GAME 5.17 (Evolving zero-sharing (EZS) forward-security game) Consider a k-out-of-n evolving zeros-sharing scheme for s epochs defined over the field $\mathbb{F}$ given by the algorithms gzs, uzs and ezs. Define the following game between the challenger and an adversary $\mathcal{A}$. The game consists of a setup phase and a challenge phase.

SETUP PHASE The challenger runs $\mathsf{gzs}(1^\ell, k, n, s, \mathbb{F})$ and obtains, for each sender i, the initial state $Z_{1,i}$.

QUERY PHASE The challenger initializes the current epoch ϵ to 1, and the set of corrupted senders $I_{1,c} = \emptyset$. The adversary can issue the following set of queries:

- $\mathsf{corrupt}(i)$. The adversary can request sender i to be corrupted. The challenger sends $Z_{\epsilon,i}$ to the adversary and it adds i to $I_{\epsilon,c}$.
- $\mathsf{zeroshares}()$. The adversary can request all the zero-sharing shares for the current epoch. The challenger calculates $z_{\epsilon,i} = \mathsf{ezs}(Z_{\epsilon,i})$ for each sender and sends these to the adversary.
- $\mathsf{next}()$. On a $\mathsf{next}()$ query the challenger updates the states of senders i by setting $Z_{\epsilon+1,i} = \mathsf{uzs}(Z_{\epsilon,i})$. Finally, the challenger sets $I_{\epsilon+1,c} = I_{\epsilon,c}$ and $\epsilon = \epsilon + 1$.

CHALLENGE PHASE In the challenge phase the adversary $\mathcal{A}$ outputs a challenge epoch ϵ^*. If the adversary already queried the challenger using $\mathsf{zeroshares}()$ in the challenge epoch ϵ^* the adversary loses. Otherwise, the challenger chooses a bit $b \in \{0,1\}$. If $b = 0$ it responds as if the adversary made a $\mathsf{zeroshares}()$ query in epoch ϵ^*. Otherwise, it creates random zero-shares (subject to the constraint imposed by the corrupted senders) and returns these. More precisely, it randomly generates a zero-sharing polynomial z of degree $k-1$, such that $z(i) = 0$ for all $i \in I_{\epsilon^*,c}$, and returns $\mathsf{ezs}(Z_{\epsilon^*,i}) + z(i)$ for each sender i.

OUTPUT PHASE Finally the adversary $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$. The adversary wins if $b = b'$.

The advantage of adversary $\mathcal{A}$ is given by $\mathsf{Adv}^{\mathrm{EZS}}_{\mathcal{A}}(1^\ell) = 2\left|\Pr[b = b'] - \frac{1}{2}\right|$, where the probability is over the random bits of the challenger and the adversary. We say the evolving zero-sharing scheme is *forward-secure* if $\mathsf{Adv}^{\mathrm{EZS}}_{\mathcal{A}}(1^\ell)$ is negligible for every PPT adversary $\mathcal{A}$.

Key-evolution scheme

We follow a variant of Cramer et al. [52]'s share conversion scheme, see Section 2.5.2, in constructing a zero-sharing polynomial in such a way that sender i can only evaluate the polynomial at the point i. For every set $A \subset [n]$ of cardinality $n-(k-2)$ we define the $k-1$ degree polynomial $g_A(X) = X\prod_{i\in[n]\setminus A}(X-i)$. Hence, $g_A(0) = 0$ and $g_A(X) = 0$ for all $X \notin A$. Our zero-sharing polynomial z_ϵ is then given by

$$z_\epsilon(X) = \sum_{\substack{A\subset[n]\\|A|=n-(k-2)}} r_{\epsilon,A}\cdot g_A(X),$$

where a factor $r_{\epsilon,A}$ is known only to the senders $i \in A$. By construction, z_ϵ is of degree $k-1$ and $z_\epsilon(0)$ is indeed 0. It can be shown that $k-2$ colluding parties cannot recover z_ϵ. Furthermore, for every zero-sharing polynomial z of degree $k-1$, there exist values for the $r_{\epsilon,A}$s such that $z_\epsilon = z$. This gives the following scheme.

SCHEME 5.18 (Evolving zero-sharing scheme) The evolving zero-sharing scheme is given by the following three algorithms.

- $\mathsf{gzs}(1^{\ell_h}, k, n, s, \mathbb{F})$. On input of a security parameter ℓ_h, a threshold k, the number n of senders and a field $\mathbb{F}$ proceed as follows. First, create two hash functions $h_1 : \{0,1\}^{\ell_h} \to \{0,1\}^{\ell_h}$ and $h_2 : \{0,1\}^{\ell_h} \to \mathbb{F}$. Next, for each $A \subset [n]$ of cardinality $n-(k-2)$ generate a random share $\bar{r}_{1,A} \in_R \{0,1\}^{\ell_h}$ and for each sender i set $Z_{1,i} = (\bar{r}_{1,A})_{A\ni i}$.
- $\mathsf{uzs}(Z_{\epsilon,i})$. This algorithm is non-interactive. First, parse $Z_{\epsilon,i}$ as $(\bar{r}_{\epsilon,A})_{A\ni i}$, and set $\bar{r}_{\epsilon+1,A} = h_1(\bar{r}_{\epsilon,A})$ for A such that $i \in A$. Then return $Z_{\epsilon+1,i} = (\bar{r}_{\epsilon+1,A})_{A\ni i}$.
- $\mathsf{ezs}(Z_{\epsilon,i})$. To derive the zero-share, parse $Z_{\epsilon,i}$ as $(\bar{r}_{\epsilon,A})_{A\ni i}$. Then, set $r_{\epsilon,A} = h_2(\bar{r}_{\epsilon,A})$ for A such that $i \in A$ and determine

$$z_{\epsilon,i} = z_\epsilon(i) = \sum_{\substack{A\subset[n]\\|A|=n-(k-2)}} r_{\epsilon,A}\cdot g_A(i).$$

  Finally, return $z_{\epsilon,i}$.

We first prove the following lemma.

LEMMA 5.19 *Let I_c be a set of senders controlled by the adversary, let $\hat{r}_{\epsilon,A}$ for $A \subset [n]$ of cardinality $n-(k-2)$ be additive shares as before and let z_ϵ be the resulting zero-sharing polynomial given by*

$$z_\epsilon(X) = \sum_{\substack{A\subset[n]\\|A|=n-(k-2)}} \hat{r}_{\epsilon,A}\cdot g_A(X).$$

Similarly, let $\tilde{z}_\epsilon$ be the zero-sharing polynomial induced by the additive shares $\tilde{r}_{\epsilon,A}$, i.e.,

$$\tilde{z}_\epsilon(X) := \sum_{\substack{A\subset[n]\\|A|=n-(k-2)}} \tilde{r}_{\epsilon,A}\cdot g_A(X).$$

Then, for each zero-sharing polynomial z of degree $k-1$ such that $z(i)=0$ for all $i\in I_c$, there exists additive shares $\tilde{r}_{\epsilon,A}$ subject to the constraint that $\tilde{r}_{\epsilon,A}=\hat{r}_{\epsilon,A}$ for all A with $A\cap I_c\neq\emptyset$, such that $\tilde{z}_\epsilon = z_\epsilon + z$.

Proof. Let $r=|I_c|$. If $r\geq k-1$, then z is identically zero, and hence the statement is clearly true. So, assume that $r<k-1$. Without loss of generality, we assume that $I_c=\{n-r+1,\ldots,n\}$.

We choose the shares $\tilde{r}_{\epsilon,A}$. To show that we did so correctly, we show that $\tilde{z}_\epsilon = z_\epsilon + z$ for $k-1$ non-zero points. This suffices because both zero-sharing polynomials $\tilde{z}_\epsilon$ and $z_\epsilon+z$ are of degree $k-1$. We set $\tilde{r}_{\epsilon,A}=\hat{r}_{\epsilon,A}$ for $A\cap I_c\neq\emptyset$ as required. This ensures that $\tilde{z}_\epsilon(i) = z_\epsilon(i)+z(i)$ for $i\in I_c$, fixing the first r points. Let $c=k-1-r$ be the remaining number of points to check. Note that

$$\tilde{z}_\epsilon(i) = \sum_{\substack{A\subset[n]\\|A|=n-(k-2)}} \tilde{r}_{\epsilon,A}g_A(i) = \underbrace{\sum_{A\cap I_c\neq\emptyset} \hat{r}_{\epsilon,A}g_A(i)}_{\text{fixed}} + \underbrace{\sum_{A\cap I_c=\emptyset} \tilde{r}_{\epsilon,A}g_A(i)}_{\text{not fixed}},$$

where the fixed part contains values that are known to the adversary, and hence cannot be changed. The not-fixed part can, however, be changed, since those additive shares are unknown to the adversary. Consider the sets:

$$\begin{aligned} A_i &= [n]\setminus(\{1,\ldots,i-1,i+1,\ldots,c\}\cup I_c)\\ &= \{i,k-r,\ldots,n-r\}\end{aligned}$$

for $i\in[c]$. These sets are of cardinality $n-(k-2)$ and are such that only set A_i influences the value for sender i (and not any other $j\in[c]\cup I_c$, $j\neq i$), i.e.

$$\tilde{z}_\epsilon(i) = \left[\sum_{\substack{A\subset[n]\\|A|=n-(k-2),\forall j:A\neq A_j}} \tilde{r}_{\epsilon,A}g_A(i)\right] + \tilde{r}_{\epsilon,A_i}g_{A_i}(i),$$

for $i\in[c]$. Now, set $\tilde{r}_{\epsilon,A}=\hat{r}_{\epsilon,A}$ for all $A\neq A_i$. Then, choose the $\tilde{r}_{\epsilon,A_i}$s such that $\tilde{z}_\epsilon(i)=z_\epsilon(i)+z(i)$ holds for $i\in[c]$ (by the construction of the A_i, these choices are independent). Since $\tilde{z}_\epsilon$ equals $z_\epsilon+z$ at k points (including the point 0), we achieved the desired equality. □

The construction and the proof of the following lemma are inspired by the Ohkubo et al. scheme [130].

THEOREM 5.20 *The evolving zero-sharing scheme from Scheme 5.18 is forward secure in the random oracle model for h_2.*

Proof. Let $Z_{\epsilon^*,i} = (\bar{r}_{\epsilon^*,A})_{A \ni i}$ and let z_{ϵ^*} be the zero-sharing polynomial in challenge epoch ϵ^*, corresponding to the normal situation, i.e., when $b = 0$. Now, we show how to change this to $z_{\epsilon^*} + z$ by modifying the random oracle for h_2 in such a way that this cannot be detected by the adversary.

Let $\hat{r}_{\epsilon^*,A} = h_2(\bar{r}_{\epsilon^*,A})$ be the additive shares that would be derived in the normal situation $b = 0$. Use Lemma 5.19 on $\hat{r}_{\epsilon^*,A}$ and I_c to construct $\tilde{r}_{\epsilon^*,A}$ such that the polynomial $\tilde{z}_{\epsilon^*}$ they induce is equal to $z_{\epsilon^*} + z$.

We now show how to change h_2 in the random oracle model so that for $b = 1$, we ensure that $r_{\epsilon^*,A} = h_2(\bar{r}_{\epsilon,A}) = \tilde{r}_{\epsilon,A}$ for all A, so that we induce the polynomial $z_{\epsilon^*} + z$ as desired.

By construction, $\hat{r}_{\epsilon^*,A} = \tilde{r}_{\epsilon^*,A}$ for most sets A. For these sets we do not change h_2. For the sets A such that $\tilde{r}_{\epsilon^*,A} \neq \hat{r}_{\epsilon^*,A}$,[13] by construction $A \cap I_c = \emptyset$ for these sets A, we change h_2 by setting $h_2(\bar{r}_{\epsilon^*,A}) = \tilde{r}_{\epsilon^*,A}$. These changes to the random oracle succeed with overwhelming probability, because (1) the new $\tilde{r}_{\epsilon^*,A}$ is random by construction, (2) the adversary does not know the (random) value $\bar{r}_{\epsilon^*,A}$, and (3) because of h_1 it cannot learn anything about $\bar{r}_{\epsilon^*,A}$ using information from later epochs.

After changing h_2 we are in the situation $b = 1$, since the adversary cannot detect the changes, the two situations are indistinguishable. □

Generating the initial additive shares

The evolving zero-sharing scheme as described here still uses a trusted party to create the initial set of base additive shares $\bar{r}_{1,A} \in_R \{0,1\}^{\ell_h}$. If some form of one-time initial communication among the senders is possible, they could use the $\mathsf{AVSS}^+(n, k-1, \mathbb{G}, g, p)$ protocol (see Protocol 2.12 in Section 2.5.1) to jointly generate these base additive shares (here $(\mathbb{G}, p, g)$ is a cyclic group of sufficient size in which the DL problem is hard).

5.4.2 *A key-evolving distributed encryption scheme*

In this section we build a key-evolving DE scheme using the evolving zero-sharing scheme of the previous section. The latter scheme generates as many distributed zero-sharing polynomials as we want. By

13 By the construction in Lemma 5.19 this is only the case for $c \leq k-1$ values.

adding the constant polynomial 1 to this polynomial we obtain the key-sharing polynomial in our DE scheme.

SCHEME 5.21 (KDE SCHEME) The key-evolving distributed encryption (KDE) scheme is constructed from the our new distributed encryption scheme given by the algorithms DE.Gen, DE.Enc and DE.Comb, and the evolving zero-sharing scheme from Scheme 5.18 given by the algorithms gzs, uzs and ezs. It is defined by the following four algorithms.

- $\mathsf{KDE.Gen}(1^\ell, k, n, s, 1^{\ell_m})$. The KDE.Gen algorithm first runs the algorithm $\mathsf{DE.Gen}(1^\ell, k, n, \ell_m)$ to obtain the group description $(\mathbb{G}, p)$ and the redundant injective map information (E, H_1, H_2) which it outputs as well. Here group $\mathbb{G}$ is of prime order p. It then calls $\mathsf{gzs}(1^\ell, k, n, s, \mathbb{Z}_p)$ to obtain $Z_{1,1}, \ldots, Z_{1,n}$, sets $s_{1,i} = 1 + \mathsf{ezs}(Z_{1,i})$ and outputs $S_{1,i} = (i, s_{1,i}, Z_{1,i})$ for each sender.
- $\mathsf{KDE.UpdKey}(S_{\epsilon,i})$. Let $S_{\epsilon,i} = (i, s_{\epsilon,i}, Z_{\epsilon,i})$. The algorithm then sets $Z_{\epsilon+1,i} = \mathsf{uzs}(Z_{\epsilon,i})$ and $s_{\epsilon+1,i} = 1 + \mathsf{ezs}(Z_{\epsilon+1,i})$ and returns $S_{\epsilon+1,i} = (i, s_{\epsilon+1,i}, Z_{\epsilon+1,i})$ or aborts if uzs aborts.
- $\mathsf{KDE.Enc}(S_{\epsilon,i}, m)$. Let $S_{\epsilon,i} = (i, s_{\epsilon,i}, Z_{\epsilon,i})$. To encrypt a message m, algorithm KDE.Enc returns the result of $\mathsf{DE.Enc}((i, s_{\epsilon,i}), m)$.
- $\mathsf{KDE.Comb}(C)$. To combine ciphertexts it runs $\mathsf{DE.Comb}(C)$.

Efficiency

The evolving zero-sharing scheme has complexity $\binom{n}{k-2}$ in both space and time to store and evaluate the zero-shares. While this number is exponential, it is almost always much smaller than the cost of recovering all k-times encrypted plaintexts in a real scenario (see Table 5.1 on page 130). In particular, it is comparable to recovering a single message in the batched scheme which we will present in the next section. Furthermore, its space complexity is independent of the number of epochs, which is a big gain with respect to the original scheme by Hoepman and Galindo [83] where the space complexity is linear in the number of epochs. In many practical applications this number will be a lot bigger than $\binom{n}{k-2}$.

Security

The security of the KDE scheme can easily be reduced to that of the DE scheme by using the properties of the evolving zero-sharing scheme.

THEOREM 5.22 *The new KDE scheme is forward secure (in the sense of Game 5.7) provided that the DE scheme is secure (in the sense of Definition 5.8) and the evolving zero-sharing scheme is forward secure (in the sense of Game 5.17). The proof is in the random oracle model for h_2.*

We first give a sketch of the proof.

Sketch. We reduce the security of the KDE scheme to that of the DE scheme. To set up the system we generate random $\bar{r}_{1,A}$s. We guess the challenge phase ϵ^* and simulate all epochs except ϵ^*, where we use our DE oracle. To ensure that this is not detected we must ensure that corrupted hosts have the correct secret shares. We do this by modifying $h_2(\bar{r}_{\epsilon^*,A})$ in the random oracle model on $\bar{r}_{\epsilon^*,A}$s that were not yet known to the adversary. Then, queries in epoch ϵ^* can be answered by our DE oracle. The distribution of the secret shares does not correspond to the initial $\bar{r}_{1,A}$s, however, the forward-security of the evolving zero-sharing scheme ensures this cannot be detected. □

Proof. Suppose we have an adversary $\mathcal{A}$ against the KDE scheme. We then build an adversary $\mathcal{B}$ against the underlying DE scheme. Adversary $\mathcal{B}$ receives the system parameters from the challenger and forwards them to $\mathcal{A}$. Next, adversary $\mathcal{B}$ makes a guess ϵ^* for the challenge epoch and initializes the set of corrupted senders I_c to $\emptyset$.

Adversary $\mathcal{B}$ will fully simulate all epochs, except epoch ϵ^*, where it will use its oracle to answer the queries. For all $A \subset [n]$, such that $|A| = n - (k-2)$ generate $\bar{r}_{1,A} \in_R \{0,1\}^{\ell_h}$.

We now look into the details of the evolving zero-sharing scheme. By generating $\bar{r}_{1,A}$'s we have completely fixed the system, but we still need to ensure that epoch ϵ^* can be answered using our oracles. To this end we will change the value of the hash function $h_2(\bar{r}_{\epsilon^*,A_i})$ for specific sets A_i belonging to corrupted parties i. These sets A_i will be chosen in such a way, that $\bar{r}_{\epsilon^*,A_i}$ is not known to any previously corrupted party.

To see how to answer $\mathsf{corrupt}(i)$ queries, we first focus on epoch ϵ^* as we have to ensure that corrupted hosts derive the correct secret shares, i.e., shares that correspond to those returned by our DE oracle, in epoch ϵ^*. Let I_c be the set of senders that were corrupted earlier and f_{ϵ^*} the secret-sharing polynomial induced by the values $\bar{r}_{\epsilon^*,A}$. First, we corrupt sender i using our oracle to obtain its internal state $(i, s_{\epsilon^*,i}) = \mathsf{corrupt}(i)$. We need to ensure that $f_{\epsilon^*}(i) = s_{\epsilon^*,i}$. Pick a set A_i of cardinality $n-(k-2)$ such that $I_c \cap A_i = \emptyset$ and $i \in A_i$. This is possible, since the constraints in the challenge phase require $|I_c \cup \{i\}| < k$, therefore, $|I_c|$ will be at most $k-2$. For all other sets $A \ni i$ obtain $r_{\epsilon^*,A} = h_2(\bar{r}_{\epsilon^*,A})$ as usual. Then choose r_{ϵ^*,A_i} such that:

$$s_{\epsilon^*,i} = f_{\epsilon^*}(i) = 1 + \sum_{\substack{A \subset [n] \\ |A| = n-k+1}} r_{\epsilon^*,A} \cdot g_A(i).$$

By the choice of the set A_i the coefficient $\bar{r}_{\epsilon^*,A_i}$ is not known to any corrupted host, hence we can use the random oracle model to ensure that

$h_2(\bar{r}_{\epsilon^*,A_i}) = r_{\epsilon^*,A_i}$. With overwhelming probability the adversary cannot detected this as the $\bar{r}_{\epsilon^*,A_i}$s are random. This construction ensures that corrupted hosts derive the correct secret shares in epoch ϵ^*. Finally, return $(i, (\bar{r}_{\epsilon,A})_{A \ni i})$ in response to a $\mathsf{corrupt}(i)$ query in epoch ϵ.

Now $\mathcal{B}$ proceeds as follows. For all epochs except ϵ^* it knows the complete state of the system, and can thus answer all $\mathcal{A}$'s queries. For epoch ϵ^*, all corrupted hosts will, by construction of the hash-function, have the correct secret shares for this epoch. All other queries can be answered by the oracle.

The forward security of the evolving zero-sharing scheme ensures that it is not possible for the adversary to detect that the underlying secret sharing in epoch ϵ^* (we use the one induced by our DE oracle) does not match the $\bar{r}_{\epsilon,A}$s of the key-evolving scheme.[14]

In the challenge phase, $\mathcal{A}$ will announce its challenge epoch ϵ'. If $\epsilon' \neq \epsilon^*$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ will pass the challenge from $\mathcal{A}$ on to its own oracle. Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. Adversary $\mathcal{B}$ has the same advantage as $\mathcal{A}$ up to a factor $1/s$ for guessing the epoch. This proves the result. □

5.4.3 *Applying this idea to Hoepman and Galindo's scheme*

Unfortunately, the idea of forward-securely evolving zero-shares does not apply nicely to Hoepman and Galindo's scheme. The core of the problem is that the underlying identity-based encryption scheme requires the public IBE key $E = h^e$ to be known to all parties. Hence, to non-interactively evolve the secret e, all senders need to construct new shares of the new secret, but also the new public IBE key needs to be known to all parties. While the former is possible using Cramer's share conversion scheme, see Section 2.5.2, it does not enable the latter without communication. We do not know of any approach that does enable the non-interactive generation of Shamir secret shares and the simultaneous publishing of the corresponding public key.

At the same time, evolving the secret e—rather than just the way it is shared, as we do in our KDE scheme—is essential in Hoepman and Galindo's scheme. Suppose it is kept constant, then, in some later epoch, the adversary can corrupt k senders, and recover the secret e. Using this secret, it can also calculate the IBE key $H(m_0)^e$, try to decrypt its challenge ciphertext shares, and hence win the game.

14 This proof only changes h_2 for corrupted hosts, whereas the proof of Theorem 5.20 changes h_2 for non-corrupted hosts, so they can indeed be combined.

5.5 EFFICIENT SOLUTIONS FOR SMALL DOMAINS

As was already analyzed by Hoepman and Galindo [83], using a distributed encryption will be rather inefficient. In the current definition, the only way to find all messages that were encrypted by at least k different senders is through brute-force. In particular, we need to try to combine all possible combinations of k shares from different senders and verify whether that combination yields a match. Simple tricks, such as attaching the hash of the message to the ciphertext shares, do not work, as they allow an attacker to test if the shares belong to a given message.

In this section, we explore a time-memory trade-off that is more efficient for small message domains. The crucial difference with the previous sections is that we will now operate in a batched setting. At the end of a epoch the sender generates a share for *every* message. It generates a proper share for every message it needs to send, as before, and a random value for all other messages. Now, we know directly which shares belong to a given message. This reduces the exponential term in the combing phase considerably. Also, since the message is known a priori, the only remaining task of the combiner is to determine whether this message was encrypted by a sufficient number of senders. In particular, we can replace the redundant injective map with a hash function.

5.5.1 *Syntax*

For small message spaces, the following definition of a *batched key-evolving distributed encryption* scheme makes sense.

SYNTAX 5.23 A k-out-of-n *batched key-evolving distributed encryption* scheme with lifetime divided into s epochs, or (k, n, s)-BKDE scheme, consists of the following four algorithms.

- $\mathsf{BKDE.Gen}(1^\ell, k, n, s, \mathcal{M})$. Given the security parameter 1^ℓ, the threshold k, the number n of senders, the number s of epochs, and a message space $\mathcal{M}$, it generates an initial encryption key $S_{1,i}$ for each sender $i \in [n]$. It returns these encryption keys as well as the system parameters.
- $\mathsf{BKDE.Enc}(S_{\epsilon,i}, M)$. Given an encryption key $S_{\epsilon,i}$ corresponding to sender i at epoch ϵ and a set of messages $M \subset \mathcal{M}$, this function returns a vector C_i of ciphertext shares of length $|\mathcal{M}|$.
- $\mathsf{BKDE.UpdKey}(S_{\epsilon,i})$. The key update function takes as input $S_{\epsilon,i}$ and outputs the key $S_{\epsilon+1,i}$ for the next epoch. The function aborts if $\epsilon + 1 > s$.

- $\mathsf{BKDE.Comb}(C_1, \ldots, C_n)$. Given the ciphertext share vectors $C_1, \ldots, C_n$ produced by the senders, the function BKDE.Comb returns a set of messages M.

A key-evolving batched distributed encryption scheme must satisfy the following correctness and safety requirements.

CORRECTNESS Let the encryption keys $S_{\epsilon,i}$ be generated as above. Consider n subsets $M_i \subset \mathcal{M}$, and let $C_i = \mathsf{BKDE.Enc}(S_{\epsilon,i}, M_i)$ for each sender i. Then the result M of $\mathsf{BKDE.Comb}(C_1, \ldots, C_n)$ is such that $m \in M$ if $m \in M_i$ for at least k different M_i.

SAFETY Let M_i, C_i for $i \in [n]$ and M be as for correctness. If $m \notin M_i$ for at least $n - k + 1$ different M_i, then $m \notin M$ with overwhelming probability.

5.5.2 *Security definition*

The following game captures the security properties of our protocol. The game is very similar to the forward-security game of the regular KDE forward-security game. The biggest change is in the challenge phase. In the regular KDE game the adversary only gets the shares for one of the challenge messages. However, in the batched scheme, the adversary knows which shares belong to which message (as this is fixed by the row in which the shares appear). To mimic this, while still encoding the challenge query, the challenger gets shares corresponding to both challenge messages, but only the shares corresponding to one of the messages are real. The challenger's task now is to decide which of these it is.[15]

GAME 5.24 (Batched KDE forward-security game) Consider a (k, n, s)-BKDE batched key-evolving distributed encryption scheme. Since the batched KDE forward-security game is very similar to the KDE forward-security game of a (k, n, s)-KDE scheme (see Game 5.7), we note only the changes. The algorithms BKDE.Gen and BKDE.UpdKey replace the algorithms KDE.Gen and KDE.UpdKey.

SETUP PHASE First, the adversary outputs a message space $\mathcal{M}$ it wants to attack, then the setup phase runs as before.

FIND PHASE The adversary is allowed to make $\mathsf{bcorrupt}(i)$, $\mathsf{bnext}()$ and $\mathsf{benc}(i, M_i)$ queries. The first two are implemented using $\mathsf{corrupt}(i)$ and $\mathsf{next}()$ respectively. On input of a query

15 In the original version of this game we gave the adversary only the set of real shares (and omitted the random ones, as a result the relation between the shares and their proper rows in the scheme was not evident). While there exists an easy reduction between these, we believe this new game more faithfully represents the adversary's capabilities.

$\mathsf{benc}(i, M_i)$, where $i \in [n]$, sender i is not yet corrupted, i.e., $i \notin I_{\epsilon,c}$, and $M_i \subset \mathcal{M}$, the challenger sends the vector $\mathsf{BKDE.Enc}(S_{\epsilon,i}, M_i)$ to the adversary.

CHALLENGE PHASE If the challenge on m_0 and m_1 at hosts I_{nc} is valid (see Game 5.7, where Q_0 and Q_1 are defined analogously: sender $j \in Q_i$ if $m_i \in M_j$ for any $\mathsf{benc}(j, M_j)$ query in epoch ϵ^*), the challenger chooses $b \in_R \{0,1\}$, sets

$$C_j = \mathsf{BKDE.Enc}(S_{\epsilon^*,j}, \{m_b\}) = (c_{mj})_{m \in \mathcal{M}}$$

for each $j \in I_{nc}$. It then returns the tuples of ciphertext shares $(c_{m_0 j}, c_{m_1 j})$ to the challenger for each $j \in I_{nc}$.

OUTPUT PHASE The output phase is unchanged.

The adversary $\mathcal{A}$'s advantage is defined as $\mathsf{Adv}^{\text{BKDE}}_{\mathcal{A}}(1^\ell) = 2\left|\Pr[b' = b] - \frac{1}{2}\right|$, where the probability is over the random bits of the challenger and the adversary. A batched key-evolving distributed encryption (BKDE) scheme is called *forward secure* if $\mathsf{Adv}^{\text{BKDE}}_{\mathcal{A}}(1^\ell)$ is negligible for every PPT adversary $\mathcal{A}$.

5.5.3 *The scheme*

In the batched scheme, sender i outputs a complete vector of ciphertext shares $C_i = (c_{mi})_{m \in \mathcal{M}}$ at the end of an epoch. Let $H : \{0,1\}^* \to \mathbb{G}$ be a cryptographic hash function, and (i, z_i) be sender i's k-out-of-n secret share of the secret zero. An element c_{mi} then equals $H(m)^{z_i}$ when sender i been asked to encrypt m, and $c_{mi} \in_R \mathbb{G}$ otherwise. Intuitively, when the secret shares are unknown, these two are indistinguishable. The full scheme is given by the following definition. Note the similarities with our KDE scheme.

SCHEME 5.25 (Batched KDE scheme) Let $(\mathsf{gzs}, \mathsf{uzs}, \mathsf{ezs})$ be an evolving zero-sharing scheme. The following algorithms define a batched key-evolving distributed encryption (BKDE) scheme.

- $\mathsf{BKDE.Gen}(1^\ell, k, n, s, \mathcal{M})$. Generate a cyclic group $\mathbb{G}$ such that its order p is of size ℓ bits. Then construct a hash function $H : \{0,1\}^* \to \mathbb{G}$. Create the secret sharing of zero by calling $\mathsf{gzs}(k, n, s, \mathbb{Z}_p)$ to obtain $Z_{1,1}, \ldots, Z_{1,n}$. Let $z_{1,i} = \mathsf{ezs}(Z_{1,i})$. Output $S_{1,i} = (z_{1,i}, Z_{1,i})$ for each sender, together with a description of $\mathbb{G}$ and the hash function H.

- BKDE.Enc$(S_{\epsilon,i}, M)$. Let $S_{\epsilon,i} = (z_{\epsilon,i}, Z_{\epsilon,i})$, and return the ciphertext share vector $C_i = (c_{mi})_{m \in \mathcal{M}}$ such that for all $m \in \mathcal{M}$

$$c_{mi} = \begin{cases} H(m)^{z_{\epsilon,i}} & \text{if } m \in M \\ h \in_R \mathbb{G} & \text{otherwise.} \end{cases}$$

- BKDE.UpdKey$(S_{\epsilon,i})$. Let $S_{\epsilon,i} = (z_{\epsilon,i}, Z_{\epsilon,i})$. Set $Z_{\epsilon+1,i} = \mathsf{uzs}(Z_{\epsilon,i})$ and $z_{\epsilon+1,i} = \mathsf{ezs}(Z_{\epsilon+1,i})$. Return $S_{\epsilon+1,i} = (z_{\epsilon+1,i}, Z_{\epsilon+1,i})$ or abort if uzs does.

- BKDE.Comb$(C_1, \ldots, C_n)$. For each $m \in \mathcal{M}$, do the following. Consider all shares $(c_{m1}, \ldots, c_{mn})$ corresponding to message m from senders 1 through n. For all possible combinations of index sets $\mathcal{I} \subseteq \{1, \ldots, n\}$ of size k verify whether

$$\prod_{i \in \mathcal{I}} (c_{mi})^{\lambda_i^{\mathcal{I}}} = 1.$$

  If so, add m to the set of messages to return.

The structure of this scheme is similar to that of our new DE and KDE schemes. Correctness and safety are easy to check. The proof of security of the batched key-evolving distributed encryption scheme closely follows the argument in Sections 5.3.5 and 5.4.2. In particular, we again construct an idealized version of this scheme—with different zero-shares for each plaintext—which is trivially secure, and then prove that the ideal scheme and the real scheme are indistinguishable.

We first prove security of the non key-evolving version. In this version the evolving zero-sharing scheme is omitted and the keys consist solely of the zero-shares. In the ideal version of the scheme, these zero-shares are message specific. For each message $m \in \mathcal{M}$ sender i uses the message-specific zero-share $z_{m,i}$. The shares $z_{m,1}, \ldots, z_{m,n}$ form a k-out-of-n zero-sharing. We first prove the analog to Lemma 5.14.

LEMMA 5.26 *The ideal batched DE scheme is secure.*

Proof. Again, we only consider the shares corresponding to the challenge messages m_0 and m_1 (the other are completely independent). After the challenge query, the adversary has at most $k-1$ non-random shares for each challenge message. To decide which bit b its challenger picked, it needs to decide whether the challenge shares corresponding to m_0 or to m_1 are non-random. However, even if the $k-1$ shares of a challenge message are real shares, then they are still indistinguishable from random, since the k-out-of-n zero-shares are random for each message (and have precisely $k-1$ degrees of freedom). Therefore, the adversary cannot decide whether m_0 or m_1 received extra challenge shares in the challenge query. □

LEMMA 5.27 *In the random oracle model for H, the ideal and real batched (non key-evolving) distributed encryption schemes are indistinguishable, provided that the DDH assumption in $\mathbb{G}$ holds.*

The following proof is very similar to the proof of Lemma 5.15. We make three noticeable changes. One, we replace the secret shares of the value one by the secret shares of the value zero. Two, because of the batched structure, all possible messages are already known, and we do not need to use the random oracle to extract them. Three, we do not need to use the redundant injective map, instead we directly control the random oracle H.

Proof. For this proof we use a hybrid scheme that is parametrized by κ. Let $(g, A = g^a, B = g^b, C = g^c)$ be a DDH instance for $\mathbb{G}$, our task is to decide whether $c = ab$.[16]

Let $(m_1, \ldots, m_{|\mathcal{M}|})$ be the ordered plaintexts. In the hybrid scheme, the ciphertext shares corresponding to messages in the set $\mathcal{M}_I = \{m_1, \ldots, m_{\kappa-1}\}$ will be created using ideal shares, whereas the ciphertext shares for messages in $\mathcal{M}_R = \{m_{\kappa+1}, \ldots, m_{|\mathcal{M}|}\}$ will be created using the real shares, compare with Figure 5.1. If $c = ab$ (of the DDH instance) then the hybrid scheme uses real shares for m_κ and ideal shares otherwise. Any distinguisher for the two variants will thus solve the DDH instance. Induction over κ completes the proof.

We now show how to play the security game. Initially we generate $\gamma_m \in_R \mathbb{Z}_p$ for all $m \in \mathcal{M}_R \cup \{m_\kappa\}$. We answer hash queries for $m \in \mathcal{M}_R$ using g^{γ_m}, while we answer the hash query $H(m_\kappa)$ using $A^{\gamma_{m_\kappa}}$. All other hash-queries are answered normally.

Then we play the game as follows. Initially, let the set of corrupted senders I_c be the empty set. The attacker only makes corruption queries at the start of the game. For every $\mathsf{bcorrupt}(i)$ query, add i to I_c and generate an arbitrary secret-share $z_i \in_R \mathbb{Z}_p$ and send $S_i = (i, z_i)$ to the challenger. The challenger aborts and the adversary loses if $|I_c| \geq k$.

We now show how to answer $\mathsf{benc}(i, M_i)$ queries. In particular, we show how to construct the components c_{mi} of the resulting vector C_i. If $m \notin M_i$, simply pick $c_{mi} \in_R \mathbb{G}$. We now consider three cases given that $m \in M_i$: the first, where $m \in \mathcal{M}_I$, for which the answers will be using ideal shares, the second when $m = m_\kappa$ and the third when $m \in \mathcal{M}_R$, for which the answers will be using real shares.

Without loss of generality, we assume that the r corrupted senders are numbered $k - r, \ldots, k - 1$, compare with Figure 5.1. As we cannot play with the corrupted senders, the corresponding shares are always given by $c_{im} = H(m)^{z_i}$ for $k - r \leq i \leq k - 1$ and all messages $m \in \mathcal{M}$.

16 Note that we deviate from our standard notation for DDH instances to prevent notational conflicts with the zero-shares.

This determines r shares. Furthermore, $1 = H(m)^0$ is also a valid share, giving $r+1$ determined shares. In the following we show how to construct c_{im} where $1 \leq i \leq k-(r+1)$ for all three cases.

Case 1. For messages $m \in \mathcal{M}_I$ generate ideal secret shares $z_{m,i}$ for senders $1 \leq i \leq k-(r+1)$. The components are $c_{im} = H(m)^{z_{m,i}}$.

For the remaining messages we use the DDH instance $(g, A = g^a, B = g^b, C = g^c)$ to compute c_{im}. Create extra DDH instances $(g, A, B_i = g^{b_i}, C_i = g^{c_i})$ as in the proof of Lemma 5.15 such that $c_i = ab_i$ when $c = ab$ in the original problem, and $c_i \in_R \mathbb{Z}_p$ otherwise. We then act as if $z_i = b_i$ for $1 \leq i \leq k-(r+1)$.

Case 2. For m_κ the components for $1 \leq i \leq k-(r+1)$ are given by $c_{im_\kappa} = C_i^{\gamma_{m_\kappa}}$. If $c_i = ab_i$ then $c_{im_\kappa} = (g^{a\gamma_{m_\kappa}})^{b_i} = H(m_\kappa)^{z_i}$, making the shares real. Otherwise, the c_i's are random, thus the shares are ideal.

Case 3. For all other messages $m \in \mathcal{M}_R$ the components for $1 \leq i \leq k-(r+1)$ are given by

$$c_{im} = B_i^{\gamma_m} = (g^{\gamma_m})^{b_i} = H(m)^{z_i},$$

as desired.

We now determined k components for every message m, the responses for senders $k, \ldots, n$, see Figure 5.1, are calculated from these by interpolating the exponents:

$$c_{im} = 1 \cdot \mathsf{enc}(1,m)^{\lambda_1^{\mathcal{I}}(i)} \cdots \mathsf{enc}(k-1,m)^{\lambda_{k-1}^{\mathcal{I}}(i)},$$

where $\mathcal{I} = \{0, \ldots, k-1\}$. We have now described how to answer encryption queries. During the challenge phase, we use precisely the same mechanism for constructing the components after selecting the challenge bit b.

Since the DDH problem is hard, two subsequent hybrid schemes are indistinguishable. Thus, the ideal scheme and the real scheme are indistinguishable as well. □

Using precisely the same reasoning as in the proof of Theorem 5.22, we obtain the following theorem of security of the batched scheme.

THEOREM 5.28 *In the random oracle model for h_2, the BKDE scheme is forward secure provided that the batched (non key-evolving) DE scheme and the evolving zero-sharing scheme are forward secure.*

5.6 ANALYSIS AND CONCLUSIONS

In this section, we first, informally, describe two modifications to our KDE scheme that might be useful in practice. Then, we compare the performance of the regular KDE scheme against that of the batched KDE scheme for two use cases.

Table 5.1: Performance comparison between the KDE and the batched KDE (BKDE) schemes. The table shows the number of exponentiations in $\mathbb{G}$ needed to recover all messages encrypted by k senders, and the number of ciphertext shares stored at the end of an epoch. Here m is the number of messages encrypted by each sender.

		Speed limiting	Canvas cutters
	Formula	$n = 2, \lvert\mathcal{M}\rvert = 10^7$ $k = 2, m = 600$	$n = 8, \lvert\mathcal{M}\rvert = 10^7$ $k = 4, m = 400$
KDE			
$\mathbb{G}$ exp.	$\binom{n}{k}m^k$	$1 \times 360 \cdot 10^3$	$70 \times 26 \cdot 10^9$
Storage	nm	1,200	3,200
BKDE			
$\mathbb{G}$ exp.	$\binom{n}{k}k\lvert\mathcal{M}\rvert$	$1 \times 10 \cdot 10^6$	$280 \times 10 \cdot 10^6$
Storage	$n\lvert\mathcal{M}\rvert$	$20 \cdot 10^6$	$80 \cdot 10^6$

5.6.1 *Practical considerations*

We propose two small extensions that can improve the KDE scheme in practice. Our KDE scheme works only with fixed-length messages. It is, however, possible to handle longer messages as well. First, append a hash of the message to authenticate the message as a whole. Then, split the message into appropriately sized chunks and run the DE scheme for each of them. After recovering the multiple blocks, combine them and check the hash before outputting the message. This procedural extension allows encrypting arbitrary length messages.

The downside of this procedural approach is that it duplicates the redundancy for every block. Another approach would be to change the redundant injective map itself to map longer messages to a tuple of group elements (and then raise each of these to the secret shares).

The second improvement deterministically encrypts the message with the public key of the combiner before running the DE scheme itself. This ensures that only the combiner—which is still assumed to be honest—can successfully recover the encrypted messages even if ciphertexts leak.

5.6.2 *Theoretical performance*

Table 5.1 shows the two methods' time complexity for recovering all messages that are encrypted by at least k different senders in terms of

group operations, and the space complexity in terms of stored ciphertext shares. A naive use of the standard combine operation in a DE scheme suggests that $\binom{n}{k}km^k$ exponentiations in $\mathbb{G}$ are required to recover all messages encrypted by k senders. However, for a given subset of k senders, each share is only raised to the power of a single Lagrange coefficient. Therefore, only $\binom{n}{k}m^k$ exponentiations in $\mathbb{G}$ are necessary.

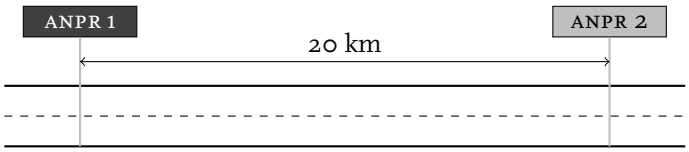
Table 5.1 also gives numbers for two specific use cases. In both, we assume the total number of vehicles is 10 million, like in the Netherlands. For the first, a speed-monitoring example, we choose the parameters to monitor a 20 kilometer stretch of highway—for simplicity, we assume there are no exits—with one ANPR system placed at the beginning, and one at the end. This situation is also illustrated by Figure 5.3. As an optimization, the first station only creates shares for the newest instance, this reduces the combining cost. We set the epoch length to 10 minutes. Every minute, we start a parallel instance of the system. This setting guarantees that every car that traverses this stretch of highway in 9 minutes or less—its speed is then at least $20/(9/60) \approx 133$ km/h—generates two shares in the same instance, and is thus always caught, while cars going between 120 km/h and 133 km/h may be caught. Using 20 parallel instances, instead of 10, will lower this bound to 126 km/h.

Suppose that 600 cars pass the ANPR systems during an epoch. The regular KDE scheme is more efficient in this setting due to the relatively small number of shares. In this setting, our key-evolution schemes ensure that the senders do not need to store $60 \cdot 24$ keys for every day the system is operational, instead they store only two.

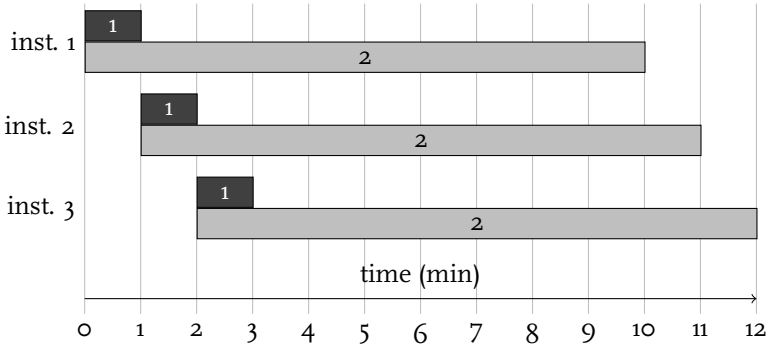
The second use case is the canvas cutters case we reintroduced at the start of this chapter. In this scenario, criminals frequently visit rest stops along a highway, cut open the canvas on lorries, and rob them. The criminals can typically be recognized by looking for cars that visit rest stops rather frequently. Suppose we monitor 8 rest stops, and consider a car suspicious if it visits at least 4 within a 4 hour period. Suppose 400 (different) cars visit each rest stop per period. Here, the BKDE scheme is clearly better. The exponential factor in the regular scheme quickly drives up the number of combines needed. In fact, this would be exacerbated if traffic increases. Nevertheless, these cases also illustrate that if storage is an issue, or fewer shares are expected, then it is better to use the non-batched scheme.

5.6.3 *Implementation*

To evaluate the performance of our two new schemes and to compare them against Hoepman and Galindo's scheme, we built and tested a prototype implementation for each of the three schemes in C using the



(a) Placement of ANPR stations along a stretch of highway



(b) Timeline of staggered DE instances

Figure 5.3: Figure (a) shows the placement of the ANPR systems in the speed-limiting example. Non-speeding cars need at least 10 minutes to traverse this stretch of highway. Figure (b) show the first three staggered distributed encryption instances. Every minute a new instance is started. Every instance runs for 10 minutes. The first ANPR station monitors for 1 minute, the second for the full 10 minutes. A speeding car that traverses the 20 kilometer stretch of highway in less than 9 minutes is always detected by both ANPR stations in one instance, and is thus caught. A car that does not speed is never seen by more than one ANPR station per instance.

Table 5.2: The running times of our prototype implementations of Hoepman and Galindo's DE scheme (HGDE), our new DE scheme (KDE) and our new batched DE scheme (BKDE) for each of the two use cases shown in Table 5.1. All numbers are on a single core.

Implementation	Speed limiting	Canvas cutters
HGDE	134 s	–
KDE	4 s	–
BKDE	190 s	35 h

RELIC cryptographic library [8].[17] We ran all experiments on a single core of an Intel i5-6200U running at 2.30 GHz. All code is trivially parallelizable. Both implementations recover all encrypted messages by trying all possible combinations of ciphertext shares.

As a small optimization, the implementations disregard shares for which they already found a corresponding message. This pruning helps speed up the combining process when there are many messages that can be recovered. Table 5.2 compares the performance of these implementations on the two use cases.

The difference in run time between Hoepman and Galindo's DE scheme and our new DE scheme is solely due to the extra pairing operation that Hoepman and Galindo's scheme needs. The run time of our new KDE scheme, however, is better than one would expect based on the number of exponentiations in $\mathbb{G}$. This is because for small values of n and k the Lagrange coefficients—which are used as exponents when combining ciphertexts—are decidedly non-random. We see that our new DE scheme is more than fast enough for the speed limiting example. It outperforms the batched DE scheme. Moreover, the batched scheme needs about 8 minutes to populate the initial share vector with random group elements.

The canvas cutters scenario remains much more challenging. The non-batched schemes simply require too much time. Based on the relative complexity of the two problems, our new non-batched DE scheme requires about 8 months to complete (not taking into account the fact that the Lagrange coefficients become less favourable in the canvas cutters scenario). While the 35 hours of our batched scheme is still a bit long, the results in Section 6.7 suggest that using a fully optimized curve gives a factor 2 speedup. Moreover, the scheme is also trivially parallelizable, bringing this just within the realm of practicality.

17 We again set up RELIC to use an optimized 254 bits BN curve. While we could have used a regular curve for our new schemes, we opted to use the same optimized setting—and thus used $\mathbb{G} = \mathbb{G}_1$ for our schemes.

5.6.4 *Conclusion*

In this chapter, we presented a new distributed encryption scheme that is simpler than previous solutions, and uses weaker assumptions. Furthermore, we described a key-evolving variant that offers proper key evolution and is therefore forward secure. Additionally, we demonstrated a batched variant of our new distributed encryption scheme that is much more efficient for small message domains.

None of the known distributed encryption schemes offer semantic security; senders always produce the same ciphertext for a given message. It would be very interesting to see solutions that use randomization to avoid this problem, although we suspect that this requires some kind of interactivity. Furthermore, it would be interesting to see if our distributed encryption schemes can be sped up to reduce their running times.

6

FAST REVOCATION OF ATTRIBUTE-BASED CREDENTIALS

More and more governments are issuing electronic identity (EID) cards to their citizens [103, 127, 129]. These EID cards can be used both offline and online for secure authentication with the government and sometimes with other parties, such as shops. Attribute-based credentials (ABCS) [34] are an emerging technology for implementing EID cards because of their flexibility and strong privacy guarantees, and because they can be fully implemented on smart cards [162]. Every credential contains attributes that the user can either reveal or keep hidden. Such attributes describe properties of a person, such as her name and age. As we saw in the introduction, ABCS enable a range of scenarios from fully-identifying to fully-anonymous. When using a credential fully anonymously (i.e., without revealing any identifying attributes), proper ABC technologies guarantee that the credential is unlinkable: it is not possible to connect multiple uses of the same credential.

When ABCS are applied, the carriers on which the credentials are stored (for example, smart cards) can be lost or stolen. In such cases, it is important that users can revoke these credentials to ensure that they can no longer be (ab)used. This is also necessary when the owner of the credential herself abuses it. Revocation may, in fact, happen often. As an example, the nationwide Belgian EID system's revocation list contains more than 375 000 credentials [33] for just over 10 million citizens. A practical revocation scheme must therefore efficiently deal with such large revocation lists.

Unfortunately, the unlinkability of ABCS precludes the use of standard, identity-based revocation. There exist many privacy-friendly revocation schemes, with different trade-offs in terms of efficiency (both for users and verifiers), connectivity requirements, and anonymity. It turns out to be hard to satisfy all of these simultaneously. All revocation schemes proposed so far suffer from at least one of the following two problems: (1) they rely on computationally powerful users, making the scheme unsuitable for smart cards, the obvious carrier for a national EID; or (2) they place a high load on verifiers, resulting in long transaction times.

The research presented in this chapter was part of the ongoing research project "I Reveal My Attributes" (IRMA).[1] The goal of this project is to demonstrate the practicality of attribute-based credentials. Within

1 See https://www.irmacard.org and https://privacybydesign.foundation/irma/.

the project, we implemented the entire user-side of the credentials on a smart card [162] and are currently working on secure implementations on smartphones to enable large key sizes and to provide a better user interface. In this chapter we focus on the smart card setting.

OUR CONTRIBUTION. Our contribution is a new revocation scheme that has very low computational cost for users and verifiers alike; it is efficient even in the smart card setting, and can therefore be used in practice. We introduce the main idea in Section 6.1. Adding revocation to an credential scheme, see Section 2.7 for a general introduction, introduces new parties and, in case of our revocation scheme, also extra requirements. We explain these in Section 6.2. We then describe the full revocation scheme in Section 6.3. In our scheme, verifiers need only constant time on average to check revocation status, making it as fast as traditional non-anonymous revocation schemes. Furthermore, the users' computational overhead is small (and updates by users to their own internal state to reflect revocations are *not* necessary). Our scheme is based on *epochs* that divide time into short (configurable) intervals. A user is unlinkable, except if she uses her credential more than once per epoch at the same verifier. We achieve this by generalizing the direct anonymous attestation (DAA) domain-specific pseudonyms. We model the security of our scheme and prove that our scheme is secure in this model—see Section 6.4. To mitigate the linkability within an epoch, we explore the idea of using multiple generators in Section 6.5. Our revocation scheme works with most credential schemes. As an example, we instantiate it for BBS+ credentials (see Section 2.7) in Section 6.6. In Section 6.7, we give pointers for implementing our scheme in practice, and give experimental results as evidence of the feasibility of our scheme. Finally, we review related work in Section 6.8 and conclude this chapter in Section 6.9.

6.1 THE IDEA

Our scheme enables efficient and privacy-friendly revocation of credentials. As our scheme resembles verifier-local revocation (VLR) schemes [9, 24, 28], we describe those first.

6.1.1 *Verifier-local revocation*

The mathematical setting is a cyclic group $\mathbb{G}$ with prime order p. Every credential encodes a random *revocation value* $r \in \mathbb{Z}_p$. If a credential has to be revoked, its revocation value r is added to the global revocation list RL. When the user shows the credential to a verifier, the verifier

needs to check whether the user's revocation value r appears on the revocation list RL. To facilitate this check without revealing r itself, the user chooses a random revocation generator $h \in_R \mathbb{G}$, calculates the *revocation token* $R = h^r$, and sends

$$(h, R) = (h, h^r) \tag{6.1}$$

to the verifier during showing. The user also proves that the revocation value r embedded into R corresponds to the credential she is showing. This proof depends on the type of credential—see Section 6.6 for an example. Each verifier holds a copy of the revocation list $RL = \{r_1, \ldots, r_k\}$. To check whether the credential is still valid, the verifier checks whether $h^{r_j} = R$ for each $r_j \in RL$ and rejects the credential if one of these equalities holds. This form of verifier-local revocation has some problems in practice:

1. Because the user chooses the revocation generator h at random, the work for the verifier increases linearly with the number of items on the revocation list. This quickly causes performance problems.

2. The scheme is not forward-secure. Once the verifier obtains a revocation value r, the verifier can link all past and future interactions involving this value, if it stores the tuples (h, R) from (6.1). Some solutions have been proposed to solve this problem—see Section 6.8—but they are not efficient enough for our purposes.

Our scheme addresses these two disadvantages.

6.1.2 *Our scheme*

We propose to split time into epochs and to use one generator per epoch and per verifier—this is a generalization of the direct anonymous attestation approach of creating domain specific pseudonyms based on a per-verifier generator, see Section 6.8. Using one per-epoch per-verifier generator limits the user to one showing per verifier per epoch if she wants to remain unlinkable (which is not a problem when epochs are small) but makes revocation checking very fast for the verifier. The user uses the per-epoch per-verifier generator $g_{\epsilon,V}$ to create the values in (6.1). In particular, she sends $R = g_{\epsilon,V}^r$ to the verifier.

To check whether the credential is revoked the verifier does not need to know the raw revocation values. Instead, a semi-trusted party, the revocation authority (RA), can store the raw revocation values of revoked users, and provide the verifier with a revocation list:

$$RL_{\epsilon,V} = \{g_{\epsilon,V}^{r_1}, \ldots, g_{\epsilon,V}^{r_k}\}.$$

The credential is revoked if $R \in RL_{\epsilon,V}$. This check takes only $O(1)$ time on average using associative arrays. The average time complexity thus decreases from linear to constant in the length of the revocation list $g_{\epsilon,V}$. While some computation load shifts to the RA, the RA does no more work creating the list than a verifier in the VLR scheme does for *every* verification. Also, the verifier can no longer link transactions in different epochs since it does not have the bare revocation values.

EPOCHS AND GENERATORS. The length of an epoch must be sufficiently short so that a user normally never shows her credential twice within the same epoch to the same verifier. If the epoch and verifier specific generator is reused, the corresponding activities of the user become linkable (but only within that epoch and at that verifier).

The generators form an attack vector for a malicious adversary to link users' activities. It is not sufficient for the user to keep track of the generators she used before. A malicious verifier could take one fixed generator $g_{\epsilon,V}$, and then create a new one by picking a random exponent $x \in_R \mathbb{Z}_p$ and sending $g_{\epsilon,V}^x$ to the user. All revocation tokens are then easily reduced to the same base $g_{\epsilon,V}$, without the user ever seeing a similar generator.

To prevent this attack, users should calculate the generators themselves. The easiest method—and the one we propose here—is to use a hash function and let the generator $g_{\epsilon,V}$ for a verifier V and epoch ϵ equal $H(\epsilon \parallel V)$, where H is a hash function from the strings to the group $\mathbb{G}$ (construction such hash functions is not always trivial, but can usually be done, also see Section 2.2.1) and the epoch ϵ is derived from the current time.[2]

HOW TO REVOKE A CREDENTIAL? In this chapter we consider two approaches to revoke a credential. Revocation is either *user-initiated* or *system-initiated*. When revocation is always user-initiated, only the owner can revoke a credential, and will do so when the credential (carrier) is lost or stolen. When revocation can also be system-initiated, the system can revoke a credential when the credential's owner loses the right to use that credential, e.g., when the owner does not pay for a subscription or when the owner abuses the credential. In our system, the choice boils down to the question: who knows the revocation value r?

When revocations are always user-initiated, only the owner needs to know the revocation value r. When her credential carrier is lost or stolen, she provides the revocation value r to the revocation authority. The revocation authority can then use this value to revoke the corresponding credential.

2 In Section 6.7.2 we explain how embedded devices like smart cards can keep track of time.

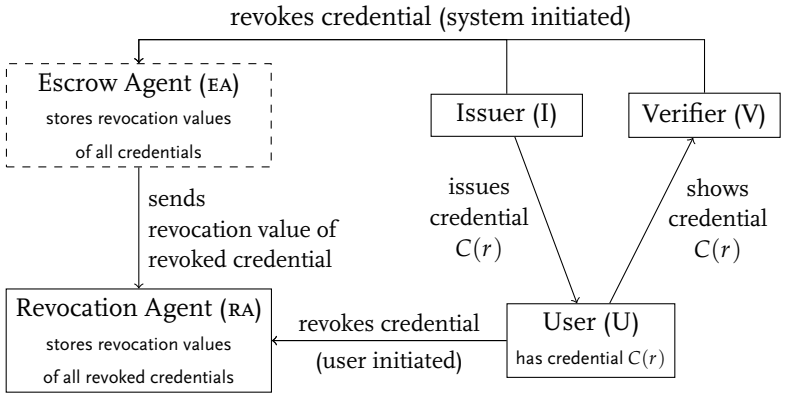


Figure 6.1: Interactions between parties in our revocation system. The escrow agent (EA) is only present when the system supports system-initiated revocation. There is typically only one revocation authority and one escrow agent, but there may be multiple issuers, verifiers and users.

When revocation can (also) be system-initiated, the revocation authority needs to be able to obtain the revocation value of revoked credentials. To this end, we introduce a trusted *escrow agent* that stores all the revocation values of all credentials. Then, when a party (for example, an issuer or a verifier) wants to revoke a credential, that party provides evidence to the escrow agent why it wants to revoke that credential. If the escrow agent accepts the evidence, it looks up the corresponding revocation value and sends it to the revocation authority. The revocation authority will use the revocation value as before to revoke the credential. In this case, no cooperation of the user is required. Also see Figure 6.1 for how the parties interact.

The downside of the system-initiated approach is that the escrow agent has a lot of power: it can use the revocation values to link a user's actions, even if her credential is not yet revoked. This is why we consider both approaches in tandem.

6.2 CREDENTIALS WITH REVOCATION

In this chapter we focus on revocation of credential schemes. We already introduced the necessary parties—users, issuers and verifiers—in Section 2.7. We repeat the description of the issuer, as its role is extended when the scheme is used with system-initiated revocation.

ISSUER The issuer issues credentials to users. It ensures that the correct data are stored in the credential. A typical credential scheme has multiple issuers. If system-initiated revocation is supported,

we assume that issuers internally assign an identifier *id* to each credential that they issue. This allows issuers to later refer to credentials when they want to revoke them. (This identifier is of course never revealed to a verifier.)

In a credential scheme with revocation, a revocation authority and, if system-initiated revocation is supported, an escrow agent, are also present. Figure 6.1 shows how the parties interact.

REVOCATION AUTHORITY The revocation authority is responsible for revoking credentials. It revokes credentials on request of users and, when present, the escrow agent. The revocation authority keeps track of all the revoked credentials. If necessary, it sends revocation information to users and verifiers.

ESCROW AGENT The escrow agent can initiate the revocation of credentials upon request of users (when they lack the information to revoke their credentials directly), verifiers and issuers. The escrow agent stores all information necessary to fulfil this task. If it decides to grant a revocation request, it sends the required information to the revocation authority.

Recall from Section 2.7 that we write $C(a_1, \ldots, a_L)$ to denote a credential over the L attributes $a_1, \ldots, a_L$. Our scheme is independent of the choice of credential scheme, but we impose three restrictions on it:

1. The credential must be able to encode a revocation value r from a sufficiently large set.[3] This value can be used to identify a revoked credential. We write $C(r)$ to denote a credential that contains the revocation value r. Depending on the type of credential, other attributes may be present.

2. The issuer should be able to issue a credential $C(r)$ without learning the revocation value r. Otherwise, the issuer can use it to trace credentials. Depending on whether the system supports system-initiated revocation, the revocation value is blindly supplied by the user or the escrow agent. Most credential schemes support the required forms of blind issuing. In Section 6.6 we show this for the BBS+ credential scheme.

3. The showing protocol must be extensible to provide the verifier with the revocation token $R = g_{\epsilon,V}^{r}$ and a proof that R and $C(r)$

3 For simplicity, we focus on attribute-based credentials, but this is not strictly necessary. Any credential scheme that can encode the revocation value and that satisfies the second restriction can be used with our scheme. One example would be to use the user's private key as the revocation value (although, in this case, extra care should be taken when using system-initiated revocation).

contain the same revocation value r. Fortunately, most credential schemes already rely on zero-knowledge proofs, and these can readily be extended to include the required proof of equality.

This chapter focuses on credentials that are multi-show unlinkable, i.e., a verifier cannot link multiple showings of the same credential. When credentials are only single-show unlinkable the situation is different. Single-show unlinkable credentials are either used multiple times and are then naturally linkable (and hence easier to revoke) or used only once to remain unlinkable. Our methods also apply to this latter case of single-show credentials that are used only once. To compensate, a user has multiple versions of the same credentials over the same attributes. If every version of the credential contains the same revocation value, then they can all be revoked simultaneously using our techniques.

6.3 THE FULL SCHEME

We now describe the full scheme. It expands on the intuition described in Section 6.1 by explicitly stating how the parties interact. Section 6.7 shows how to implement this scheme. Our scheme always supports user-initiated revocation. In addition, it can be configured to support system-initiated revocation.

The revocation authority runs the SetupRA algorithm once.

- SetupRA(1^ℓ). This algorithm takes as input a security parameter 1^ℓ. It chooses a cyclic group $\mathbb{G}$ of prime order p with generator g such that the DDH problem is hard in $\mathbb{G}$ and p has ℓ bits. Furthermore, it picks a hash function $H : \{0,1\}^* \to \mathbb{G}$ that maps strings onto this group. It outputs $(\mathbb{G}, g, p, H)$. These parameters are public and known to all other parties. The RA keeps track of the current epoch ϵ, which it initializes to 0, and the initially empty master revocation list MRL containing revoked credentials identified by their revocation values.

If system-initiated revocation is supported, the escrow agent is present. The escrow agent runs the following setup algorithm.

- SetupEA(). The escrow agent (EA) keeps track of a list RV containing a tuple (id, r_{id}) for each issued credential with identifier id storing the corresponding revocation value $r_{id} \in \mathbb{Z}_p$.

Users and verifiers run the algorithms SetupU and SetupV respectively.

- SetupU(). The user keeps track of the current epoch ϵ. She also stores sets $\mathcal{T}_C$ of the verifiers that she has shown credential C to in this epoch. Initially, $\mathcal{T}_C = \emptyset$.

- SetupV(). The verifier calls GetRevocationList to get an initial revocation list from the revocation authority—see below. It also keeps track of the current epoch ϵ.

At the beginning of a new epoch, all parties increase the current epoch ϵ by 1. In particular, we assume that all users know the current epoch.[4] At the start of a new epoch, users additionally clear the lists $\mathcal{T}_C$ of verifiers that have seen credential C in this epoch. Every verifier V runs the GetRevocationList protocol with the revocation authority to get its revocation list for the current epoch.

- GetRevocationList(). This protocol is run between a verifier V and the revocation authority. The parties execute the following steps:
  1. The verifier V identifies itself to the revocation authority.
  2. The revocation authority
     a) calculates the generator $g_{\epsilon,V} = H(\epsilon \parallel V) \in \mathbb{G}$ for verifier V;
     b) computes the sorted list
     $$RL_{\epsilon,V} = \mathsf{sort}(\{g_{\epsilon,V}^r \mid r \in \mathsf{MRL}\}); \text{ and}$$
     c) sends $RL_{\epsilon,V}$ to verifier V.

Sorting the revocation lists $RL_{\epsilon,V}$ ensures that unlinkability is preserved for all previous activities, even for revoked users (if $|\mathsf{MRL}| > 1$).[5]

To obtain a credential, the user and issuer run the ObtainCredential protocol. This protocol describes two variants, one for user-initiated revocation (in which case the user generates the revocation value r), and one for system-initiated revocation (in which case the EA generates r).

- ObtainCredential(I). This protocol is run between a user U wishing to obtain a credential, an issuer I, and, only if the system is configured to support system-initiated revocation, the escrow agent. The parties proceed as follows.
  - If the system is not configured with system-initiated revocation, the user U and issuer I run the normal issuance protocol. As part of this protocol, the user generates its revocation value $r \in_R \mathbb{Z}_p$, on which the issuer blindly issues a credential $C(r)$. This credential might contain other attributes, but

4 As we explain in Section 6.7.2, epochs are represented as time intervals. Users test their knowledge of the current time against this interval to make sure the interval is not in the past.

5 For this purpose, it suffices to sort on the representation of the elements. All that matters is that the order depends only on information in the list itself.

as we explained in Section 6.2, we omit these, as our focus is on the revocation scheme.

- If the system is configured with system-initiated revocation, the escrow agent generates the revocation value $r \in_R \mathbb{Z}_p$ for the credential, and sends it to the user. The user and the EA then jointly run the issuance protocol with the issuer. The EA blindly provides the revocation value r to the issuer, while the user provides the other attributes (if any) as in the normal issuance protocol to the issuer. Again, the user obtains the credential $C(r)$. Internally, the issuer assigns an identifier id to the credential, provides this identifier to the EA, and stores a record of the issuance process. The EA adds the tuple (id, r) to its list RV.

Most credential schemes can be extended to support the second configuration where the EA determines the revocation value. In Section 6.6 we show how to do so for BBS+ credentials.

If the user wishes to revoke one of her credentials with revocation value r, she runs the following protocol.

- UserInitiatedRevoke(r). When the revocation authority is asked to revoke a credential with revocation value r, it adds r to the master revocation list MRL.

If, on the other hand, another party in the system wishes to revoke a credential they can run the following protocol with the escrow agent.

- SystemInitiatedRevoke$(id, (R, \bar{g}), \text{evidence})$. The escrow agent can be asked to revoke a credential by supplying either the credential's identifier id or a corresponding revocation token R generated using generator $\bar{g}$. The caller additionally supplies some evidence evidence. The EA first examines evidence to determine if this is sufficient grounds for revocation. If the EA decides to grant the request, it recovers the revocation value r_{id} as follows.
  - If the credential identifier id is supplied, it looks up the pair $(id, r_{id}) \in$ RV to find r_{id}.
  - If the tuple $(R, \bar{g})$ is supplied, the EA iterates over $(id, r_{id}) \in$ RV to find the revocation value r_{id} such that $\bar{g}^{r_{id}} = R$.

  If the EA cannot find the revocation value it returns an error. Otherwise, it makes a UserInitiatedRevoke(r_{id}) request to the RA.

The idea of using the revocation token R to identify the corresponding credential is identical to the tracing mechanism provided by Boneh and Shacham [24]. In Section 6.7.1 we discuss some extra options for revoking credentials in practice.

To show a credential, the user and the verifier run the ShowCredential protocol. The user takes as input the verifier V and a credential. She sends the revocation token to the verifier and proves that she has a corresponding credential. The verifier checks the validity of the credential and whether it has been revoked.

- ShowCredential(C, V). This protocol is run by a user taking as input a credential C and a verifier V. The user interacts with a verifier.[6] The protocol proceeds as follows.
  1. The user aborts if $V \in \mathcal{T}_C$ (before contacting V).
  2. The user calculates the verifier and epoch specific generator $g_{\epsilon,V} = H(\epsilon \parallel V)$, and adds V to the list of seen verifiers $\mathcal{T}_C$.
  3. The user sends her revocation token $R = g_{\epsilon,V}^{r}$ to the verifier. Here, r is the revocation value encoded into the user's credential $C(r)$.
  4. The user and the verifier run the normal showing protocol for the user's credential $C(r)$, but in addition the user proves in zero-knowledge that her revocation token R is well-formed, i.e., that the exponent r is the same as the revocation value encoded in the credential. Section 6.6 shows an example of such a proof for BBS+ credentials.
  5. The verifier checks the validity of the credential and whether R is well-formed. Finally, it confirms that R is not on its revocation list $RL_{\epsilon,V}$ for the current epoch. It aborts if any of these checks fail.

The list $\mathcal{T}_C$ and the epoch ϵ uniquely determine the generators that the user has used for credential C in this epoch. The checks above ensure that the user never reuses a generator. Also, the user always calculates the generators herself. This prevents the verifier from cheating with the generators.

Checking that $R \notin RL_{\epsilon,V}$ can be done in constant time (on average) if the verifier processes the revocation list $RL_{\epsilon,V}$ into an associative array. Some tricks help keep the size of the revocation lists manageable—see Section 6.7.5.

6.4 SECURITY MODEL AND PROOFS

A good revocation scheme needs to satisfy two properties: (1) non-revoked credentials are still unlinkable, and (2) a revoked credential is

6 For our revocation scheme it is not necessary that the user authenticates the verifier, she relies on her own input of the verifier V to determine the generator. However, for a credential scheme in general, it is essential to authenticate the verifier, lest the user reveal attributes to a party to whom she did not want to reveal them.

no longer usable. In this section, we prove that our scheme has these two properties, which we call unlinkability and unavoidability.

For both security definitions, we use credentials that are indistinguishable from one another. Most credential schemes provide this type of unlinkability, as long as the revealed attributes (if any) are the same. For unavoidability, we also require that credentials are unforgeable, but all credential schemes that we know of satisfy this property.

6.4.1 *Unlinkability game*

We say that a revocation scheme is unlinkable if no adversary can win the following game. This game is very similar to the anonymity games defined for group signatures [24, 117] in general, and the backward unlinkable anonymity game of Nakanishi and Funabiki [124] in particular.

GAME 6.1 (The unlinkability game) In the unlinkability game, the adversary's goal is to determine which of two credentials is shown to him. Let $\mathcal{S}$ be a credential scheme, n the number of credentials in the system, and ℓ the security parameter.

SETUP PHASE To set up the system, the challenger runs $\mathsf{SetupRA}(1^\ell)$ on the RA and $\mathsf{SetupEA}()$ on the EA. It sets up the credential system $\mathcal{S}$ and initializes n users using $\mathsf{SetupU}()$. Then, it issues corresponding credentials $C_1, \ldots, C_n$ containing revocation values $r_1, \ldots, r_n$ using the ObtainCredential protocol to these users. Finally, it initializes the current epoch ϵ to 0. The adversary is responsible for setting up the verifiers.

QUERY PHASE The adversary may issue the following queries.

- $\mathsf{CorruptCredential}(i)$. The adversary can request credential C_i to be corrupted. It receives the revocation value r_i and the entire internal state of the credential.
- $\mathsf{Verify}(V, i)$. The adversary can request to act as verifier V for credential C_i in the $\mathsf{ShowCredential}(C_i, V)$ protocol.
- $\mathsf{Revoke}(i)$. The adversary can ask to revoke credential C_i. The challenger calls $\mathsf{UserInitiatedRevoke}(r_i)$ on the RA.[7]
- NextEpoch. The adversary requests to move to the next epoch. The challenger ensures that the revocation authority and the users move to the next epoch and updates its own epoch ϵ as well. The adversary is responsible for moving the epoch of the verifiers.

7 This oracle call also captures the abilities given by regular SystemInitiatedRevoke calls, as the adversary can always identify the credentials (except in the challenge phase).

The verifiers under the adversary's control can at any point in time make GetRevocationList queries to the RA.

CHALLENGE PHASE Let the current epoch be ϵ^*. The adversary selects two credentials with identifiers i_0 and i_1 and a verifier V^* such that linking is not trivial, i.e.,

1. neither credential i_0 nor i_1 was revoked in ϵ^* or earlier,
2. neither credential i_0 nor i_1 was corrupted, and
3. verifier V^* did not verify the credentials i_0 and i_1 during epoch ϵ^*.

The challenger then picks a bit $b \in \{0,1\}$ at random and runs $\mathsf{Verify}(V^*, i_b)$ (as defined above) with the adversary.

RESTRICTED QUERIES PHASE The adversary can make CorruptCredential, Verify, Revoke and NextEpoch queries as in the query phase. However, the adversary is not allowed to call CorruptCredential on the challenge credentials i_0 and i_1. Furthermore, during the challenge epoch ϵ^*, the adversary is not allowed to revoke the challenge credentials i_0 and i_1 nor make $\mathsf{Verify}(V^*, i_0)$ and $\mathsf{Verify}(V^*, i_1)$ queries on them.

OUTPUT PHASE The adversary outputs a bit b'. It wins if $b = b'$.

The advantage of an adversary $\mathcal{A}$ for a credential system $\mathcal{S}$ is given by $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{LINK}} = 2\left|\Pr[b = b'] - \frac{1}{2}\right|$, where the probability is over the random bits of the challenger and the adversary. The revocation scheme for credential scheme $\mathcal{S}$ is unlinkable if $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{LINK}}$ is negligible for every probabilistic polynomial time (PPT) algorithm $\mathcal{A}$.

This game models the fact that the revocation authority is trusted; it will not give the raw revocation values to the adversary. The game also models the forward security of the scheme. The adversary is allowed to revoke the challenge credentials i_0 and i_1 in epochs beyond ϵ^*, but should nevertheless have a negligible advantage in distinguishing the credentials in epoch ϵ^*.

In the following reduction we let the revocation token R of the challenge users depend on a DDH instance. As a result, we do not know its discrete logarithm, so we cannot create the equality proof required in the protocol. Instead, we require that we can forge this proof in the reduction. In most applications, this proof will be a non-interactive zero-knowledge proof resulting from the Fiat-Shamir heuristic [64]. In this case, our reduction can forge these proofs assuming a random oracle. (For regular zero-knowledge proofs we can use rewinding techniques.)

THEOREM 6.2 *Our credential scheme with our revocation scheme is unlinkable (in the sense of Game 6.1) in the random oracle model provided that the* DDH *problem is hard in the group* $\mathbb{G}$ *and the underlying credential system is unlinkable.*

Proof. We reduce the security of our revocation scheme to the hardness of the DDH problem. We encode the DDH instance into the revocation token of two specific users. We do this in such a way that in the challenge epoch ϵ^* we give the correct token if $z = xy$ and a random one otherwise. Any distinguisher thus breaks DDH.

More precisely, we construct a new game, which we call the random-challenge-token game, in which the revocation token $R = g_{\epsilon^*,V^*}^{r_{i_b}}$ of the challenge user i_b (picked by the challenger) is replaced by a random token $R \in_R \mathbb{Z}_p$. Our claim is that no adversary can distinguish the normal unlinkability game from this random-challenge-token game.

If this claim holds, we can replace the revocation token in the challenge of the regular indistinguishability game with a random token without being detected. Clearly, if the revocation token is random, the adversary has no extra information with respect to the regular unlinkability game for credentials. Therefore, since the credentials themselves are unlinkable, the adversary has, by definition, no chance of winning the random-challenge-token game. As a result, the adversary also cannot win the unlinkability game for credential schemes extended with our revocation mechanism.

We now prove that no adversary $\mathcal{A}$ can distinguish the normal unlinkability game from the random-challenge-token game. Suppose such an adversary $\mathcal{A}$ does exist. We will construct a challenger $\mathcal{B}$ that breaks the DDH assumption. As input, the challenger $\mathcal{B}$ takes a DDH problem $(g, X = g^x, Y = g^y, Z = g^z)$, and $\mathcal{B}$'s task is to determine whether $z = xy$ or $z \in_R \mathbb{Z}_p$.

First, challenger $\mathcal{B}$ guesses the challenge epoch ϵ^*, the challenge verifier V^* and the special credentials i_0 and i_1. Challenger $\mathcal{B}$ generates $n-2$ revocation values $r_i \in_R \mathbb{Z}_p$ for all $i \neq i_0, i_1$ and issues $n-2$ credentials $C(r_i)$ corresponding to these values. For credentials i_0, i_1, it picks random values $a_0, a_1, c_0, c_1 \in_R \mathbb{Z}_p$, and it will act as if credential i_b's revocation value r_{i_b} equals $x \cdot a_b$ for $b \in \{0,1\}$. It also creates corresponding credentials $C(c_0)$ and $C(c_1)$. It does not know r_{i_0} or r_{i_1} themselves, and since $c_b \neq r_{i_b}$, challenger $\mathcal{B}$ always needs to fake the equality proofs involving $C(c_b)$.

Note that the challenger can answer almost all queries honestly. The only changes that we make are how it computes revocation tokens for users i_0 and i_1, and how it answers hash and challenge queries.

The challenger changes the random oracle H to choose the generators $g_{\epsilon,V}$. A judicious choice of the generators makes it possible to

create the revocation token for credentials i_0, i_1, even though x is unknown. For every epoch ϵ and verifier V the challenger chooses an exponent $e_{\epsilon,V} \in_R \mathbb{Z}_p$. For verifier $V = V^*$ in epoch $\epsilon = \epsilon^*$ the challenger sets $g_{\epsilon^*,V^*} = H(\epsilon^* \parallel V^*) := Y^{e_{\epsilon^*,V^*}}$. For all other ϵ, V pairs, it sets $g_{\epsilon,V} = H(\epsilon \parallel V) := g^{e_{\epsilon,V}}$. The challenger runs the adversary $\mathcal{A}$, and honestly answers all queries not involving credentials i_0, i_1. For credential $i_d, d \in \{0,1\}$ it creates the revocation token as

$$R_{\epsilon,V,i_d} = g_{\epsilon,V}^{r_{i_d}} = (g^{e_{\epsilon,V}})^{x \cdot a_d} = (g^x)^{e_{\epsilon,V} \cdot a_d} = X^{e_{\epsilon,V} \cdot a_d}$$

unless $V = V^*$ and $\epsilon = \epsilon^*$. For $d \in \{0,1\}$, if the adversary ever makes a $\mathsf{Verify}(V^*, i_d)$ query in epoch ϵ^*, corrupts credential i_d, or revokes credential i_d at or before epoch ϵ^*[8] challenger $\mathcal{B}$ aborts. In all cases, the challenger forges the proof of equality of the revocation value and credential $C(c_d)$ using its random oracle.

Eventually, the adversary makes its challenge query for credentials $\hat{i}_0, \hat{i}_1$ at verifier V in epoch ϵ. If $\mathcal{B}$ did not guess these correctly, i.e., if $\epsilon^* \neq \epsilon$, $\{\hat{i}_0, \hat{i}_1\} \neq \{i_0, i_1\}$, or $V^* \neq V$, it aborts. Otherwise, it picks a bit $b \in_R \{0,1\}$ and answers with $R = Z^{e_{\epsilon^*,V^*} \cdot a_b}$ (and forges the corresponding equality proof). If $z = xy$ then R belongs to credential i_b (because then $Z = Y^x$) and if $z \in_R \mathbb{Z}_p$ then R is random. After the challenge query, $\mathcal{B}$ answers the restricted queries as before.

If $\mathcal{A}$, at the end of the game, indicates that it plays the normal unlinkability game, then $\mathcal{B}$ answers 1 to indicate that $z = xy$ and if $\mathcal{A}$ indicates that it plays the random-challenge-token game, then $\mathcal{B}$ answers 0 to indicate that $z \neq xy$. Any non-negligible advantage that $\mathcal{A}$ has in distinguishing the two games results in challenger $\mathcal{B}$ having a non-negligible advantage for solving the DDH problem. □

6.4.2 *Unavoidability game*

A revoked credential should no longer be usable in the ShowCredential protocol. This requirement is expressed by the unavoidability game. As this notion is very close to traceability in group signatures (see for example Manulis et al. [117]), we express it using a similar game.

The idea of the game is that we revoke every credential that the adversary obtains. The goal of the adversary is to use these credentials to sidestep the revocation mechanism. The adversary has two options to obtain a credential: it can corrupt an honest user's credential or it can request a credential to be issued to him. In the former situation, the game models a call to UserInitiatedRevoke to model that the honest user revokes his credential. In the latter case, we *must* use

8 Even though we do not know r_{i_d} we can still revoke credential i_d in later epochs because we *can* calculate the revocation token R_{ϵ,V,i_d} as shown.

system-initiated revocation (a malicious user cannot be trusted to voluntarily revoke its credentials). To model this, the game makes a call to SystemInitiatedRevoke. (As a direct consequence of this, if the system does not support system-initiated revocation, then the adversary is not allowed to obtain any new credentials in this game.)

GAME 6.3 (The unavoidability game) In the unavoidability game, the adversary's goal is to convince the challenger's verifier that it has a valid and unrevoked credential while in fact all credentials it has *are* revoked. Let $\mathcal{S}$ be a credential system, ℓ the security parameter and n the number of honest users' credentials in the system.

SETUP PHASE To set up the system, the challenger runs $\mathsf{SetupRA}(1^\ell)$ on the RA and $\mathsf{SetupEA}()$ on the EA. It also sets up an issuer I for credential system $\mathcal{S}$ to construct credentials and a verifier for its own use. The challenger sets up the honest users by running $\mathsf{SetupU}()$ for each of them. The challenger also runs the protocol $\mathsf{ObtainCredential}(I)$ on behalf of each honest user i, so that they obtain a credential C_i with revocation value r_i. The adversary is responsible for setting up the users it controls.

QUERY PHASE The adversary may issue the following queries.

- $\mathsf{CorruptCredential}(i)$. The adversary can request the corruption of credential C_i. Then, r_i is the revocation value corresponding to this credential. The challenger sends the complete credential C_i to the adversary. Thereafter, the challenger calls $\mathsf{UserInitiatedRevoke}(r_i)$ to revoke the credential.
- $\mathsf{ObtainCredential}(id)$. The adversary can request a credential with identifier id to be issued to a user it controls. To this end, the challenger lets the adversary run the system-initiated revocation variant of the $\mathsf{ObtainCredential}(I)$ protocol with the escrow agent and the issuer (the challenger controls the latter two entities). At the end of this protocol, the adversary's user now has a new credential $C(r)$. The challenger immediately makes a $\mathsf{SystemInitiatedRevoke}(id, \bot, \bot)$ call to the EA to revoke this new credential (for the purpose of this game, the escrow agent accepts $\bot$ as evidence).
- $\mathsf{Verify}(V, i)$. The adversary can request to act as verifier V for credential C_i in the $\mathsf{ShowCredential}(C_i, V)$ protocol.
- NextEpoch. The adversary requests to move to the next epoch. The challenger ensures that the revocation authority, the escrow agent, and the users it controls move to the next epoch and updates its own epoch ϵ as well. The adversary

is responsible for moving the epoch of the verifiers and the users it has created or corrupted.

CHALLENGE PHASE The adversary picks a verifier V. The challenger sets verifier V to the current epoch, and subsequently runs the GetRevocationList algorithm to get the latest revocation list. The adversary will run the ShowCredential protocol with this verifier. The adversary wins if the verifier accepts.

The advantage of adversary $\mathcal{A}$ for a given credential scheme $\mathcal{S}$ is given by $\mathsf{Adv}_{\mathcal{A}}^{\text{AVOID}} = \Pr[V \text{ accepts}]$, where the probability is over the random bits of the challenger and the adversary. The revocation scheme for credential scheme $\mathcal{S}$ is unavoidable if $\mathsf{Adv}_{\mathcal{A}}^{\text{AVOID}}$ is negligible for every PPT algorithm $\mathcal{A}$.

For simplicity we do not give the challenger the option to revoke credentials nor to obtain the revocation list since all this information is already encoded into its credentials.

THEOREM 6.4 *Our revocation scheme is unavoidable (see Game 6.3) in the random oracle model provided that the underlying credential scheme is unforgeable.*

We first provide a sketch of this proof.

Sketch. Suppose the challenger's verifier accepts the credential $C(r)$ that is shown by the adversary. Since credentials are unforgeable, the adversary obtained this credential using either a ObtainCredential or a CorruptCredential query, thus the embedded revocation token r is on the master revocation list MRL. Let g_V be the verifier's generator. The equality proof guarantees that the revocation token R presented by the adversary is of the form g_V^r. Since g_V^r is on the verifier's revocation list, the verifier will never accept the adversary's proof. □

To give a more formal security proof, we first define the unforgeability of a credential scheme.

GAME 6.5 (The unforgeability game) In the unforgeability game for a credential system $\mathcal{S}$, the adversary's goal is to show possession of a credential with attributes that were not previously issued by the issuer. Let ℓ be the security parameter and L the number of attributes contained within each credential.

SETUP PHASE The challenger sets up the system by setting up the credential system $\mathcal{S}$ and a corresponding issuer I.

QUERY PHASE The adversary may issue the following queries.

- $\mathsf{ObtainCredential}(a_1, \ldots, a_L)$. The adversary can request a credential on the tuple of attributes $(a_1, \ldots, a_L)$. The challenger engages in the issue protocol with the adversary to issue to the adversary a credential with these attributes. The challenger controls the issuer I.
- $\mathsf{VerifyCredential}(a_1, \ldots, a_L)$. The adversary can also request that a credential with attributes $(a_1, \ldots, a_L)$ is shown to him. To do so, the challenger issues himself a credential $C(a_1, \ldots, a_L)$, and runs the showing protocol for this credential with the adversary acting as verifier.

CHALLENGE PHASE The challenger sets up a verifier V and engages in the showing protocol with the adversary. Let $\mathcal{D} \subset \{1, \ldots, L\}$ be the indices of the attributes that were disclosed by the adversary. The adversary wins if the verifier accepts and the adversary never made a $\mathsf{ObtainCredential}(a'_1, \ldots, a'_L)$ query where $a'_i = a_i$ for all $i \in \mathcal{D}$ (note that if $\mathcal{D} = \emptyset$, this implies that the adversary did not make any ObtainCredential queries).

The advantage of adversary $\mathcal{A}$ is given by $\mathsf{Adv}^{\mathrm{FORGE}}_{\mathcal{A}} = \Pr[\mathcal{A} \text{ wins}]$, where the probability is over the random bits of the challenger and the adversary. The credential scheme $\mathcal{S}$ is unforgeable if $\mathsf{Adv}^{\mathrm{FORGE}}_{\mathcal{A}}$ is negligible for every PPT algorithm $\mathcal{A}$.

We can now formally prove Theorem 6.4.

Proof. Assume that an adversary $\mathcal{A}$ can break the unavoidability of our revocation scheme as applied to a credential system $\mathcal{S}$. We show how to construct an adversary $\mathcal{B}$ that breaks the unforgeability of credentials in the credential system $\mathcal{S}$.

Adversary $\mathcal{B}$ plays the role of challenger in the unavoidability game, and the role of adversary in the unforgeability game for the credential system. Adversary $\mathcal{B}$ runs the setup and query phases of the unavoidability game honestly. However, it does not construct its own issuer, instead it makes use of its oracle access to the unforgeability challenger. In particular:

- Adversary $\mathcal{B}$ picks revocation values r_i for each honest user i during the setup phase, but does not request credentials for them.
- Whenever the adversary $\mathcal{A}$ makes a $\mathsf{Verify}(V, i)$ query, adversary $\mathcal{B}$ makes use of its $\mathsf{VerifyCredential}(r_i)$ oracle to respond to the request.
- If adversary $\mathcal{A}$ makes a $\mathsf{CorruptCredential}(i)$ query, adversary $\mathcal{B}$ first makes an $\mathsf{ObtainCredential}(r_i)$ query to its oracles to

obtain a credential $C(r_i)$. It then sends this credential to adversary $\mathcal{A}$. As part of the game, adversary $\mathcal{B}$ also makes a UserInitiatedRevoke(r_i) call.

- If adversary $\mathcal{A}$ makes a ObtainCredential query, adversary $\mathcal{B}$ first generates a revocation value r (acting as the EA) and then makes a ObtainCredential(r) call to its challenger and relays the messages to $\mathcal{A}$ as in the system-initiated variant of the ObtainCredential protocol. Let *id* be the identifier of this credential, then $\mathcal{B}$ makes a SystemInitiatedRevoke$(id, \perp, \perp)$ call to revoke it.

During the challenge phase, adversary $\mathcal{A}$ runs the ShowCredential protocol with $\mathcal{B}$. Assume that $\mathcal{A}$ wins, i.e., $\mathcal{B}$ accepts the credential that is shown to him (if $\mathcal{A}$ loses, adversary $\mathcal{B}$ aborts). Since $\mathcal{B}$ accepts, the proof of knowledge produced by $\mathcal{A}$ shows that there is a revocation value r such that (1) the credential $C(r)$ shown by $\mathcal{A}$ is valid and (2) the revocation token R is of the form $R = g_{\epsilon^*,V}^{r}$. Finally, since $\mathcal{B}$ accepts, $R \notin RL_{\epsilon,V}$.

Since $R \notin RL_{\epsilon,V}$, $\mathcal{B}$ never made a request to issue a credential with r as attribute (every time it made an ObtainCredential request, the corresponding revocation value was added to the MRL as a result of a subsequent revocation call). Hence, the credential $C(r)$ is a forgery.

To use this credential as a forgery, adversary $\mathcal{B}$ runs the knowledge extractor on the proof of knowledge of $\mathcal{A}$ to obtain the credential $C()$ and the revocation value r. Adversary $\mathcal{B}$ then shows this credential, with the revocation value r as part of the disclosed attributes, to its challenger. Since r was never issued as part of a credential, adversary $\mathcal{B}$ wins the unforgeability game.

To conclude, if an adversary breaks the unavoidability of the revocation scheme, then it must also break the unforgeability of the underlying credential system. □

6.5 MULTIPLE GENERATORS

The single generator protocol we described above is secure and efficient. Yet, a user is linkable in exceptional cases: when she uses a credential multiple times for the same verifier within one epoch (which is then too long). Also, the load on the revocation authority can become quite high (it needs to create a revocation list for each verifier). In this section, we make a detour to explore the question whether multiple generators—shared among the verifiers,[9] or even per verifier—alleviate these minor problems. The answer is positive, however, using multiple generators

9 This approach using global generators is similar to the starting point taken in the revocation technique by Verheul [161].

can make the user somewhat linkable (but less linkable than if the user uses the same credential multiple times within an epoch in the original scheme). Before we explain this linkability, we extend our scheme with multiple generators.

6.5.1 *Multiple generators for revocation*

We propose two methods for creating multiple generators: the global and the local. In the global method there are m generators (per epoch) that are shared among the verifiers. In the local method there are m generators (per epoch) for each verifier individually. In the latter case m can be smaller than in the former.

Global generators ensure that the user has m different generators to choose from. These can be spent at any verifier, several different generators can even be used at the same verifier. If m is smaller than the number of verifiers, using global generators reduces the load on the revocation authority as well. Local generators give the user m generators per verifier instead of only one.

Instead of generating per-epoch per-verifier generators, we now create the ith generator, with $1 \leq i \leq m$ as follows:

$$g_{\epsilon,i} = \begin{cases} H(\epsilon \parallel i) & \text{if mode is global} \\ H(\epsilon \parallel V \parallel i) & \text{if mode is local.} \end{cases}$$

The verifier is implicit for local generators. The verifier can now request revocation lists for each of these generators (for global generators the RA will cache the responses). During showing, a user randomly picks one of the unused generators (or aborts if she cannot). To this end, she keeps track of the indexes of generators used in this epoch for global generators, or pairs (V, i) when she used the ith verifier of verifier V. She informs the verifier of her generator choice, and proves that she created her revocation token R with respect to this generator.

This method was inspired by ideas for traceable signatures [48], where signatures can be traced because the signer can only produce a finite number of unique tags. When tracing a signature the tracing agent produces all these tags, much like we generate revocation tokens for all the generators. The traceable signature scheme does not suffer from the problem we describe next because the (inefficient) proofs of knowledge hide which generators are used.

6.5.2 *Distinguishing credentials*

The unlinkability game is easily extended to the above setting. We first note some positive results. For local generators with $m = 1$ we exactly

have the same scheme as before, with the same security requirements. Similarly, if two credentials have never been used in this epoch (at this verifier, for local generators), it can be shown that the adversary has no chance of linking them.

However, when a credential is used multiple times, before the challenge phase, the user creates an internal state—the generators that she has already used—that can be recognized by the verifier with a non-negligible probability. This attack works independent of the mode.

The verifier only needs two credentials C_0 and C_1 to have an advantage in the unlinkability game. It makes $m-1$ verify queries to C_0. It can observe which generators C_0 chooses, let $\tilde{g}$ be the generator it did *not* yet use. The adversary makes no queries to C_1. In the challenge phase it again requests credentials C_0 and C_1. If the credential uses generator $\tilde{g}$ the adversary guesses it is communicating with C_0, otherwise it guesses C_1. Since C_0 always uses $\tilde{g}$ and C_1 uses it with probability $1/m$ the adversary is correct with probability $1-1/(2m)$. Note that this attack does not work if $m=1$. A similar attack works for any two credentials for which the internal state $\mathcal{T}_C$ differs.

While it is not nice to have credentials that are linkable in this way, the effect of this attack—that does not identify credentials directly, nor makes them fully linkable—may be acceptable to either significantly reduce the load on the RA for global generators, or allow the options of multiple authentications with the same verifier within an epoch at a small loss of privacy. (The original scheme makes the credentials fully linkable in this case.)

6.5.3 *Making multiple generators work*

The essential difficulty in the multiple-generator scheme we sketched above is that the user reveals the generator she used. This is not necessary. Instead, given a set of generators $g_{\epsilon,1},\ldots,g_{\epsilon,m}$ the user with revocation value r can make a zero-knowledge proof that

$$R = g_{\epsilon,1}^r \vee R = g_{\epsilon,2}^r \vee \cdots \vee R = g_{\epsilon,m}^r.$$

The verifier checks if R is on any of its revocation lists (corresponding to the m generators). It can be shown that this variant is secure.

This zero-knowledge proof is not complicated, but it is computationally intensive for the user: its complexity is $O(m)$. However, it provides us with a trade-off between efficiency and perfect unlinkability. Moreover, when m is small, we still outperform other fast solutions (such as accumulators, see Section 6.8) without requiring updates to the user.

6.6 INTEGRATING OUR SCHEME WITH BBS+ CREDENTIALS

In this section we explicitly show how our revocation scheme can be integrated into a credential system using BBS+ signatures, as introduced in Section 2.7.1. We can easily generalize these BBS+ signature to form credentials over the attributes $(a_1, \ldots, a_L)$, the user's private key x and the revocation value r. This credential is given by the signature (A, e, s) where $e, s \in \mathbb{Z}_p$ and

$$A = \left(g B_0^s B_U^x B_R^r \prod_{i=1}^{L} B_i^{a_i} \right)^{\frac{1}{e+\gamma}} \in \mathbb{G}_1,$$

γ is the signer's private key, and $\Delta = (w = h_0^\gamma, B_U, B_R, B_0, \ldots, B_L)$ is the public key.

To incorporate the revocation mechanism, we first describe how a credential is issued, and then how it is verified. As described in Section 6.3, the system can provide user-initiated and system-initiated revocation. In the former case, the user embeds the revocation value in the credential, while in the latter case, the escrow agent does. We describe both.

Issuance in the case of user-initiated revocation

In a system in which only the user knows the revocation value r, the revocation value is blindly signed by the issuer, in the same way that the issuer already signed the blinded secret key sk_U. To highlight the difference with the system-initiated variant we recall the regular BBS.Issue protocol, see Section 2.7.1, for this particular instance. The user first commits to r, sk_U, and the other hidden attributes in $\mathcal{H}$:

$$C = B_0^{s'} B_U^x B_R^r \prod_{i \in \mathcal{H}} B_i^{a_i}$$

using a random $s' \in \mathbb{Z}_p$, and sends it to the issuer along with a proof of knowledge of the exponents $s', x, \{a_i\}_{i \in \mathcal{H}}$. If the proof verifies, the issuer randomly generates $s'', e \in_R \mathbb{Z}_p$, sets

$$A = \left(g B_0^{s''} C \prod_{i \in \mathcal{D}} B_i^{a_i} \right)^{\frac{1}{e+\gamma}} \in \mathbb{G}_1,$$

and returns the transient signature (A, e, s'') to the user. The user calculates $s = s' + s''$ and stores the signature $\sigma = (A, e, s)$.

Issuance in the case of system-initiated revocation

In the case of system-initiated revocation, the escrow agent generates the revocation value and supplies it to the issuer. Moreover, the EA and the user share the generation of the random value $s' = s'_{\text{EA}} + s'_U$. First, both the EA and the user create a commitment: the EA commits to the revocation value r using

$$C_{\text{EA}} = B_0^{s'_{\text{EA}}} B_R^r,$$

while the user commits to the secret key sk_U and the hidden attributes using

$$C_U = B_0^{s'_U} B_U^x \prod_{i \in \mathcal{H}} B_i^{a_i}.$$

They send their commitment C_{EA} and C_U to the issuer together with the corresponding zero-knowledge proofs of knowledge via a secure and authenticated channel (this allows the issuer to verify that the EA participated in the protocol). Furthermore, the EA sends the exponents s'_{EA}, r to the user. Similar to the other case, the issuer constructs the transient signature (A, e, s'') by computing

$$A = \left(g B_0^{s''} C_{\text{EA}} C_U \prod_{i \in \mathcal{D}} B_i^{a_i} \right)^{\frac{1}{e+\gamma}} \in \mathbb{G}_1$$

with random values $s'', e \in_R \mathbb{Z}_p$. Finally, upon receiving (A, e, s'') with the corresponding proof of knowledge, the user constructs the signature (A, e, s) by computing $s = s'_{\text{EA}} + s'_U + s''$.

Verification

We can easily integrate our revocation scheme in the selective disclosure proof of such a credential (A, e, s). Suppose the user wants to prove that she has a credential containing the disclosed attributes $(a_i)_{i \in \mathcal{D}}$. She can then construct the following proof of knowledge:

$$\text{PK}\{(A, e, x, r, (a_i)_{i \in \mathcal{H}}, s) \ : \\ A^{e+\gamma} = g B_0^s B_U^x B_R^r \prod_{i \in \mathcal{H}} B_i^{a_i} \prod_{i \in \mathcal{D}} B_i^{a_i} \wedge R = g_{\epsilon,V}^r\}.$$

See Section 2.7.1 for how to instantiate the first conjunct of this proof.

6.7 IMPLEMENTATION

We now address some implementation challenges when using our revocation scheme.

6.7.1 *How to revoke a credential*

In the preceding part of this chapter we considered two options for revoking credentials: user-initiated and system-initiated. In this section we revisit these options.

To revoke a credential one needs to know its revocation value. However, this value also poses a privacy risk: the party that stores it could revoke the credential and hence detect its use. Many revocation schemes suffer from the same problem, see Section 6.8. Clearly, user-initiated revocation offers better privacy, as the user controls who she gives the revocation value to. However, she loses this control when she wants to revoke her credential.

More privacy for user-initiated revocation

If the user does not trust the RA to only use her revocation value r to revoke future uses of her credential, she can instead adopt the following approach. Rather than sending the raw revocation value r to the RA, the user calculates the revocation tokens herself for all verifiers and all remaining epochs in which the credential is valid (assuming credentials expire)—she uses the fact that the generators can be calculated in advance.

This is costly, but does give forward-privacy for the user without trusting the RA. To reduce this cost, the RA can add structure to the generators of a single epoch as follows. It picks $z_{\epsilon,V} \in_R \mathbb{Z}_p$ and sets $g_{\epsilon,V} = H(\epsilon)^{z_{\epsilon,V}}$. The user only needs to do one exponentiation per epoch—she calculates $H(\epsilon)^r$—and the RA creates the per-verifier specific values (as $R_V^r = (H(\epsilon)^r)^{z_{\epsilon,V}}$). The user is no longer able to check the verifier specific generators herself (she does not know $z_{\epsilon,V}$), instead, the RA issues certificates on the verifiers' generators.[10] To ensure that the user does not reuse generators (verifiers might collude and provide each other's signed generators to the user), the user stores the used, verified generators instead.

Managing revocation values

If the system uses only user-initiated revocation, users need to store their revocation values so that they are available in the case the credential needs to be revoked. This is not the case when the credentials (and thus the revocation values) are stored on a smart card. So, when the card is lost or stolen, the revocation values needed to revoke the credentials

10 To prevent the RA from being able to link credentials across epochs (this specific construction of the generators already allow it to do so within epochs), the certificate should also include a zero-knowledge proof that the RA knows $z_{\epsilon,V}$ such that $g_{\epsilon,V} = H(\epsilon)^{z_{\epsilon,V}}$.

need to be available elsewhere. One option would be to use a trusted terminal to print the (card-generated) revocation values (for example as a QR code) when a credential is issued. The user can then store the revocation values separately from the card. Note that if system-initiated revocation is enabled, the user can always use that avenue to revoke her own credentials.

Escrowing revocation information

An alternative to storing all revocation values at the escrow agent, is to escrow the revocation information during the showing protocol. Similar to identity escrow [95], the user provides a verifiable encryption of her revocation value (encrypted with the public key of the escrow agent) to the verifier. While this does not protect against lost or stolen credentials, it does allow verifiers to ask for revocation of a credential when it can present sufficient grounds to do so. Since the user attaches the encrypted revocation value to each showing, the escrow agent does not need to store the revocation values anymore.

However, the escrow agent can still identify a user based on the encrypted revocation value, just like it can for our system-initiated revocation approach by checking the revocation token against the list of revocation values. Therefore, we do not gain any security, and lose efficiency, because verifiable encryption is computationally intensive.

Once again, using an escrow agent trades privacy of the user for a better security of the system as a whole. It depends on the application of the system whether trusting the escrow agent is better than simply allowing some abuse.

6.7.2 *Instantiating epochs*

To keep the protocol description simple, we assumed that all parties are aware of the current epoch. To achieve this, epochs are, in practice, based on time. The revocation authority determines the length of an epoch, by specifying its start time t_s and end time t_e, so the current epoch ϵ is modelled by the tuple $\epsilon = (t_s, t_e)$.

In step 2 of the ShowCredential protocol, the user checks that $t_s \leq t \leq t_e$ where t is the current time. If this equation is not correct, the user aborts. In this way, users always use the correct generator.

Embedded devices

The above description does not suffice for smart cards, our target platform, as they lack a built-in clock, and thus have no notion of time. Nev-

ertheless, an embedded device must also be able to calculate the generators itself, to prevent a verifier from adversarially choosing them.

We propose the following solution, similar to the method used in Machine Readable Travel Documents, such as the new European passport [29]. The embedded device keeps track of an estimate t^* of the current time. The estimate is always at or before the current time. Every time the embedded device interacts with a verifier, it

1. receives a description of the current epoch (t_s, t_e) signed by the RA;
2. confirms that the epoch (t_s, t_e) is possible given its time estimate t^* by checking that $t^* \leq t_e$ (this is done in step 1 of the ShowCredential protocol); and
3. updates its estimate $t^* \leftarrow \max(t_s, t^*)$ if the signature is valid.

The signature by the revocation authority on the epoch makes it impossible for verifiers to trick the device into creating a too futuristic estimate t^* of the current time.

6.7.3 *How to choose the epochs*

Epochs determine during what period a credential is linkable. Ideally, at most one showing happens at each verifier within an epoch. The period between two showings wildly differs among applications. For example, a citizen credential may be used only a couple of times a year for filing tax returns with the government, while it may be used weekly to prove having reached legal drinking age in a pub or a store. A credential for accessing an online newspaper subscription could even be used daily.

At the same time, computing revocation lists for every epoch can become computationally intensive and transferring the list uses bandwidth. Therefore, we propose not to have a global epoch, but instead create epochs per verifier. The length of the epoch should be chosen in such a way that no credential is normally reused within the epoch for that particular verifier.[11] Using time to instantiate epochs (as described in Section 6.7.2) allows us to use verifier-specific epochs easily.

6.7.4 *Experiments*

We did two experiments to prove the validity of our scheme: we estimated the performance impact on our existing smart card implementa-

11 Note that when a user *does* use her credential more often within the same epoch a lot of anonymity remains. The uses within this epoch are linkable, but they are still unlinkable to uses in other epochs or at other verifiers. In particular, this will usually not reveal the user's identity.

tion and tested the impact on the revocation authority. As the extra work for the verifier is extremely small, we did not measure its overhead.

Fast smart card implementation

We estimate the efficiency of this scheme based on the work by Vullers and Alpár [162] in the IRMA project. To assess the performance of the implementation, we compare it to its version without revocation. Similarly to Section 6.6, we add an extra attribute to every credential to hold the revocation value.

As group $\mathbb{G}$, we use a subgroup of $\mathbb{Z}_q$ of prime order p. Here q is a 1024-bit prime such that $q - 1 = bp$. This choice of q is small, but matches the security level used in the implementation of Vullers and Alpár [162]. The group $\mathbb{G}$ is cyclic and the DDH problem is hard. Furthermore, hashing onto this group is rather easy. It takes five 256-bit hash calculations to get a (statistically uniformly) random element in $\mathbb{Z}_q$ and one exponentiation to the power b, the cofactor, to obtain an element in $\mathbb{G}$ [87]. The exponentiation can be precomputed as part of the revocation value. Calculating a 256-bit hash takes about 10 milliseconds. We estimate a total extra time of 390 milliseconds for including the revocation value as an attribute, generating the revocation token and adding the equality proof [139]. This is very practical. Since showing a credential takes 0.8–1.5 seconds, the overhead is limited too.

We estimate that the cost of verifying the epoch certificate (recall that this certificate is necessary to keep track of time) is approximately 150 milliseconds.

Fast revocation list calculation

The main remaining burden of the revocation scheme is on the revocation authority, which has to generate revocation lists for all verifiers, and has to do so for each epoch. This can amount to a large number of exponentiations. However, the reader should be aware that the amount of work the revocation authority has to do per generator (i.e., per epoch and per verifier) equals the work that a verifier has to do for *every verification* in the standard VLR setting.

Idemix [36, 87] uses a modular arithmetic setting for their credential scheme. One option is to reuse this setting to create a cyclic group $\mathbb{G}$, as described in the previous section. We created a (non-optimized) test application, built using the GMP big number library[12] to get an estimate for the time required to build the revocation list. Our application calculates approximately 7 500 revocation tokens per second on a single

12 The GNU multiple precision big number arithmetic library: https://gmplib.org/.

core of a first generation mobile Intel Core i7 at 2.66 GHz. This is already acceptable in a system with a small number of users and service providers, for example with 450 service providers and 10 000 revoked users all lists can be generated in just 10 minutes.

However, nothing prevents us from choosing a more efficient group. It does not matter for the proof of knowledge. The only impediment might be that the smart card may not support this group. For reference, we also created an optimized implementation using the elliptic curve cryptography (ECC) library by Bernstein et al. [14]. The authors of this library already went to great lengths to create fast exponentiation for a fixed generator. We extended this library somewhat to also support dynamic generators (and do the pre-computation on the fly). This implementation performs approximately 50 000 exponentiations per second, on a single core 2.53 GHz machine. This should be fast enough for even nationwide deployment.

There is one technicality that one has to take care of when using an ECC library such as the one by Bernstein et al. Often, points are represented internally in projective coordinates. This saves an expensive inversion operation in the underlying field. However, it also means that points do not have a unique representation. Such a unique representation is, however, essential to our fast revocation check. We normalize the representation by using Montgomery's trick [123] to calculate the inverses. By using this trick we only require 1 inversion and $3n$ multiplications to calculate n inverses. This causes a significant speedup over the naive approach of normalizing each element separately. The inversion cost is taken into account in the performance measures given above.

The specific curve we used above is generally not available on smart cards, but other curves are; see for example Hein et al. [78]. Finally, our results with the optimized ECC library suggest that also in the modular arithmetic setting improvements in speed can still be obtained.

6.7.5 *The size of a revocation list*

Our scheme requires the distribution of revocation lists. It might seem that when the revocation list contains many items, the size of these revocation lists could become prohibitive. We will show that this is not the case.

Throughout, let ν be the number of items on the revocation list. The list contains group elements, therefore their size depends on the group. We consider two types of cyclic groups, both of prime order p of about 256 bits. We follow the 2012 ECRYPT advisory [61] in selecting sizes that give long term protection:[13]

13 In the 2016 ECRYPT-CSA advisory [62] the key-sizes we mention here are considered secure

- A cyclic subgroup of the integers modulo a prime q. For a group order of 256 bits, q itself needs to have 3248 bits. A group element is thus 406 bytes.
- An elliptic curve group of order p, such that p is about 256 bits. Only the x coordinate and one bit for the y coordinate need to be stored. Thus a group element takes about 32 bytes to represent.

Table 6.1 compares the storage requirements for a single revocation list, for $\nu = 2^{15}$, $\nu = 2^{18}$ and $\nu = 2^{21}$ elements. We see that especially for the integers modulo q the storage requirements are considerable.

For traditional $O(1)$ access structures, the verifier needs to store at least the entire list itself, and additionally some overhead. Since we only test membership of the revocation list, and do perform calculations based on the elements on the revocation list, it suffices to store the hashes of the elements. This reduces the storage requirements immediately, see Table 6.1. However, if we accept a very small error probability, we can do even better by applying Bloom filters.

Bloom filters

Bloom filters are a probabilistic data structure that can very efficiently store revocation tokens, at a constant number of bits per item, independent of the size of the item itself [17]. The number of bits per item is so small that a Bloom filter gives a one or two orders of magnitude improvement over storing the elements directly.

This increased efficiency comes at a price: the filter can give false positives, i.e., it can claim that an element is on the revocation list, while in fact it is not. However, the false positive rate can be made small. We think that in this setting a small (in the order of 10^{-6}) false positive rate is acceptable for two reasons. One, in a practical system the error probability due to other means (such as intermittent connections and user error) is probably much higher, and two, when a sufficient number of generators are available, the user can easily retry with a fresh one (the probability that both fail is extremely small). In fact, we might even accept a higher false positive rate if the verifier can always fall back to an online check with the RA (that then does need to store the full list).

A Bloom filter is constructed as follows. It consists of a bit array of length κ together with λ hash functions H_i that map strings into indices of this array, i.e., they map into the range $\{1, \ldots, \kappa\}$. To store an item m in the filter, calculate $H_1(m), \ldots, H_\lambda(m)$, and set those bits in the array to one. To check if an item m is on the list, calculate $H_1(m), \ldots, H_\lambda(m)$. If all these indices are set, the item is most likely in the filter.

for at least the next 10 years (until 2026), as opposed to the 2012 version that considered them secure until 2040.

Table 6.1: Comparison of storage requirements for a single revocation list. We consider different sizes of the revocation list, and two types of groups: the integers modulo q, with elements of 406 bytes and an elliptic curve, with elements of 32 bytes. Verifiers in our scheme only do membership tests with the revocation list, so instead of storing the elements themselves it suffices to hash them (with SHA256 in this case), or to store them in a Bloom filter. The data are parameterized by the false positive probability P of the Bloom filter (based on $\lambda = \lfloor \ln(2)\kappa/\nu \rfloor$ hash functions), the length κ of the filter and ν the number of revoked items.

	Nr. of revoked items (ν)		
	2^{15}	2^{18}	2^{21}
Integers modulo q	13 MiB	102 MiB	812 MiB
Elliptic curve	1 MiB	8 MiB	64 MiB
Hashes of elements	1 MiB	8 MiB	64 MiB
Bloom filter			
$P = 4.6 \cdot 10^{-4}, \kappa/\nu = 16$	64 KiB	512 KiB	4 MiB
$P = 9.9 \cdot 10^{-6}, \kappa/\nu = 24$	96 KiB	768 KiB	6 MiB
$P = 2.1 \cdot 10^{-7}, \kappa/\nu = 32$	128 KiB	1 MiB	8 MiB

It can be shown that the probability P of a false positive for a Bloom filter storing ν items is given by

$$P \approx \left(1 - e^{\frac{-\lambda\nu}{\kappa}}\right)^{\lambda}.$$

This probability is minimal for $\lambda = \ln(2)\kappa/\nu$. Table 6.1 shows that a Bloom filter uses at least an order of magnitude less storage than a traditional solution at acceptable false positive rates.

The number of hash function calls is small too. The biggest filter with $\kappa/\nu = 32$ and $\nu = 2^{21}$ contains 2^{26} items. We need $\lambda = \lfloor \ln(2)32 \rfloor = 22$ hash functions with a 26-bit output. If we make one SHA256 call, we get 256 bits, therefore we need to make 3 SHA256 calls (with appropriate padding to get different hash functions) for every item.

6.8 RELATED WORK

Revocation has been widely studied in the literature; we refer to, for example, Lapon et al. [100] for a nice overview of current revocation techniques for attribute-based (Idemix) credentials. Traditional revocation techniques, such as CRLs and OCSPs, require credentials to have a unique identifier that is always visible to the verifier. A certificate revocation list (CRL) [50] is a list of revoked credential identifiers, published by the issuer. Alternatively, the verifier can ask the issuer if a credential is still

Table 6.2: We compare CRLs [50], accumulators (ACC) [33, 37, 128], traditional VLR schemes [9, 24, 28], VLR schemes with backward unlinkability (VLR-BU) [124], blacklistable anonymous credentials (BLAC) [159], and our scheme (OURS). We compare the complexity of the operations and data transfers. A proving time of 1 means that it is constant, while a proving time of $|RL|$ means that it scales linearly with the size of the revocation list. Of all the constant-time proving schemes, the accumulator has the biggest overhead. Our scheme is the only privacy-friendly scheme that has constant-time proving and verification while users do not need to receive updates.

	CRL	ACC	VLR	VLR-BU	BLAC	OURS
User can be offline	✓	×	✓	✓	✓	✓
Data to verifier						
per epoch	$\|RL\|$	1	$\|RL\|$	$\|RL\|$	$\|RL\|$	$\|RL\|$
per update	1	1	1	1	1	1
Proving (time)	1	1	1	1	$\|RL\|$	1
Verifying (time)	1	1	$\|RL\|$	$\|RL\|$	$\|RL\|$	1
Privacy	-	+	+/-	+	+	+

valid using the online certificate status protocol (OCSP) [141]. Both situations require the credential to be recognizable, which is undesirable for ABCs. However, revocation is fast: there is no extra work required on the side of the user, and the verifier can test validity in constant time.

Domain-specific pseudonyms [26, 87, 99] only slightly improve the situation: instead of being globally linkable, different uses are only linkable by the same verifier, but not across different verifiers. We believe this still weakens the unlinkability too much.

A final trick would be to use verifiable encryptions, see for example Camenisch and Shoup [38], to encode the revocation information at the cost of less efficiency. A trusted third party can then decrypt the ciphertext and check whether the credential has been revoked. Clearly this party is privy to too much information, and such a solution should thus be avoided.

We now focus our attention on solutions that do offer sufficient privacy guarantees for the user. Table 6.2 compares these schemes with our scheme and the CRL scheme. A digital accumulator is a constant-sized representation of a set of values. Every value in the accumulator comes with a witness, which enables efficient membership checks. Camenisch and Lysyanskaya [37] proposed an updatable accumulator that can be used for revocation. A credential is unrevoked as long as

it appears on the whitelist, represented by the accumulator. Another approach is to accumulate revoked credentials to create a blacklist. A credential is unrevoked if it is not on this blacklist [104, 128].

Accumulators change. For whitelists, this is after an addition; for blacklists, this is after a revocation. Thus users need to receive updates (for schemes such as Camenisch et al. [33], these updates are public and can be provided by the verifier) and process them, inducing extra load on carriers such as smart cards. Additionally, the (non-)membership proofs are expensive. Lapon et al. [100] show an overhead of 300% in the showing protocol. Other schemes, such as Libert et al. [105] are equally inefficient, making them impractical.

Where accumulators place the load on users—who need to get new witnesses after revocations or additions—and the revocation authority—who needs to create those witnesses—verifier-local revocation (VLR) [9, 24, 28] places the majority of the load at the verifier. As we saw in Section 6.1, the verifier needs to do a check that is linear in the length of the revocation list, however, apart from sending the extra revocation token, the extra work for the user is minimal.

A downside of traditional VLR schemes is that once a user is revoked, all of its transactions (also past ones) become linkable. Nakanishi and Funabiki [124] proposed a VLR scheme that is backward unlinkable, such as our scheme. Similar to our scheme, they create different revocation tokens per epoch, so that verifiers cannot use the revocation token for the current epoch and apply it to earlier ones. However, their scheme is still linear in the number of revoked users, and needs to perform a pairing operation per revoked user. This makes it less efficient than previous solutions as well as our solution. The security of their scheme hinges on the fact that the per-epoch revocation tokens are maintained by a trusted party. It thus requires the same trusted party as our scheme.

Direct anonymous attestation (DAA) [27, 28] uses the same technique as Boneh and Shacham's VLR scheme to revoke tags—that is, the user (or *tag* in the DAA terminology) creates a tuple (h, h^{sk_U}) for a random generator h where sk_U is the user's private key. A rogue user's tuples can then be recognized because her private key sk_U is known, see Section 6.1. In DAA this same mechanism is also used to derive domain-specific pseudonyms. To do so, the user creates the generator h as the hash of the verifier's basename. We generalize this by creating a generator not only based on the verifier's identity, but also on the current epoch. Choosing generators in this fashion drastically limits the linkability that a user would otherwise incur, also see above.

Blacklistable anonymous credentials (BLAC) [159] take a different approach to revocation: misbehaving users can be blacklisted without requiring a trusted third party (TTP) to provide a revocation token. In every

transaction, the user provides a ticket, similar to our revocation token, that is bound to the user. To blacklist a user, the verifier places this ticket on the blacklist. In the second step of the authentication, the user proves that her ticket is not on the blacklist. The complexity of this proof is linear in the number of items on the blacklist, so this scheme places a high load on the user. Even if a user's credential is revoked, the verifier does not learn her identity, nor can the verifier trace her.

Finally, the FAUST [107] and unlinkable serial transactions [157] systems suggest another method to prevent users from becoming linkable when a user uses her credential more than once per epoch at the same verifier. To do so, after the first showing of her unrevoked credential at a verifier in an epoch, the verifier blindly issues a token to the user. The next time the user contacts this verifier within the same epoch, the user uses this token to prove that she was not revoked in this epoch. (If tokens are one-time use, the verifier issues a new token after every use of a credential.) This approach only leaks that the user authenticated before. However, because of its construction, credentials cannot be revoked during the epoch once they have been used once.

6.9 DISCUSSION AND CONCLUSION

Our revocation scheme is fast. It can be combined with ABC showing protocols and can be *fully* implemented on a smart card. It incurs minimal overhead, while at the same time the revocation check can be performed efficiently by the verifier. We created a security model for our scheme and proved that our scheme is forward secure as long as the revocation authority is trusted. We showed that we can remove this trust assumption when the users calculate the revocation tokens themselves. Finally, we showed that by using multiple generators we can even limit the linkability within an epoch.

To obtain this speedup, we traded some traceability, but with an appropriate choice of epoch length this should not be a problem in practice. The fact that this enables us to create a revocation system that is truly practical makes this a worthwhile trade-off.

We believe our scheme is a valuable contribution to making large scale attribute-based credentials possible. It would be interesting to investigate protocols that further reduce the trust assumption on the revocation authority.

7

SUBLINEAR SCALING FOR PRIVATE INFORMATION RETRIEVAL

Private information retrieval (PIR) was introduced in the seminal work of Chor et al. in 1995 [47]. In PIR, a client wishes to retrieve information from online database servers while revealing to the database operators *no information* about what data she seeks. That this is even possible is counterintuitive, but consider the *trivial download* scheme: the database server sends the entirety of the database to the client, who searches it herself. This is clearly private, but comes at a high communication cost for large databases. Non-trivial PIR schemes aim to achieve the same level of privacy while transmitting far less data. The simplest PIR schemes assume that the database consists of an array of equal-sized blocks, and that the client knows the index of the block she wishes to retrieve. However, previous work showed that this simple query mechanism can be used as a black box to realize more expressive database search functionality, including search by keywords [46] and private SQL queries [132].

Chor et al.'s original 1995 work showed that one cannot have both information-theoretic privacy (i.e., privacy even when the server is computationally unbounded) and a sublinear (in the size of the database) communication cost if only one server is used. Information-theoretic PIR (IT-PIR) schemes, however, circumvent this impossibility result by using multiple database servers and a *noncollusion* assumption—that at most a bounded number of servers (less than the total number) will collude against the client. They achieve a communication cost much smaller than the size of the database, and modern ones additionally achieve *robustness*—even if some of the servers are unresponsive, buggy, or actively malicious, the client can nonetheless retrieve her information (and identify the misbehaving servers) [12, 57, 72].

If information-theoretic privacy is not required, computational PIR (CPIR) schemes can be used. These schemes rely on computational or cryptographic assumptions to guarantee privacy against a single database server at low communication cost [98]. Devet and Goldberg [56] also recently proposed a hybrid PIR scheme that combines a CPIR scheme with an IT-PIR scheme to achieve some of the desirable properties of both (in particular, fast computation and lower communication costs), while hedging against violations of either the computational or noncollusion assumptions.

While much effort has gone into reducing the communication costs of PIR protocols, it is also important to consider the computational cost. A PIR server typically must process the entirety (or at least a significant fraction) of the database when handling each query, lest it learn information about what the client is likely *not* seeking.

Not all PIR schemes can beat the trivial download approach. Sion and Carbunar [149] found that it would always be faster to simply download the entire database than to use Kushilevitz and Ostrovsky's CPIR scheme [98]. Later, Olumofin and Goldberg [133] noted that a more modern CPIR scheme by Aguilar-Melchor and Gaborit [5], as well as a number of IT-PIR schemes, are orders of magnitude faster than the trivial download scheme. However, the computation costs are still nontrivial, requiring on the order of 1 s of CPU time[1] per gigabyte of database size, for each IT-PIR query. Recent improvements by Aguilar-Melchor et al. [4] show that CPIR using lattice-based schemes is, in fact, practical. The performance of their XPIR scheme comes within one order of magnitude of modern IT-PIR schemes.

In order to reduce the per-query CPU cost, a number of authors have proposed *batch* techniques, in which a PIR server performs a computation over the database and a batch of simultaneous queries, resulting in less work than computing over the database once for each query separately. Henry et al. [81] propose a batching method based on *ramp schemes* particular to Goldberg's IT-PIR scheme [72], while Ishai et al. [88] use *batch codes* (discussed in more detail below) to provide multi-query computational speedups for any PIR scheme.

Both of these proposals, however, require that the *clients* construct their queries in a special way to achieve the batching speedups. This means that these approaches help only in those scenarios where single clients (or closely cooperating groups of clients) are fetching large batches of queries at the same time.

OUR CONTRIBUTIONS. In this work, we address the more general case in which a PIR server wishes to process a batch of queries simultaneously, whether they were received all from the same client, each query from a unique client, or anything in between. We approach this problem by first observing a mathematical relationship between Ishai et al.'s method of applying batch codes to speed up IT-PIR and a special case of matrix multiplication where the left matrix has a specific structure, see Section 7.2. We then generalize this observation to the case of general matrix multiplication. In doing so, we remove all restrictions on the structure of the queries to be batched. We accept a more modest batch-

1 However, this CPU time is almost completely parallelizable if multiple cores or servers are available.

ing speedup to remove the single (or coordinated) client restriction and to remove the potentially large amount of communication induced by Ishai et al.'s method.

We apply our new technique to the setting of Certificate Transparency, see Section 7.3, in which web clients fetch information about TLS certificates from log servers, but should hide from the log servers which certificate's information it fetches. This appears to be a perfect opportunity to employ PIR, but the large number of *non-cooperating* clients expected to use the system makes multi-client batching imperative. We note that while batching queries reduces the total computation time at the cost of increasing the latency for individual queries, this extra latency is not an issue in this particular application. We implemented our new technique on top of the open-source Percy++ PIR library [73]. In Section 7.4 we measure its performance.

While our practical improvements—a little more than a 4-fold speedup—are modest, we do offer *sublinear* scaling in the number of queries for *independent* clients, something simpler improvements cannot offer. Additionally, any other system-level optimizations can easily be used on top of our algorithmic ones.

7.1 BACKGROUND

Our construction combines Goldberg's robust IT-PIR scheme [72] with fast matrix multiplication techniques inspired by batch codes. Therefore, we first review and compare these notions.

7.1.1 *Goldberg's robust IT-PIR scheme*

Goldberg models the database as an $r \times s$ matrix $\mathbf{D}$ over a finite field $\mathbb{F}$. Every row in $\mathbf{D}$ corresponds to a single *block* in the database (recall that users are interested in retrieving such a block); every block consists of s field elements. To request block i (non-privately) the client could simply send i to the server. However, as a first step, we express the PIR operation as a vector-matrix multiplication before producing a true privacy-friendly scheme. The client constructs the ith standard basis vector e_i of $\mathbb{F}^r$ (i.e., the vector of length r with all zeros except for a 1 in the ith position) and sends it to the server. The requested block i is then obtained by calculating the vector-matrix product $e_i \cdot \mathbf{D}$.

To make the query privacy friendly, the client in Goldberg's scheme creates a k-out-of-n Shamir secret sharing [146] of this standard basis vector e_i. It sends one share to each of the n database servers, which compute the vector-matrix product with the database and return the result. Lagrange interpolation of the shared vectors gives the standard ba-

sis vector; since matrix multiplication and Lagrange interpolation are linear, interpolation of the results yields the ith block of the database. The secret sharing scheme guarantees that as long as at most $k-1$ servers collude, they learn nothing about the target block.

Goldberg's scheme is robust [57, 72]. It permits some of the servers to misbehave, while still enabling the client to recover her record and identify the misbehavers.

COMMUNICATION COST. To read a single block, the client sends r field elements to, and receives s field elements from, each server. For a fixed database size of v field elements, it is best to select $r = s = \sqrt{v}$. Henry et al. [81] show how to build on this simple fixed-block-size PIR primitive to handle more realistic databases with variable-sized records.

SERVING MULTIPLE SIMULTANEOUS QUERIES. Suppose a server receives multiple queries $v_1, \ldots, v_q$ simultaneously. It could answer them by computing the q vector-matrix products $v_i \cdot \mathbf{D}$ individually. However, it can also first group the queries into one matrix $\mathbf{Q}$ where row i consists of query v_i. Then the server computes the matrix-matrix product $\mathbf{Q} \cdot \mathbf{D}$. Row i of the result is the response to the ith query.

With a naive matrix multiplication algorithm the work the server needs to do is the same in both cases: about $2qrs$ operations (qrs multiplications, and about the same number of additions). However, as we will see, using better matrix-multiplication techniques will significantly improve the situation.

RAMP SCHEME. Henry et al. [81] replace the Shamir secret sharing in Goldberg's PIR scheme with a *ramp scheme*. In this way, a single client can encode more information in each server request, and can retrieve q blocks instead of just 1 *without* increasing the per-server computation or communication cost at all. The large drawback to this scheme (in addition to being useful only for single clients making multiple queries, and not for multiple clients making single queries) is that it must trade some of the robustness of Goldberg's scheme for extra parallel queries, or conversely, that it requires $q-1$ extra servers in order to maintain the same level of robustness.

7.1.2 *Batch codes*

Batch codes can be used to answer multiple queries efficiently. The idea, proposed by Ishai et al. [88], is to encode the database in a special way, so that a single client can efficiently make multiple queries. This idea is best illustrated using an example. As in the rest of this chapter, we

apply the batch codes to Goldberg's IT-PIR scheme.

Suppose we want to prepare a database with r rows for two simultaneous queries. We create three separate databases: $\mathbf{D}_1$, containing the first $r/2$ rows; $\mathbf{D}_2$, containing the last $r/2$ rows; and $\mathbf{D}_3 = \mathbf{D}_1 \oplus \mathbf{D}_2$. Any two queries, say for blocks i_1 and i_2, can be answered by making at most one PIR query to each of the $\mathbf{D}_i$: if blocks i_1 and i_2 are not in the same half of the database, the queries can be answered by making one PIR query to $\mathbf{D}_1$ and one to $\mathbf{D}_2$. Suppose, on the other hand, that i_1 and i_2 are both in the first half. Then block i_1 can be retrieved directly, while block i_2 is obtained by making one query to $\mathbf{D}_2$ and one to $\mathbf{D}_3$. Taking the XOR of the latter two results yields the desired row in the first half. Two queries for the second half are handled similarly.

This procedure reduces the computational cost for the server. As we saw in the previous section, a naive method requires $4rs$ field operations; in contrast, the batch code solution requires only $3rs$ field operations. (Again, half multiplications and half additions.)

Note that to hide which indices the client is querying she needs to make a query to each of the three parts, even if two would suffice to get the answer. This means that the client sends $\frac{3}{2}r$ elements to, and receives $3s$ elements from, each server. In the naive case she sends $2r$ elements and receives only $2s$ elements.

In general, an (r, N, q, m) batch code will take a database of r blocks and create m subdatabases, such that the total number of blocks in the subdatabases is N. The code can be used to answer q queries by making one request to each of the m subdatabases. The example we sketched before gives an $(r, \frac{3}{2}r, 2, 3)$ batch code.

Suppose we use an (r, N, q, m) batch code to speed up PIR queries to a database with r blocks, each consisting of s field elements. Let $N_1, \ldots, N_m$ be the number of blocks in the m subdatabases (so that $\sum_i N_i = N$). To make q queries, a client needs to make one PIR query to each of the m subdatabases with respectively $N_1, \ldots, N_m$ blocks. The query to subdatabase i costs $2N_i s$ field operations. Therefore the total computational load on the server is $2Ns$. The client sends N group elements, and receives ms elements.

Subcube batch code

Ishai et al. [88] generalize the sketch above as follows. First, instead of splitting the database into 2 parts, it can be split into ℓ parts. A final $\ell + 1$th part is added, being the XOR of all the previous parts. Again, any two items can be obtained using $\ell + 1$ queries, one to each of the subdatabases—if the two items happen to be in the same part it is necessary to retrieve and calculate the XOR of all the other items. This gives rise to an $(r, \frac{\ell+1}{\ell}r, 2, \ell + 1)$ batch code. Obviously, this is good

Table 7.1: Summary of batch codes with parameters [88]. The subcube code is parametrized by t and $\ell \geq 2$, while the subset code is parametrized by ℓ, r' and $0 < \alpha < \frac{1}{2}$, where $w = \alpha\ell$. The parameters r and r' scale the codes to support more blocks, without essentially changing their structure.

	Subcube	Subset
Number of blocks (r)	r	$r'\binom{\ell}{w}$
Sum of subdatabase sizes (N)	$\left(\frac{\ell+1}{\ell}\right)^t r$	$r' \sum_{j=0}^{w} \binom{\ell}{j}$
Number of queries (q)	2^t	$\geq 2^w$
Number of subdatabases (m)	$(\ell+1)^t$	$\sum_{j=0}^{w} \binom{\ell}{j}$

for computation, as the server needs to do only $2\frac{\ell+1}{\ell}rs$ field operations. While the sending cost drops to $\frac{\ell+1}{\ell}r$ elements, the receiving cost rises to $(\ell+1)s$ elements. (Note that the client always needs to retrieve the $\ell+1$ records to protect her privacy.)

For simplicity, let us return to the case where $\ell = 2$. The scheme can be applied recursively to answer more queries. Suppose the client makes $q = 4$ queries. Group these into two pairs. Each pair can be answered by making only one query to each of the three parts $\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3$. In total, two queries are made to each $\mathbf{D}_i$, so we can apply the above scheme again, but now on the smaller databases.

Recursively applying this scheme gives a system that can handle $q = 2^t$ queries. Table 7.1 summarizes the important parameters of this scheme. By taking ℓ large, this scheme gets arbitrarily close to the optimal processing time for the server: it can answer 2^t queries with only slightly more processing than is required for a single query. However, the price is a higher communication cost for the client.

The subset batch code

Ishai et al. also describe another batch code that has more favourable properties: the subset batch code. It is, however, also more complex. We only summarize the results in Table 7.1, and refer to their paper [88] for a full description of this scheme. The scheme is parametrized by ℓ, r', and $0 < \alpha < \frac{1}{2}$. The value w is then given by $\alpha\ell$.

It can be shown that, for this code, doing q queries is approximately $(1-\alpha)/(1-2\alpha)$ times more expensive than doing a single query. Thus, picking a small α brings the computational overhead for the server arbitrarily close to optimal. Contrary to the subcube codes the communication overhead is also polynomial in q, however, in practice the overhead turns out to be rather high, especially when α is small.

Table 7.2: Comparison of multi-query PIR schemes: the naive scheme, the ramp scheme [81], subcube batch codes (SBC) [88], and our work. We show counts of per-server field operations, as well as the number of field elements sent to and received from each server, the number of extra servers the scheme requires to maintain the same robustness level as for a single query, and an indication of whether independent clients can use the method, or whether all queries must be sent by a single client (or coordinated clients). The database consists of r blocks, each containing s field elements. The number of simultaneous queries, q, is assumed to be a power of 2, and much smaller than either r or s. Note that our work achieves sublinear scaling of computation in the number of queries q, while also admitting independent clients.

	Naive	Ramp	SBC	Our work
$\mathbb{F}$ mult.	qrs	rs	$q^{\lg((\ell+1)/\ell)}rs$	$q^{0.80735}rs$
$\mathbb{F}$ additions	$q(r-1)s$	$(r-1)s$	$\frac{(\ell^2-1)}{\ell}q^{\lg((\ell+1)/\ell)}rs$	$\frac{8}{3}q^{0.80735}rs$
Send	qr	r	$q^{\lg((\ell+1)/\ell)}r$	qr
Receive	qs	s	$q^{\lg(\ell+1)}s$	qs
Extra servers	0	$q-1$	0	0
Indep. clients	✓	×	×	✓

Consider the following example. We want α to be somewhat small, so we take $\ell = 20$ so that with $\alpha = 0.2$ we get $w = 4$. Suppose we make $q = 16$ queries, Then, $N/r = 1.279$ so the computation cost for 16 queries is only 27.9% more than for 1 query. However, we need to receive $m/q = 387$ times more data than the naive approach for $q = 16$ queries. So, using this code at low computational cost can incur extremely high communication costs.

Challenges

The two main drawbacks of using batch codes for PIR are: (1) the requirement that all of the queries be generated by a single client (or by closely cooperating clients); and (2) the increased communication cost, which becomes especially prohibitive for large databases.

We will address both of these issues in this chapter. See Table 7.2 for a comparison of multi-query PIR schemes. Although we only list the subcube batch code and not the subset batch code in the table for conciseness, the two salient challenges listed above are the same for both types.

7.1.3 *Matrix multiplication algorithms*

Naive matrix multiplication of a matrix $\mathbf{Q}$ of size $q \times r$ with a database $\mathbf{D}$ of size $r \times s$ requires qrs multiplications and at least $q(r-1)s$ additions (although most implementations will actually use qrs additions). For two square matrices of size $\nu \times \nu$ the complexity is $O(\nu^3)$.

Faster matrix multiplication algorithms exist that have an asymptotic complexity with a better exponent. In this chapter we focus on Strassen's algorithm [156] because of its relative simplicity. This algorithm achieves a time complexity of $O(\nu^{\lg 7}) = O(\nu^{2.8074})$. Faster algorithms exist, such as that of Coppersmith and Winograd [51], which achieves an even better bound of $O(\nu^{2.3729})$. However, this comes at the cost of a much larger multiplicative constant.

Strassen's algorithm is extremely simple. It splits each matrix into four, equal-sized submatrices. A naive block-matrix multiplication of these would require 8 multiplications of the smaller sized matrices. However, using Strassen's algorithm, only 7 are needed. This technique is then applied recursively to the multiplications of the smaller matrices.

Strassen's algorithm in detail

Strassen's algorithm is best explained by looking at matrix multiplication from a block-matrix perspective. For simplicity, assume that all matrices have size $\nu \times \nu$ where ν is even. If

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} \\ \mathbf{Q}_{21} & \mathbf{Q}_{22} \end{pmatrix} \quad \text{and} \quad \mathbf{D} = \begin{pmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} \\ \mathbf{D}_{21} & \mathbf{D}_{22} \end{pmatrix},$$

then the matrix product $\mathbf{R} = \mathbf{Q} \cdot \mathbf{D}$ is given by

$$\mathbf{R} = \begin{pmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{R}_{21} & \mathbf{R}_{22} \end{pmatrix},$$

where

$$\begin{aligned} \mathbf{R}_{11} &= \mathbf{Q}_{11} \cdot \mathbf{D}_{11} + \mathbf{Q}_{12} \cdot \mathbf{D}_{21} \\ \mathbf{R}_{12} &= \mathbf{Q}_{11} \cdot \mathbf{D}_{12} + \mathbf{Q}_{12} \cdot \mathbf{D}_{22} \\ \mathbf{R}_{21} &= \mathbf{Q}_{21} \cdot \mathbf{D}_{11} + \mathbf{Q}_{22} \cdot \mathbf{D}_{21} \\ \mathbf{R}_{22} &= \mathbf{Q}_{21} \cdot \mathbf{D}_{12} + \mathbf{Q}_{22} \cdot \mathbf{D}_{22}. \end{aligned}$$

It thus reduces to 8 matrix multiplications of size $\nu/2$. In Strassen's algorithm the following 7 matrix products are calculated first (note that

in fields of characteristic 2, the $+$ and $-$ operations are the same):

$$\begin{aligned}
\mathbf{M}_1 &= (\mathbf{Q}_{11} + \mathbf{Q}_{22}) \cdot (\mathbf{D}_{11} + \mathbf{D}_{22}) \\
\mathbf{M}_2 &= (\mathbf{Q}_{21} + \mathbf{Q}_{22}) \cdot \mathbf{D}_{11} \\
\mathbf{M}_3 &= \mathbf{Q}_{11} \cdot (\mathbf{D}_{12} - \mathbf{D}_{22}) \\
\mathbf{M}_4 &= \mathbf{Q}_{22} \cdot (\mathbf{D}_{21} - \mathbf{D}_{11}) \\
\mathbf{M}_5 &= (\mathbf{Q}_{11} + \mathbf{Q}_{12}) \cdot \mathbf{D}_{22} \\
\mathbf{M}_6 &= (\mathbf{Q}_{21} - \mathbf{Q}_{11}) \cdot (\mathbf{D}_{11} + \mathbf{D}_{12}) \\
\mathbf{M}_7 &= (\mathbf{Q}_{12} - \mathbf{Q}_{22}) \cdot (\mathbf{D}_{21} + \mathbf{D}_{22}).
\end{aligned}$$

The matrix product is then given by:

$$\begin{aligned}
\mathbf{R}_{11} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\
\mathbf{R}_{12} &= \mathbf{M}_3 + \mathbf{M}_5 \\
\mathbf{R}_{21} &= \mathbf{M}_2 + \mathbf{M}_4 \\
\mathbf{R}_{22} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6.
\end{aligned}$$

Using this algorithm, only 7 matrix multiplications of size $\nu/2$ are necessary. Applying this trick recursively gives a complexity of $O(\nu^{\lg 7})$.

7.2 BATCH CODES AS MATRIX MULTIPLICATION

We have seen that answering multiple PIR queries in Goldberg's protocol requires calculating the matrix-matrix product $\mathbf{Q} \cdot \mathbf{D}$, as the rows of the resulting product are exactly the responses to the given queries. At the same time, batch codes speed up this computation. Hence, batch codes are in some way implementing fast matrix multiplication. In this section we identify this relation, explain the limitations of batch codes in this application, and demonstrate the similarities with Strassen's algorithm. For simplicity of exposition (and because this is the typical case in practice), we assume a field $\mathbb{F}$ of characteristic 2, so that additions are just XORS.

7.2.1 *An example*

In Section 7.1.2 we showed how a batch code can be used to reduce two queries for the full database to three half-sized queries. In terms of matrix multiplication, the client constructs its three half-sized queries q_1, q_2, q_3 and sends them to the server. The server expresses the

database $\mathbf{D}$ as a concatenation of two parts, $\mathbf{D} = \begin{pmatrix} \mathbf{D}_1 \\ \mathbf{D}_2 \end{pmatrix}$, and constructs the matrices

$$\overline{\mathbf{Q}} = \begin{pmatrix} q_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & q_2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & q_3 \end{pmatrix}_{3 \times \frac{3}{2}r} \quad \text{and} \quad \mathbf{M} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \\ \mathbf{I} & \mathbf{I} \end{pmatrix}_{\frac{3}{2}r \times r},$$

so that

$$\mathbf{M} \cdot \mathbf{D} = \begin{pmatrix} \mathbf{D}_1 \\ \mathbf{D}_2 \\ \mathbf{D}_1 \oplus \mathbf{D}_2 \end{pmatrix}_{\frac{3}{2}r \times s},$$

where $\overline{\mathbf{Q}}$ is the block-diagonal matrix of the queries, $\mathbf{I}$ is the identity matrix, and $\mathbf{M}$ is the matrix form of the batch code (representing the linear combinations of the parts that make up the resulting subdatabases). (Note that we annotate the matrices with their dimensions.) The server now computes the linear combinations of the parts, $\mathbf{M} \cdot \mathbf{D}$, and multiplies queries $\overline{\mathbf{Q}}$ by them. This results in the familiar response structure

$$\begin{aligned} \overline{\mathbf{Q}}_{3 \times \frac{3}{2}r} \cdot \mathbf{M}_{\frac{3}{2}r \times r} \cdot \mathbf{D}_{r \times s} &= \overline{\mathbf{Q}}_{3 \times \frac{3}{2}r} \cdot \begin{pmatrix} \mathbf{D}_1 \\ \mathbf{D}_2 \\ \mathbf{D}_1 \oplus \mathbf{D}_2 \end{pmatrix}_{\frac{3}{2}r \times s} \\ &= \begin{pmatrix} q_1 \cdot \mathbf{D}_1 \\ q_2 \cdot \mathbf{D}_2 \\ q_3 \cdot (\mathbf{D}_1 \oplus \mathbf{D}_2) \end{pmatrix}_{3 \times s}. \end{aligned}$$

After receiving the results, the client combines the three rows as appropriate to recover the answers to her two original queries.

7.2.2 *General batch codes as matrix multiplication*

When using general batch codes to speed up PIR we see a similar structure. Recall that an (r, N, q, m) batch code can answer q queries to a database of r rows by splitting the computation across m subdatabases $\mathbf{K}_1, \ldots, \mathbf{K}_m$ containing a total of N rows. In the preceding example—an $(r, \frac{3}{2}r, 2, 3)$ batch code—these subdatabases were $\mathbf{K}_1 = \mathbf{D}_1$, $\mathbf{K}_2 = \mathbf{D}_2$ and $\mathbf{K}_3 = \mathbf{D}_1 \oplus \mathbf{D}_2$. For general batch codes these subdatabases will be more complicated linear combinations of the parts $\mathbf{D}_i$. The $N \times r$ matrix $\mathbf{M}$ represents these linear combinations.

Again, the client first uses the batch code to convert her q queries into m subqueries $q_1, \ldots, q_m$, where each q_i is a row vector of length equal to the number of rows in $\mathbf{K}_i$. She sends these to the server. The server constructs the linear combinations of the parts, $\mathbf{M} \cdot \mathbf{D}$, and applies the

queries to them

$$\overline{\mathbf{Q}}_{m\times N}\cdot(\mathbf{M}_{N\times r}\cdot\mathbf{D}_{r\times s})=\begin{pmatrix} q_1 & & \mathbf{0}\\ & \ddots & \\ \mathbf{0} & & q_m\end{pmatrix}_{m\times N}\cdot(\mathbf{M}_{N\times r}\cdot\mathbf{D}_{r\times s})\,.$$

The server can quickly compute this product using the block-diagonal structure of $\overline{\mathbf{Q}}$. The result is an m-row response. The client can combine those m rows to produce her desired q blocks.

Note that it is the special structure of $\mathbf{Q}=\overline{\mathbf{Q}}\cdot\mathbf{M}$ that enables the server to speed up the matrix multiplication $\mathbf{Q}\cdot\mathbf{D}$. In the PIR setting, this necessitates that $\mathbf{Q}$ be produced by a single client, or by cooperating clients.

7.2.3 *Comparison with Strassen's algorithm*

Strassen's algorithm is similar to the matrix multiplication form of the batch codes above. In particular it also

1. partitions the database into parts $\mathbf{D}_i$,
2. forms linear combinations of the parts $\mathbf{D}_i$ to construct the subdatabases $\mathbf{K}_i$,
3. multiplies parts of the queries with the subdatabases, and
4. computes linear combinations of the products to produce the final result.

However, there are also differences. First, batch codes require the queries to be preprocessed by the client, or alternatively that the query matrix $\mathbf{Q}$ is given by $\overline{\mathbf{Q}}\cdot\mathbf{M}$ as above. Strassen's algorithm, on the other hand, works with any matrix $\mathbf{Q}$. Second, batch codes are essentially one-dimensional; as a result, steps 1, 2 and 3 above for batch codes operate only on complete rows, while Strassen's algorithm *subdivides* and takes *linear combinations* of rows in addition to taking *subsets* of rows (in both $\mathbf{Q}$ and $\mathbf{D}$).

While Strassen's algorithm has a higher server computational cost than batch codes, the fact that Strassen's algorithm can deal with *any* matrix $\mathbf{Q}$ is of tremendous benefit. In our PIR setting, this means that clients do not need to coordinate their queries. Indeed, they do not need to be aware that the server is implementing this optimization at all.

7.3 APPLICATION: CERTIFICATE TRANSPARENCY

We now examine an application where multi-client PIR is particularly useful: Certificate Transparency.[2] Websites use digital certificates to tie possession of a particular private key to their domain name. These certificates are signed by certificate authority (CA). To verify the validity of a website the user's browser checks that a CA it trusts signed the certificate (or that there is a certificate chain from a trusted CA leading to the certificate). Events in recent years, such as the hack of the Dutch CA DigiNotar [65], have shown that CAs cannot be trusted unconditionally. When a CA is compromised it can be used to issue false certificates that allow third parties to eavesdrop on the communication between a user and a website. The browser will not detect this as long as the compromised CA is still trusted. Certificate Transparency, as described in RFC 6962 [101], aims to detect wrongly issued certificates in a timely manner without introducing extra trust assumptions. It roughly works as follows.

1. Before a certificate is issued it is recorded by one or more *log servers*. Each of these log servers creates a signed certificate timestamp (SCT) for this certificate and will eventually add the certificate to an append-only data structure.

2. When presenting a certificate the website will also send along the SCTs from the log servers. The browser will verify that at least one trusted log server signed the certificate description.

3. The following consistency checks are done asynchronously by monitors, who check the logs for suspicious certificates, and auditors, who ensure the validity of the logs.

    a) An auditor, usually the browser, will check that the log server signed certificates do indeed appear in the append-only log of the log server.

    b) Auditors and monitors check that the logs are consistent; i.e., that no certificates have been changed or retroactively inserted into the log.

    c) Monitors, usually CAs and webservers, monitor the log to detect inconsistencies such as two certificates, by different CAs, for the same domain.

2 Incidentally, as Goldberg and Devet explain [56], PIR is also very useful to make regular checking of revocation status of (traditional) certificates using certificate revocation lists (CRLs) privacy friendly.

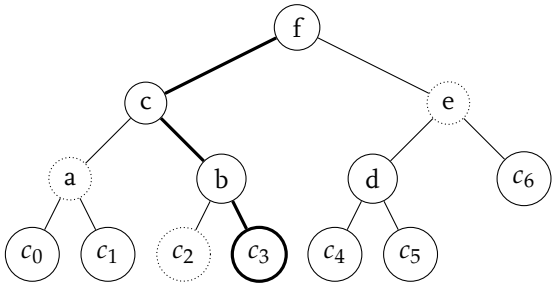


Figure 7.1: An example Merkle hash tree for 7 certificates $c_0, \ldots, c_6$ encoded into the leaves. For certificate c_3 the proof of inclusion consists of all the dotted nodes: c_2, a and e. This proof can be checked as follows. First, calculate the hash of the certificate to get c_3. Then, b is the hash of c_2 and c_3; c is the hash of a and b; and, finally, f is the hash of c and e. If the calculated root f matches the signed root the auditor is convinced that the tree contains c_3.

It is essential that the first consistency check is done, because monitors can only detect falsely issued certificates when they appear in the log. However, the first check also reveals to the log server which websites the user is visiting. We will use multi-client PIR to allow many independent clients to query a log server for the proofs of inclusion of certificates.

7.3.1 *Proving that a certificate is included in the log*

The certificates are recorded in a Merkle hash tree. A Merkle hash tree is a binary tree, in which every leaf contains the hash of a certificate, while every internal node contains the hash of its children. The root then captures information about all the children. Periodically, log servers add all the newly logged certificates to the tree and sign the new root.

The number of leaves, n, determines the structure of the Merkle hash tree. Let t be the largest power of 2 smaller than n, so that $t < n \leq 2t$. Then the left subtree of the Merkle hash tree of n nodes is the full binary tree with t leaves, while the right subtree is the Merkle hash tree of the remaining $n - t$ nodes. See Figure 7.1 for an example.

This format allows log servers to construct a proof of inclusion of a certificate for an auditor. The auditor already has the certificate, and thus also the leaf corresponding to the certificate (the leaf contains the hash of the certificate). The log server gives the auditor those node hashes needed to recalculate the root of the tree starting with the certificate. The extra nodes needed for this proof are all the siblings on the path from the leaf to the root; see Figure 7.1. Finally, the auditor compares the calculated root with the signed root from the server.

The length of the proof is no larger than the height of the tree. Therefore, the size of the proof grows only logarithmically in the number of leaves. The specification requires SHA256 as the hash function for the internal nodes, so a node contains 32 bytes of data.

7.3.2 *The number of web certificates*

To determine the feasibility of retrieving the proofs of inclusion using PIR, we need to estimate the number of active and valid TLS certificates—it does not make much sense to retrieve proofs for expired certificates. We estimate the number of TLS certificates based on the following sources.

EFF SSL OBSERVATORY The EFF SSL observatory[3] observed about 1.4 million valid certificates in 2010.

PUBLIC NETCRAFT DATA In their public sample of May 2013,[4] Netcraft claimed that Symantec at that time had produced more than one third of all certificates. In their April, 2012 press release[5] Symantec quotes 811,511 installed certificates. This gives an estimate of approximately 2.4 million certificates in 2012.

PILOT CT SERVER As of early July 2014, Google's pilot certificate transparency server had logged about 4.5 million certificates.[6] It is not clear how reliable this number is, since at that time there was no incentive to add all certificates to this list. Also, the log is append-only, so this number is probably higher than it should be.

CT OBSERVATORY At the end of 2016, the CT observatory[7] run by the University of Bonn reports to have seen about 4.1 million currently valid certificates (out of a total of 19.4 million).

Given these data points, we estimate that the number of valid TLS certificates is currently around 2^{22} or approximately 4 million.

7.3.3 *Retrieving proofs of inclusion using PIR*

To make privacy-friendly retrieval of the proofs of inclusion possible, we store the proofs as records in a database and use PIR to retrieve them.

3 https://www.eff.org/observatory
4 https://www.netcraft.com/internet-data-mining/ssl-survey/, last accessed July 2014. These data have since been replaced by sample data from 2015 (checked on February 17, 2017).
5 https://www.symantec.com/about/newsroom/press-releases/2012/symantec_0419_01
6 Obtained by querying the server's API: https://ct.googleapis.com/pilot/ct/v1/get-sth
7 https://www.ct-observatory.org/, accessed December 2016

To retrieve the proof, the client needs to know in which record the proof is stored. In the original system, the proof is usually retrieved from the log server by using the hash of the certificate itself, but that would violate our privacy requirements. Instead, we propose that webservers provide the record indices of the certificates for which the server provides the SCTs to the clients (it is not possible to include these in the X.509 certificate as the index is not yet known when the certificate is created). Alternatively, an index structure such as a B+ tree could be used in the typical way that PIR lookups by keywords are done [46, 132].

To check a proof, the auditor needs three things: the certificate itself, the list of sibling hashes, and the signed root. We assume that the auditor has already retrieved the certificate in question. The signed root can be directly retrieved as it does not give any information about the specific certificate. Therefore, the proofs that are stored in the database consist solely of the hashes that help in reconstructing the signed root. We will next consider how these proofs are stored in the database.

Storing proofs in the PIR database

We first count the number of proofs that need to be stored. The log is append-only, and therefore keeps growing. However, expired certificates can safely be removed from the database of proofs. Regular clients will not query for expired certificates, so a fallback to identifying methods is not a problem. Thus, we assume that the database only contains proofs for valid certificates. We estimated this to be about 2^{22} proofs.

In the following we consider a tree containing $2^{\ell-1} < \nu \leq 2^{\ell}$ items. The length of the inclusion proofs in such a Merkle tree is at most ℓ hashes. However, it can be less; for example, the inclusion proof of c_6 in Figure 7.1 consists only of the nodes d and c. For simplicity, we allocate the full ℓ hashes for every proof in the database, resulting in proofs of 32ℓ bytes.

Goldberg's PIR scheme is most efficient when the number of blocks equals the number of field elements per block [81]. It thus makes sense to bundle multiple proofs into a single block (as the number of proofs is exponential in ℓ, while the length of a proof is only linear in ℓ). The location of a proof is then given by its block, and its index within the block (the size of the tree fixes the length of the proofs). When this location is provided by the webserver it should remain fixed while the certificate is valid.

Given the size of a proof and the estimated number of valid TLS certificates we get a storage requirement of $32 \cdot 22 \cdot 2^{22} \approx 0.7 \cdot 2^{32}$ bytes, or about 3 GiB. Therefore, in the following section, we evaluate our algorithm on databases of sizes 1–4 GiB.

We also note that, if the client is willing to reveal a subset of the

database that contains the certificate she seeks, she can reduce the server's computational load at the cost of revealing some information about her query [134]. While trivially downloading the entire subset is one approach, PIR offers a lower communication cost—only about one *block* of information is sent to and from each PIR server—without leaking information about which certificate within the entire subset the client is querying for.

7.4 IMPLEMENTATION AND EVALUATION

We implemented fast matrix multiplication using Strassen's algorithm as an extension to the Percy++ open-source PIR library [73]. We implemented Strassen for the small fields $\mathrm{GF}(2^8)$ and $\mathrm{GF}(2^{16})$—the finite fields with 2^8 and 2^{16} elements respectively—as well as the integers modulo p. All measurements were taken in Ubuntu 12.04.4 LTS running on a machine with eight Intel(R) Xeon(R) E7-8870 CPUs, but each PIR server, which used only one core, was assigned to a different CPU.

7.4.1 *Implementation*

Strassen's algorithm works perfectly when multiplying matrices where all the dimensions are powers of two. In the PIR setting, however, this need not hold. Whenever one or more of the three matrix dimensions (q, r, or s) is odd, we split off the single excess row(s) and/or column(s) and use the naive matrix multiplication algorithm for those products. The resulting dimensions are all even, so that we can do another Strassen recursive step.

Dealing with dimensions that are non-powers of two can be costly. For example, a dimension of $2^\ell - 1$ will incur extra calculations at every step, resulting in a larger computation time than if the dimension were 2^ℓ instead. Hence, our algorithm is designed to dynamically increase the number of queries (by inserting a dummy all-zeroes query) if this yields better performance.

Every recursion step yields a small overhead. Part of this is mitigated by not allocating memory every time, but at small sizes the overhead still trumps the gain possible. We have analyzed when this happens; see Figure 7.2 for an example. We then tuned our implementation to stop recursing at the optimal depth.

7.4.2 *Experiments*

For q less than about 165, the number of additions in Strassen's algorithm is slightly larger than for the naive algorithm due to the multi-

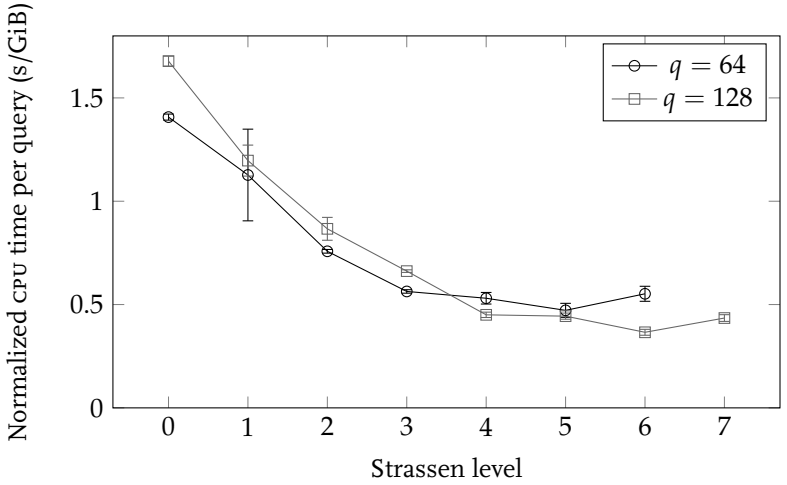


Figure 7.2: Normalized CPU time per query (seconds per GiB of database size) for a 1 GiB database consisting of 32768 32768-byte records over $\mathrm{GF}(2^8)$. We plot different Strassen levels (i.e., depth of recursion in the Strassen algorithm) and two numbers of queries, 64 and 128. Consider the $q = 64$ case. After 6 Strassen steps, the problem size has been reduced to a 1×512 matrix times a 512×512 matrix, and the algorithm bottoms out. In both cases it is better to skip this final reduction step. Error bars are shown, but most are small, and may be difficult to see.

plicative constant of 8/3 (see Table 7.2). However, as explained above, every recursive Strassen step reduces the number of multiplications by a factor of 7/8 or 12.5%, starting with the very first. The small fields and the integers modulo p have in common that multiplication is a lot more expensive than addition; therefore, we expect that even one or two recursive steps of Strassen's algorithm would have a measurable effect on the performance, and the measurements in Figure 7.2 bear this out. The initial dimensions of the problem dictate how many recursive steps of Strassen's algorithm can be applied, as each dimension is cut in half at each step. In practice, we expect that it is the number of queries that is the limiting factor (that is, q will be much smaller than either r or s), so that is what we focus on.

Figure 7.3 compares the performance of our new algorithm with the one in the 0.9.0 release of Percy++. All measurements are done over $\mathrm{GF}(2^8)$, as that is the most efficient field supported by Percy++. We notice that Percy++ slows down considerably when more queries are used (we suspect cache issues may be to blame for this, but it was a completely repeatable effect). Our scheme does not suffer from this problem, and indeed produces the desired *decrease* in per-query cost as

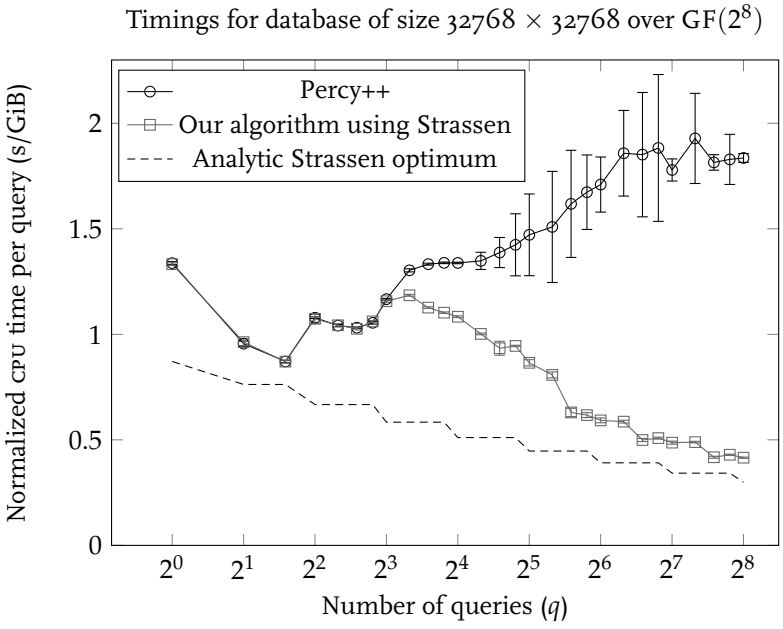


Figure 7.3: Comparison between the original $\mathrm{GF}(2^8)$ PIR implementation in Percy++ and our new version using Strassen. For the first part of the graph, the new and original algorithms give the same results, as the Strassen code is not invoked for small problem sizes. We can also easily see the effect of the hand-optimized loop in Percy++ for handling $q \leq 3$. We also compare our algorithm to the best theoretical improvements that using Strassen's algorithm can provide, using the fastest per-query time of the original Percy++ code ($q = 3$) as a reference point. Error bars are shown, but most are small, and may be difficult to see. The peak memory usage of our algorithm (for $q = 256$) was 1422 MiB, whereas Percy++ used 1060 MiB.

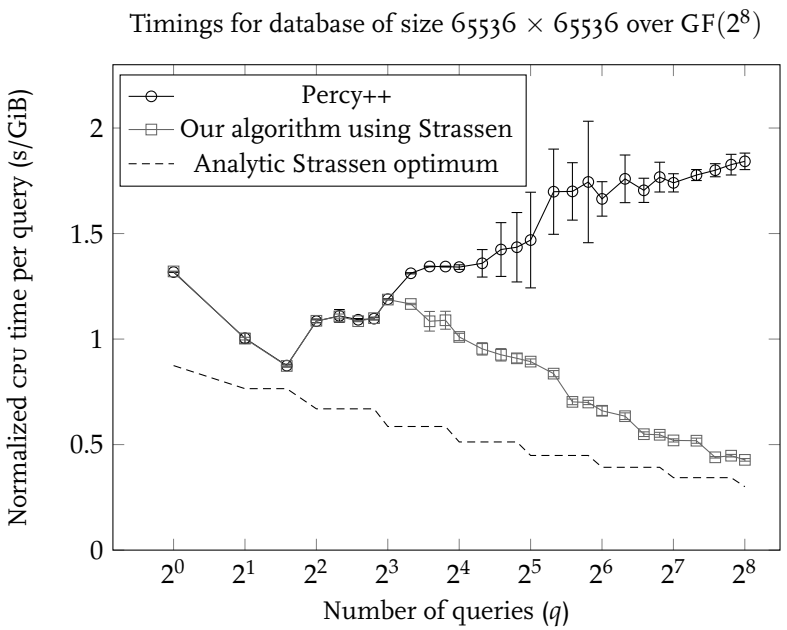


Figure 7.4: We repeat the experiment of Figure 7.3, but with a 4 GiB database consisting of 65536 65536-byte records. The peak memory usage of our algorithm (for $q = 256$) was 5556 MiB, whereas Percy++ used 4148 MiB.

the number of queries increases. We observe a 4.4-fold performance improvement over Percy++ for $q = 256$ simultaneous queries.

We have also drawn the analytical improvements we expect from using Strassen's algorithm. Figure 7.3 shows the theoretical bound of the optimal Strassen gain that would be possible, measured against the *fastest* per-query time (obtained at $q = 3$) measured for the original Percy++ code. For example, for 256 queries, this gain would be $(\frac{7}{8})^8 \approx 0.344$. We see that we are quite close to this value, even though we always skip the final Strassen step.

For each Strassen recursion step, our algorithm incurs an extra subproblem of 1/4 the size, which needs to be stored in memory. The extra memory consumption as a result of this is at most a factor of $\frac{1}{4} + (\frac{1}{4})^2 + \cdots = \frac{1}{3}$. This is confirmed by the memory usage given in Figures 7.3 and 7.4.

CERTIFICATE TRANSPARENCY. Figure 7.4 shows performance measures for a 4 GiB database, a size that nicely matches up with a log server's database of inclusions proofs. Again we see that using Strassen's algorithm gives a significant performance gain over just using

single queries.

While batching queries results in a significantly lower processing time per query, the latency does increase. However, this is not a problem for auditing proofs of inclusion, as they are performed asynchronously. In particular, the goal is to detect misbehaving log servers, and not to protect users against falsely issued certificates directly. Some latency is therefore acceptable.

If a lower latency is required, the algorithm can easily be parallelized. Each of the seven Strassen subproblems is completely independent from the others, and creating these and recombining the result is very cheap. While we did not implement parallelization, we expect the overhead of doing so to be extremely small.

7.5 CONCLUSIONS

In this chapter we showed how we can significantly speed up PIR queries if we allow the server to batch queries, and answer them simultaneously. Such an idea was proposed earlier in the setting of batch codes, but that proposal required coordination among the clients, which is not desirable in the PIR setting.

We analyzed batch codes in the setting of Goldberg's PIR scheme and have shown that essentially they provide a fast method for doing matrix multiplication under specific constraints on the matrices. However, since multi-client PIR in Goldberg's scheme is essentially a matrix multiplication, we can use our method to obtain sublinear scaling in the number of queries *without* requiring the queries to have been created by a single client or cooperating clients.

We described how multi-client PIR can be used to make certificate transparency more privacy friendly. We implemented Strassen's algorithm as part of Percy++ and have shown that we indeed manage to get a significant speedup when batching multi-client queries. While further system-level optimizations to Percy++ (which is already heavily optimized) might give comparable speedups in absolute terms, these will almost surely be a constant factor, whereas our algorithmic improvements *increase* with the number of simultaneous queries. Furthermore, any such system-level optimizations are likely to be able to be combined with our algorithmic improvements to yield compounded benefits.

Our implementation is open source and has been incorporated into the 1.0 release of Percy++ [74].

8 CONCLUSIONS

In the beginning of this thesis we set out to explore whether we could build new practical systems that offer security and privacy simultaneously, without relying on legal and procedural measures. To answer this question, we reexamined the notion of revocable privacy, and introduced four practical systems that offer security and privacy simultaneously.

8.1 OVERVIEW

Underlying the research question are two comparisons in which privacy has to yield to other principles that are deemed more important. The first comparison we questioned is the one of security versus privacy, and the belief that increasing privacy necessarily reduces security. In Chapter 3 we reexamined the concept of revocable privacy. In summary, a system that implements revocable privacy offers full anonymity to all participants that follow the rules, while it may reduce the anonymity of those that do not.

In Chapter 3 we highlighted some use cases, in the real as well as digital world, that could benefit from such an approach, and highlighted a few existing solutions that already offer revocable privacy. To answer the first aspect of the research question, we introduced two new systems in Chapters 4 and 5 that also implemented revocable privacy.

Chapter 4 introduced the vote-to-link system. This system allows users of an online platform, such as Wikipedia, to act anonymously, while the system can protect itself against misbehavior from these anonymous users. To do so, the vote-to-link system enables moderators to vote on malicious actions. If a sufficient number of votes are combined, all other actions by that same user within a predetermined time frame can be identified. This linking ensures that the malicious actions by that user can be distinguished from all the benign actions, and can therefore be dealt with much more easily.

Chapter 5 introduces a new distributed encryption scheme. This scheme implements a threshold rule: a user is not allowed to perform an action at more than k different locations. If she does, her identity will become known. Whereas users in the vote-to-link system interact with the system themselves, the distributed encryption scheme is non-interactive. Users are observed by sensors, that process their identity on

the user's behalf. Chapter 3 shows that such a non-interactive threshold primitive is particularly suited for real-world applications where the user's identity can be derived, and where it would be too costly to interact directly with the user. In particular, we applied it to two license plate scenarios, showing that it is possible to identify suspicious cars, without revealing any information about those cars that do not behave suspiciously.

These three chapters together show that it is indeed possible to build new systems that offer both security and privacy, and thereby also refute the general claim that more privacy always comes at the cost of less security. This brings us to the second aspect of the research question, and the second comparison underlying it: are these systems practical?

Chapters 4 to 7 show that the systems we developed, and that all offer both security and privacy, are practical. In Chapter 6 we presented a new revocation scheme for attribute-based credentials that is particularly efficient for both users and verifiers (typical privacy friendly revocation schemes are not efficient enough for at least one of these). To enable such efficiency, we trade a very small amount of anonymity to obtain performance that is comparable to traditional non-private revocation schemes. Similarly, the basic vote-to-link scheme from Chapter 4 is also only a constant factor slower than the naive non-private solution.

The multi-client private information retrieval (PIR) scheme from Chapter 7 allows multiple clients to retrieve records from a database, without revealing to that database which records they retrieved. Our new scheme batches queries from many clients to offer an algorithmic speed-up over the regular scheme. With this speedup, PIR is practical enough to use it to make certificate transparency privacy friendly.

However, even this batched scheme is still considerably less efficient than the naive non-private scheme. Similarly, the distributed encryption scheme from Chapter 5 is considerably less efficient than its non-private counterpart. So, while both schemes are fast enough to be used in practical situations, using them does incur an efficiency cost.

This cost in efficiency might be an artifact of our current designs. However, we believe that just as the performance of traditional IT-PIR is bounded away from its non-private counterpart, the revocable privacy based solutions may, in some cases, be inherently less efficient.

8.2 FUTURE WORK

Chapter 3 surveys a number of scenarios that could benefit from the notion of revocable privacy. In this thesis we proposed solutions to some of these, however, many of them remain as of yet unsolved. In particular, it would be very interesting to find schemes that implement predicate

based rules, so that different primitives, such as distributed encryption, can be combined to implement more complicated rules.

In particular, implementing a larger class of rules using revocable privacy systems would further weaken the claim that privacy always comes at the cost of security. As suggested in Chapter 3, rules based on predicates or more general threshold rules might be relatively easy to implement.

Focusing on our specific schemes, it would be interesting to see if the moderator anonymous versions of the vote-to-link scheme can be designed so that the performance degradation for the service provider is somewhat lower.

For the distributed encryption scheme it would be interesting to see whether we can prove a lower bound on its performance, and whether ideas from Chapter 6—trading a small amount of privacy to obtain much higher performance—can be applied to the distributed encryption scheme as well.

8.3 GENERAL CONCLUSIONS

This work confirms that it is possible to achieve security and privacy simultaneously in various situations, even if, intuitively, this would seem impossible. Moreover, the solutions we proposed here are practical: they can be deployed in real-world scenarios. This shows that security can often be achieved without negatively impacting privacy.

On the other hand, achieving both security and privacy does not come for free. If privacy-friendly solutions do not yet exist (fortunately many of them do), they have to be designed and implemented, which is not always easy. And even if they do exist, they might be less efficient—though usually not prohibitively so—than non-private solutions.

Since privacy and security can often be obtained simultaneously, the catch-all argument that we cannot have privacy because it would (always) come at the cost of security, is no longer universally valid. Instead, I hope that this thesis, and the ideas contained therein, contribute to a more balanced discussion about the benefits and costs of privacy.

BIBLIOGRAPHY

[1] Gora Adj, Alfred Menezes, Thomaz Oliveira, and Francisco Rodríguez-Henríquez. "Weakness of $\mathbb{F}_{3^{6\cdot1429}}$ and $\mathbb{F}_{2^{4\cdot3041}}$ for discrete logarithm cryptography". In: *Finite Fields and Their Applications* 32 (2015), pp. 148–170. ISSN: 1071-5797. DOI: 10.1016/j.ffa.2014.10.009 (cit. on p. 15).

[2] Gora Adj, Alfred Menezes, Thomaz Oliveira, and Francisco Rodríguez-Henríquez. "Weakness of $\mathbb{F}_{3^{6\cdot509}}$ for Discrete Logarithm Cryptography". In: PAIRING 2013. LNCS vol. 8365. Springer, 2014, pp. 20–44. DOI: 10.1007/978-3-319-04873-4_2 (cit. on p. 15).

[3] Charu C. Aggarwal and Philip S. Yu, eds. *A General Survey of Privacy-Preserving Data Mining – Models and Algorithms*. Vol. 34. Advances in Database Systems. Springer, 2008. ISBN: 978-0-387-70991-8. DOI: 10.1007/978-0-387-70992-5 (cit. on p. 54).

[4] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. "XPIR : Private Information Retrieval for Everyone". In: *PoPETs* 2016.2 (2016), pp. 155–174. DOI: 10.1515/popets-2016-0010 (cit. on p. 168).

[5] Carlos Aguilar-Melchor and Philippe Gaborit. "A Lattice-Based Computationally-Efficient Private Information Retrieval Protocol". In: WEWORC 2007. 2007. URL: https://eprint.iacr.org/2007/446 (cit. on p. 168).

[7] Elli Androulaki, Seung Geol Choi, Steven M. Bellovin, and Tal Malkin. "Reputation Systems for Anonymous Networks". In: PETS 2008. LNCS vol. 5134. Springer, 2008, pp. 202–218. DOI: 10.1007/978-3-540-70630-4_13 (cit. on p. 93).

[8] D. F. Aranha and C. P. L. Gouvêa. *RELIC is an Efficient LIbrary for Cryptography*. URL: https://github.com/relic-toolkit/relic (cit. on pp. 91, 133).

[9] Giuseppe Ateniese, Dawn Xiaodong Song, and Gene Tsudik. "Quasi-Efficient Revocation in Group Signatures". In: FC 2002. LNCS vol. 2357. Springer, 2003, pp. 183–197. DOI: 10.1007/3-540-36504-4_14 (cit. on pp. 136, 164, 165).

[10] Man Ho Au, Sherman S. M. Chow, and Willy Susilo. "Short E-Cash". In: INDOCRYPT 2005. LNCS vol. 3797. Springer, 2005, pp. 332–346. DOI: 10.1007/11596219_27 (cit. on p. 52).

[11] Man Ho Au, Willy Susilo, and Yi Mu. "Constant-Size Dynamic *k*-TAA". In: SCN 2006. LNCS vol. 4116. Springer, 2006, pp. 111–125. DOI: 10.1007/11832072_8 (cit. on pp. 27–30, 91).

[12] Amos Beimel and Yoav Stahl. "Robust Information-Theoretic Private Information Retrieval". In: *Journal of Cryptology* 20.3 (2007), pp. 295–321. ISSN: 1432-1378. DOI: 10.1007/s00145-007-0424-2 (cit. on p. 167).

[13] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. "A Concrete Security Treatment of Symmetric Encryption". In: FOCS '97. IEEE, 1997, pp. 394–403. DOI: 10.1109/SFCS.1997.646128 (cit. on p. 70).

[14] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-speed high-security signatures". In: *Journal of Cryptographic Engineering* 2.2 (2012), pp. 77–89. ISSN: 2190-8516. DOI: 10.1007/s13389-012-0027-1 (cit. on p. 161).

[15] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. "Elligator: elliptic-curve points indistinguishable from uniform random strings". In: CCS 2013. ACM, 2013, pp. 967–980. DOI: 10.1145/2508859.2516734 (cit. on p. 101).

[16] Joachim Biskup and Ulrich Flegel. "Transaction-Based Pseudonyms in Audit Data for Privacy Respecting Intrusion Detection". In: RAID 2000. LNCS vol. 1907. Springer, 2000, pp. 28–48. DOI: 10.1007/3-540-39945-3_3 (cit. on pp. 50, 51).

[17] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Communications of the ACM* 13.7 (July 1970), pp. 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692 (cit. on p. 162).

[18] Carlo Blundo and Douglas R. Stinson. "Anonymous secret sharing schemes". In: *Discrete Applied Mathematics* 77.1 (1997), pp. 13–28. ISSN: 0166-218x. DOI: 10.1016/S0166-218X(97)89208-6 (cit. on p. 94).

[19] Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. "How the Estonian Tax and Customs Board Evaluated a Tax Fraud Detection System Based on Secure Multi-party Computation". In: FC 2015. LNCS vol. 8975. Springer, 2015, pp. 227–234. DOI: 10.1007/978-3-662-47854-7_14 (cit. on pp. 50, 54).

[20] Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, and Riivo Talviste. "Students and Taxes: a Privacy-Preserving Study Using Secure Computation". In: *PoPETs* 2016.3 (2016), pp. 117–135. DOI: 10.1515/popets-2016-0019 (cit. on p. 54).

[21] Dan Boneh and Xavier Boyen. "Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups". In: *Journal of Cryptology* 21.2 (Apr. 2008), pp. 149–177. ISSN: 1432-1378. DOI: 10.1007/s00145-007-9005-7 (cit. on p. 19).

[22] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: CRYPTO 2001. LNCS vol. 2139. Springer, 2001, pp. 213–229. DOI: 10.1007/3-540-44647-8_13 (cit. on pp. 18, 107).

[23] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *Journal of Cryptology* 17.4 (Sept. 2004), pp. 297–319. ISSN: 1432-1378. DOI: 10.1007/s00145-004-0314-9 (cit. on p. 14).

[24] Dan Boneh and Hovav Shacham. "Group signatures with verifier-local revocation". In: CCS 2004. ACM, 2004, pp. 168–177. DOI: 10.1145/1030083.1030106 (cit. on pp. 136, 143, 145, 164, 165).

[25] Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, USA, 2000 (cit. on p. 26).

[26] Stefan Brands, Liesje Demuynck, and Bart De Decker. "A Practical System for Globally Revoking the Unlinkable Pseudonyms of Unknown Users". In: ACISP 2007. LNCS vol. 4586. Springer, 2007, pp. 400–415. DOI: 10.1007/978-3-540-73458-1_29 (cit. on p. 164).

[27] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. "Direct anonymous attestation". In: CCS 2004. ACM, 2004, pp. 132–145. DOI: 10.1145/1030083.1030103 (cit. on p. 165).

[28] Ernie Brickell, Jan Camenisch, and Liqun Chen. "The DAA scheme in context". In: *Trusted Computing*. Vol. 6. Professional Applications of Computing. Institution of Engineering and Technology, 2005. Chap. 5, pp. 143–174. ISBN: 978-0-863-41525-8. DOI: 10.1049/PBPC006E_ch5 (cit. on pp. 136, 164, 165).

[29] BSI. *Advanced Security Mechanisms for Machine Readable Travel Documents – Extended Access Control (EAC)*. Tech. rep. TR-03110. Bonn, Germany: Bundesamt für Sicherheit in der Informationstechnik (BSI), 2006 (cit. on p. 159).

[30] Jan Camenisch, Manu Drijvers, and Anja Lehmann. *Anonymous Attestation Using the Strong Diffie Hellman Assumption Revisited*. Cryptology ePrint Archive, Report 2016/663. 2016. URL: http://eprint.iacr.org/2016/663 (cit. on pp. 28, 29).

[31] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. "How to win the clonewars: efficient periodic *n*-times anonymous authentication". In: CCS 2006. ACM, 2006, pp. 201–210. DOI: 10.1145/1180405.1180431 (cit. on pp. 50, 52, 98).

[32] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. "Compact E-Cash". In: EUROCRYPT 2005. LNCS vol. 3494. Springer, 2005, pp. 302–321. DOI: 10 . 1007 / 11426639 _ 18 (cit. on p. 52).

[33] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. "An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials". In: PKC 2009. LNCS vol. 5443. Springer, 2009, pp. 481–500. DOI: 10.1007/978-3-642-00468-1_27 (cit. on pp. 135, 164, 165).

[34] Jan Camenisch, Ioannis Krontiris, Anja Lehmann, Gregory Neven, Christian Paquin, Kai Rannenberg, and Harald Zwingelberg. *D2.1 Architecture for Attribute-based Credential Technologies*. Tech. rep. ABC4Trust, 2011 (cit. on pp. 26, 135).

[35] Jan Camenisch and Anna Lysyanskaya. "A Signature Scheme with Efficient Protocols". In: SCN 2002. LNCS vol. 2576. Springer, 2003, pp. 268–289. DOI: 10.1007/3-540-36413-7_20 (cit. on p. 26).

[36] Jan Camenisch and Anna Lysyanskaya. "An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation". In: EUROCRYPT 2001. LNCS vol. 2045. Springer, 2001, pp. 93–118. DOI: 10 . 1007 / 3 - 540 - 44987 - 6 _ 7 (cit. on pp. 8, 160).

[37] Jan Camenisch and Anna Lysyanskaya. "Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials". In: CRYPTO 2002. LNCS vol. 2442. Springer, 2002, pp. 61–76. DOI: 10.1007/3-540-45708-9_5 (cit. on p. 164).

[38] Jan Camenisch and Victor Shoup. "Practical Verifiable Encryption and Decryption of Discrete Logarithms". In: CRYPTO 2003. LNCS vol. 2729. Springer, 2003, pp. 126–144. DOI: 10.1007/978-3-540-45146-4_8 (cit. on p. 164).

[39] Jan Camenisch and Markus Stadler. "Efficient Group Signature Schemes for Large Groups (Extended Abstract)". In: CRYPTO '97. LNCS vol. 1294. Springer, 1997, pp. 410–424. DOI: 10.1007/BFb0052252 (cit. on p. 26).

[40] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. "Algebraic MACs and Keyed-Verification Anonymous Credentials". In: CCS 2014. ACM, 2014, pp. 1205–1216. DOI: 10.1145/2660267.2660328 (cit. on p. 27).

[41] Sanjit Chatterjee and Alfred Menezes. "On cryptographic protocols employing asymmetric pairings – The role of Ψ revisited". In: *Discrete Applied Mathematics* 159.13 (2011), pp. 1311–1322. ISSN: 0166-218X. DOI: 10.1016/j.dam.2011.04.021 (cit. on pp. 18, 19).

[42] Sanjit Chatterjee and Palash Sarkar. *Practical Hybrid (Hierarchical) Identity-Based Encryption Schemes Based on the Decisional Bilinear Diffie-Hellman Assumption*. Tech. rep. CACR 2010-20. Center for Applied Cryptographic Research, University of Waterloo, ON, Canada, 2010. URL: http://cacr.uwaterloo.ca/techreports/2010/cacr2010-20.pdf (cit. on p. 18).

[43] David Chaum. "Blind Signatures for Untraceable Payments". In: CRYPTO '82. Springer, 1983, pp. 199–203. DOI: 10.1007/978-1-4757-0602-4_18 (cit. on pp. 44, 52).

[44] David Chaum, Amos Fiat, and Moni Naor. "Untraceable Electronic Cash". In: CRYPTO '88. LNCS vol. 403. Springer, 1990, pp. 319–327. DOI: 10.1007/0-387-34799-2_25 (cit. on pp. 6, 31, 50, 52).

[45] David Chaum and Eugène van Heyst. "Group Signatures". In: EUROCRYPT '91. LNCS vol. 547. Springer, 1991, pp. 257–265. DOI: 10.1007/3-540-46416-6_22 (cit. on pp. 53, 93).

[46] Benny Chor, Niv Gilboa, and Moni Naor. *Private Information Retrieval by Keywords*. Technical Report TR CS0917. Department of Computer Science, Technion, Israel, 1997 (cit. on pp. 167, 181).

[47] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. "Private Information Retrieval". In: FOCS 1995. IEEE, 1995, pp. 41–50. DOI: 10.1109/SFCS.1995.492461 (cit. on p. 167).

[48] Sherman S. M. Chow. "Real Traceable Signatures". In: SAC 2009. LNCS vol. 5867. Springer, 2009, pp. 92–107. DOI: 10.1007/978-3-642-05445-7_6 (cit. on p. 153).

[49] Julie E. Cohen. "What Privacy is For". In: *Harvard Law Review* 126.7 (2013), pp. 1904–1933 (cit. on pp. 2, 3).

[50] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280 (Proposed Standard). Updated by RFC 6818. Internet Engineering Task Force, May 2008 (cit. on pp. 163, 164).

[51] Don Coppersmith and Shmuel Winograd. "Matrix Multiplication via Arithmetic Progressions". In: *Journal of Symbolic Computation* 9.3 (Mar. 1990), pp. 251–280. ISSN: 0747-7171. DOI: 10.1016/S0747-7171(08)80013-2 (cit. on p. 174).

[52] Ronald Cramer, Ivan Damgård, and Yuval Ishai. "Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation". In: TCC 2005. LNCS vol. 3378. Springer, 2005, pp. 342–362. DOI: 10.1007/978-3-540-30576-7_19 (cit. on pp. 25, 115, 118).

[53] David Davenport. "Anonymity on the Internet: Why the Price May Be Too High". In: *Communications of the ACM* 45.4 (2002), pp. 33–35. ISSN: 0001-0782. DOI: 10.1145/505248.505267 (cit. on p. 5).

[54] Cécile Delerablée and David Pointcheval. "Dynamic Threshold Public-Key Encryption". In: CRYPTO 2008. LNCS vol. 5157. Springer, 2008, pp. 317–334. DOI: 10.1007/978-3-540-85174-5_18 (cit. on p. 94).

[55] Yvo Desmedt and Yair Frankel. "Threshold cryptosystems". In: CRYPTO '89. LNCS vol. 435. Springer, 1990, pp. 307–315. DOI: 10.1007/0-387-34805-0_28 (cit. on p. 94).

[56] Casey Devet and Ian Goldberg. "The Best of Both Worlds: Combining Information-Theoretic and Computational PIR for Communication Efficiency". In: PETS 2014. LNCS vol. 8555. Springer, 2014, pp. 63–82. DOI: 10.1007/978-3-319-08506-7_4 (cit. on pp. 167, 178).

[57] Casey Devet, Ian Goldberg, and Nadia Heninger. "Optimally Robust Private Information Retrieval". In: USENIX 2012. USENIX, 2012, pp. 269–283. URL: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/devet (cit. on pp. 167, 170).

[58] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. "Towards Measuring Anonymity". In: PETS 2002. LNCS vol. 2482. Springer, 2002, pp. 54–68. DOI: 10.1007/3-540-36467-6_5 (cit. on p. 34).

[59] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. "Tor: The Second-Generation Onion Router". In: USENIX 2004. 2004, pp. 303–320. URL: https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router (cit. on p. 32).

[60] John R. Douceur. "The Sybil Attack". In: IPTPS 2002. LNCS vol. 2429. Springer, 2002, pp. 251–260. DOI: 10.1007/3-540-45748-8_24 (cit. on p. 63).

[61] ECRYPT II. *ECRYPT II Yearly Report on Algorithms and Key Lengths (2011-2012)*. Revision 1.0. 2012 (cit. on p. 161).

[62] ECRYPT-CSA. *D5.2: Algorithms, Key Size and Protocols Report (2016)*. Revision 1.0. 2016 (cit. on p. 161).

[63] Taher ElGamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *IEEE Transactions on Information Theory* 31.4 (July 1985), pp. 469–472. ISSN: 0018-9448. DOI: 10.1109/TIT.1985.1057074 (cit. on p. 70).

[64] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: CRYPTO '86. LNCS vol. 263. Springer, 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12 (cit. on pp. 26, 146).

[65] Fox-IT BV. *Black Tulip: Report of the investigation into the DigiNotar Certificate Authority breach*. Delft, The Netherlands, Aug. 2012 (cit. on p. 178).

[66] Matthew K. Franklin. "A survey of key evolving cryptosystems". In: *International Journal of Security and Networks* 1.1/2 (2006), pp. 46–53. DOI: 10.1504/IJSN.2006.010822 (cit. on p. 98).

[67] Jun Furukawa and Shoko Yonezawa. "Group Signatures with Separate and Distributed Authorities". In: SCN 2004. LNCS vol. 3352. Springer, 2005, pp. 77–90. DOI: 10.1007/978-3-540-30598-9_6 (cit. on p. 93).

[68] Steven D. Galbraith. "Pairings". In: *Advances in Elliptic Curve Cryptography*. Vol. 317. London Mathematical Society Lecture Note Series. Cambridge University Press, 2005. Chap. IX. ISBN: 978-0-511-11029-0 (cit. on p. 14).

[69] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. "Pairings for cryptographers". In: *Discrete Applied Mathematics* 156.16 (2008), pp. 3113–3121. ISSN: 0166-218x. DOI: 10.1016/j.dam.2007.12.010 (cit. on pp. 14, 15).

[70] Essam Ghadafi. "Efficient Distributed Tag-Based Encryption and Its Application to Group Signatures with Efficient Distributed Traceability". In: LATINCRYPT 2014. LNCS vol. 8895. Springer, 2015, pp. 327–347. DOI: 10.1007/978-3-319-16295-9_18 (cit. on pp. 53, 93).

[72] Ian Goldberg. "Improving the Robustness of Private Information Retrieval". In: S&P 2007. IEEE, 2007, pp. 131–148. DOI: 10.1109/SP.2007.23 (cit. on pp. 167–170).

[73] Ian Goldberg, Casey Devet, Paul Hendry, and Ryan Henry. *Percy++ project on SourceForge*. http://percy.sourceforge.net. Version 0.9.0. Accessed Sept. 2014. 2013 (cit. on pp. 169, 182).

[74] Ian Goldberg, Casey Devet, Wouter Lueks, Ann Yang, Paul Hendry, and Ryan Henry. *Percy++ project on SourceForge*. http://percy.sourceforge.net/. Version 1.0. Accessed Nov. 2014. 2014 (cit. on pp. 12, 186).

[75] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. "Breaking '128-bit Secure' Supersingular Binary Curves – (Or How to Solve Discrete Logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 357}}$)". In: CRYPTO 2014. LNCS vol. 8617. Springer, 2014, pp. 126–145. DOI: 10.1007/978-3-662-44381-1_8 (cit. on p. 15).

[76] Jens Groth. "A Verifiable Secret Shuffle of Homomorphic Encryptions". In: *Journal of Cryptology* 23.4 (Oct. 2010), pp. 546–579. ISSN: 1432-1378. DOI: 10.1007/s00145-010-9067-9 (cit. on pp. 88, 89).

[77] Mida Guillermo, Keith M. Martin, and Christine M. O'Keefe. "Providing Anonymity in Unconditionally Secure Secret Sharing Schemes". In: *Designs, Codes and Cryptography* 28.3 (Apr. 2003), pp. 227–245. ISSN: 1573-7586. DOI: 10.1023/A:1024198519111 (cit. on p. 94).

[78] Daniel M. Hein, Johannes Wolkerstorfer, and Norbert Felber. "ECC Is Ready for RFID - A Proof in Silicon". In: SAC 2008. LNCS vol. 5381. Springer, 2009, pp. 401–413. DOI: 10.1007/978-3-642-04159-4_26 (cit. on p. 161).

[79] Ryan Henry and Ian Goldberg. "Formalizing Anonymous Blacklisting Systems". In: S&P 2011. IEEE, 2011, pp. 81–95. DOI: 10.1109/SP.2011.13 (cit. on pp. 53, 63).

[80] Ryan Henry and Ian Goldberg. "Thinking inside the BLAC box: smarter protocols for faster anonymous blacklisting". In: WPES 2013. ACM, 2013, pp. 71–82. DOI: 10.1145/2517840.2517855 (cit. on p. 53).

[81] Ryan Henry, Yizhou Huang, and Ian Goldberg. "One (Block) Size Fits All: PIR and SPIR with Variable-Length Records via Multi-Block Queries". In: NDSS 2013. The Internet Society, 2013. URL: http://internetsociety.org/doc/one-block-size-fits-all-pir-and-spir-variable-length-records-multi-block-queries (cit. on pp. 168, 170, 173, 181).

[82] Jaap-Henk Hoepman. "Revocable Privacy". In: *ENISA Quarterly Review* 5.2 (June 2009), pp. 16–17. ISSN: 1830-3609 (cit. on pp. 5, 6, 32, 34, 37).

[83] Jaap-Henk Hoepman and David Galindo. "Non-interactive Distributed Encryption: A New Primitive for Revocable Privacy". In: WPES 2011. ACM, 2011, pp. 81–92. DOI: 10.1145/2046556.2046567 (cit. on pp. 32, 51, 97, 98, 104, 106–108, 121, 124).

[84] Jaap-Henk Hoepman, Bert-Jaap Koops, and Wouter Lueks. "Anoniem misdaad melden via Internet: technische en juridische risico's". Dutch. In: *Nederlands Juristenblad* 43 (Dec. 2014), pp. 3056–3063 (cit. on pp. 1, 218).

[85] Jaap-Henk Hoepman, Bert-Jaap Koops, and Wouter Lueks. *Haalbaarheid van een anoniem misdaadmeldpunt via het Internet. Een quickscan*. Dutch. Nijmegen/Den Haag, 2014. URL: https://www.wodc.nl/onderzoeksdatabase/2398-wenselijkheid-en-haalbaarheid-van-online-anoniem-melden-van-misdaden.aspx (cit. on p. 1).

[87] IBM Research Zürich Security Team. *Specification of the Identity Mixer Cryptographic Library, version 2.3.4*. Tech. rep. IBM Research, Zürich, Feb. 2012 (cit. on pp. 8, 14, 160, 164).

[88] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. "Batch codes and their applications". In: STOC 2004. ACM, 2004, pp. 262–271. DOI: 10.1145/1007352.1007396 (cit. on pp. 168, 170–173).

[89] Gene Itkis. "Forward Security – Adaptive Cryptography: Time Evolution". In: *Handbook of Information Security*. Vol. 3. John Wiley and Sons, 2006, pp. 927–944. ISBN: 978-0-471-64832-1 (cit. on p. 98).

[90] Antoine Joux. "A One Round Protocol for Tripartite Diffie-Hellman". In: ANTS IV. LNCS vol. 1838. Springer, 2000, pp. 385–394. DOI: 10.1007/10722028_23 (cit. on p. 18).

[91] Mitchell Kapor. *Architecture is Politics (and Politics is Architecture)*. Original at http://blog.kapor.com/?p=29 last accessed January 3, 2017 via https://web.archive.org/web/20060615043915/http://blog.kapor.com/?p=29. 2006 (cit. on pp. 6, 36).

[92] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: ASIACRYPT 2010. LNCS vol. 6477. Springer, 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11 (cit. on p. 90).

[93] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. 2nd ed. Chapman & Hall/CRC Cryptography and Network Security Series. CRC press, Taylor & Francis Group, 2014. ISBN: 9781466570269 (cit. on pp. 13, 15, 17, 89).

[94] Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. "Traceable Signatures". In: EUROCRYPT 2004. LNCS vol. 3027. Springer, 2004, pp. 571–589. DOI: 10.1007/978-3-540-24676-3_34 (cit. on p. 93).

[95] Joe Kilian and Erez Petrank. "Identity Escrow". In: CRYPTO '98. LNCS vol. 1462. Springer, 1998, pp. 169–185. DOI: 10 . 1007 / BFb0055727 (cit. on p. 158).

[96] Taechan Kim and Razvan Barbulescu. "Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case". In: CRYPTO 2016. LNCS vol. 9814. Springer, 2016, pp. 543–571. DOI: 10.1007/978-3-662-53018-4_20 (cit. on p. 15).

[97] Bert-Jaap Koops, Bryce Clayton Newell, Tjerk Timan, Ivan Škorvánek, Tom Chokrevski, and Maša Galič. "A Typology of Privacy". In: *University of Pennsylvania Journal of International Law* 38.2 (2017), pp. 483–575. URL: http://scholarship.law.upenn.edu/jil/vol38/iss2/4 (cit. on p. 3).

[98] Eyal Kushilevitz and Rafail Ostrovsky. "Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval". In: FOCS '97. IEEE, 1997, pp. 364–373. DOI: 10.1109/SFCS.1997.646125 (cit. on pp. 167, 168).

[99] Mirosław Kutylowski, Łukasz Krzywiecki, Przemysław Kubiak, and Michał Koza. "Restricted Identification Scheme and Diffie-Hellman Linking Problem". In: INTRUST 2011. LNCS vol. 7222. Springer, 2011, pp. 221–238. DOI: 10.1007/978-3-642-32298-3_15 (cit. on p. 164).

[100] Jorn Lapon, Markulf Kohlweiss, Bart De Decker, and Vincent Naessens. "Analysis of Revocation Strategies for Anonymous Idemix Credentials". In: CMS 2011. LNCS vol. 7025. Springer, 2011, pp. 3–17. DOI: 10.1007/978-3-642-24712-5_1 (cit. on pp. 163, 165).

[101] Ben Laurie, Adam Langley, and Emilia Kasper. *Certificate Transparency*. RFC 6962 (Experimental). Internet Engineering Task Force, June 2013. URL: http://www.ietf.org/rfc/rfc6962.txt (cit. on p. 178).

[102] Ronald E. Leenes. "Do they know me? Deconstructing identifiability". In: *University of Ottawa Law and Technology Journal* 4.1&2 (2007), pp. 135–161 (cit. on p. 33).

[103] Anja Lehmann et al. *Survey and Analysis of Existing eID and Credential Systems*. Tech. rep. Deliverable D32.1. FutureID, 2013 (cit. on p. 135).

[104] Jiangtao Li, Ninghui Li, and Rui Xue. "Universal Accumulators with Efficient Nonmembership Proofs". In: ACNS 2007. LNCS vol. 4521. Springer, 2007, pp. 253–269. DOI: 10.1007/978-3-540-72738-5_17 (cit. on p. 165).

[105] Benoît Libert, Thomas Peters, and Moti Yung. "Group Signatures with Almost-for-Free Revocation". In: CRYPTO 2012. LNCS vol. 7417. Springer, 2012, pp. 571–589. DOI: 10.1007/978-3-642-32009-5_34 (cit. on p. 165).

[106] Benoît Libert and Moti Yung. "Non-interactive CCA-Secure Threshold Cryptosystems with Adaptive Security: New Framework and Constructions". In: TCC 2012. LNCS vol. 7194. Springer, 2012, pp. 75–93. DOI: 10.1007/978-3-642-28914-9_5 (cit. on p. 94).

[107] Peter Lofgren and Nicholas Hopper. "FAUST: Efficient, TTP-Free Abuse Prevention by Anonymous Whitelisting". In: WPES 2011. ACM, 2011, pp. 125–130. DOI: 10.1145/2046556.2046572 (cit. on p. 166).

[108] Wouter Lueks, Gergely Alpár, Jaap-Henk Hoepman, and Pim Vullers. "Fast Revocation of Attribute-Based Credentials for Both Users and Verifiers". In: IFIP SEC 2015. IFIP AICT vol. 455. Springer, 2015, pp. 463–478. DOI: 10.1007/978-3-319-18467-8_31 (cit. on pp. 12, 217).

[109] Wouter Lueks, Gergely Alpár, Jaap-Henk Hoepman, and Pim Vullers. "Fast revocation of attribute-based credentials for both users and verifiers". In: *Computers & Security* 67 (2017), pp. 308–323. ISSN: 0167-4048. DOI: 10.1016/j.cose.2016.11.018 (cit. on pp. 12, 217).

[110] Wouter Lueks, Maarten H. Everts, and Jaap-Henk Hoepman. *Revocable Privacy 2012 – use cases*. Tech. rep. 35627. TNO, 2012 (cit. on pp. 32, 34, 39).

[111] Wouter Lueks, Maarten H. Everts, and Jaap-Henk Hoepman. "Revocable Privacy: Principles, Use Cases, and Technologies". In: APF 2015. LNCS vol. 9484. Springer, 2016, pp. 124–143. DOI: 10.1007/978-3-319-31456-3_7 (cit. on pp. 10, 218).

[112] Wouter Lueks, Maarten H. Everts, and Jaap-Henk Hoepman. "Vote to Link: Recovering from Misbehaving Anonymous Users". In: WPES 2016. ACM, 2016, pp. 111–122. DOI: 10.1145/2994620.2994634 (cit. on pp. 11, 50, 217).

[113] Wouter Lueks and Ian Goldberg. "Sublinear Scaling for Multi-Client Private Information Retrieval". In: FC 2015. LNCS vol. 8975. Springer, 2015, pp. 168–186. DOI: 10.1007/978-3-662-47854-7_10 (cit. on pp. 12, 218).

[114] Wouter Lueks, Jaap-Henk Hoepman, and Klaus Kursawe. "Forward-Secure Distributed Encryption". In: PETS 2014. LNCS vol. 8555. Springer, 2014, pp. 123–142. DOI: 10.1007/978-3-319-08506-7_7 (cit. on pp. 11, 32, 50, 51, 218).

[116] Mark Manulis. "Democratic Group Signatures: on an Example of Joint Ventures". In: ASIACCS 2006. ACM, 2006, p. 365. DOI: 10.1145/1128817.1128882 (cit. on p. 93).

[117] Mark Manulis, Nils Fleischhacker, Felix Günther, Franziskus Kiefer, and Bertram Poettrering. *Group Signatures: Authentication with Privacy*. Tech. rep. Bundesamt für Sicherheit in der Informationstechnik (BSI), 2012 (cit. on pp. 93, 145, 148).

[118] Alex Marthews and Catherine E. Tucker. *Government Surveillance and Internet Search Behavior*. 2015. DOI: 10.2139/ssrn.2412564 (cit. on p. 2).

[119] Silvio Micali. "Fair Public-Key Cryptosystems". In: CRYPTO '92. LNCS vol. 740. Springer, 1992, pp. 113–138. DOI: 10.1007/3-540-48071-4_9 (cit. on p. 6).

[120] Huib Modderkolk. "'Dreiging is in jaren nog niet zo groot geweest'". In: *Volkskrant* (Sept. 2016) (cit. on p. 5).

[123] Peter L. Montgomery. "Speeding the Pollard and Elliptic Curve Methods of Factorization". In: *Mathematics of Computation* 48.177 (1987), pp. 243–264. ISSN: 0025-5718. DOI: 10.1090/S0025-5718-1987-0866113-7 (cit. on p. 161).

[124] Toru Nakanishi and Nobuo Funabiki. "Verifier-Local Revocation Group Signature Schemes with Backward Unlinkability from Bilinear Maps". In: ASIACRYPT 2005. LNCS vol. 3788. Springer, 2005, pp. 533–548. DOI: 10.1007/11593447_29 (cit. on pp. 63, 72, 73, 75, 77, 93, 145, 164, 165).

[125] Moni Naor and Omer Reingold. "Number-theoretic Constructions of Efficient Pseudo-random Functions". In: *Journal of the ACM* 51.2 (Mar. 2004), pp. 231–262. ISSN: 0004-5411. DOI: 10.1145/972639.972643 (cit. on p. 114).

[126] National Institute of Standards and Technology. "Escrowed Encryption Standard". In: Federal Information Processing Standard (FIPS) vol. 185. Feb. 1994 (cit. on p. 47).

[127] Ingo Naumann and Giles Hogben. "Privacy features of European eID card specifications". In: *Network Security* 2008.8 (Aug. 2008), pp. 9–13. ISSN: 1353-4858. DOI: 10.1016/S1353-4858(08)70097-7 (cit. on p. 135).

[128] Lan Nguyen and Christian Paquin. *U-Prove Designated-Verifier Accumulator Revocation Extension*. Tech. rep. MSR-TR-2014-85. Microsoft Research, June 2014 (cit. on pp. 164, 165).

[129] OECD. "National Strategies and Policies for Digital Identity Management in OECD Countries". In: *OECD Digital Economy Papers* 177 (2011). ISSN: 2071-6826. DOI: 10.1787/5kgdzvn5rfs2-en (cit. on p. 135).

[130] Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. "Efficient Hash-Chain Based RFID Privacy Protection Scheme". In: UBICOMP 2004, WORKSHOP PRIVACY. Sept. 2004 (cit. on pp. 115, 120).

[131] Paul Ohm. "Broken Promises of Privacy: Responding to the Surprising Failure of Anonymization". In: *UCLA Law Review* 57.6 (2008). ISSN: 0041-5650. URL: http://www.uclalawreview.org/broken-promises-of-privacy-responding-to-the-surprising-failure-of-anonymization-2/ (cit. on p. 33).

[132] Femi G. Olumofin and Ian Goldberg. "Privacy-Preserving Queries over Relational Databases". In: PETS 2010. LNCS vol. 6205. Springer, 2010, pp. 75–92. DOI: 10.1007/978-3-642-14527-8_5 (cit. on pp. 167, 181).

[133] Femi G. Olumofin and Ian Goldberg. "Revisiting the Computational Practicality of Private Information Retrieval". In: FC 2011. LNCS vol. 7035. Springer, 2012, pp. 158–172. DOI: 10.1007/978-3-642-27576-0_13 (cit. on p. 168).

[134] Femi G. Olumofin, Piotr K. Tysowski, Ian Goldberg, and Urs Hengartner. "Achieving Efficient Query Privacy for Location Based Services". In: PETS 2010. LNCS vol. 6205. Springer, 2010, pp. 93–110. DOI: 10.1007/978-3-642-14527-8_6 (cit. on p. 182).

[135] European Parliament and European Council. "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)". In: *Official Journal of the European Union* 119 (Apr. 2016), pp. 1–88. URL: http://data.europa.eu/eli/reg/2016/679/oj (cit. on p. 3).

[136] Torben P. Pedersen. "A Threshold Cryptosystem without a Trusted Party (Extended Abstract)". In: EUROCRYPT '91. 1991, pp. 522–526. DOI: 10.1007/3-540-46416-6_47 (cit. on pp. 21, 22).

[137] Jonathon W. Penney. "Chilling Effects: Online Surveillance and Wikipedia Use". In: *Berkeley Technology Law Journal* 31.1 (2016), pp. 117–182. ISSN: 1086-3818. DOI: 10.15779/Z38SS13 (cit. on p. 2).

[138] Andreas Pfitzmann and Marit Hansen. *A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. v0.34. Aug. 2010. URL: http://dud.inf.tu-dresden.de/literatur/Anon%5C_Terminology%5C_v0.34.pdf (cit. on p. 34).

[139] Antonio de la Piedra, Jaap-Henk Hoepman, and Pim Vullers. "Towards a Full-Featured Implementation of Attribute Based Credentials on Smart Cards". In: CANS 2014. LNCS vol. 8813. Springer, 2014, pp. 270–289. DOI: 10.1007/978-3-319-12280-9_18 (cit. on p. 160).

[140] Julian Sanchez. *The Trouble With "Balance" Metaphors*. Accessed: June 29, 2017. 2011. URL: http://www.juliansanchez.com/2011/02/04/the-trouble-with-balance-metaphors/ (cit. on p. 5).

[141] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. RFC 6960 (Proposed Standard). Internet Engineering Task Force, June 2013 (cit. on p. 164).

[142] Bruce Schneier. "Protecting Privacy and Liberty". In: *CryptoGram* (Sept. 2001). URL: https://www.schneier.com/cryptogram/archives/2001/0930.html#8 (cit. on p. 5).

[143] Bruce Schneier. "What Our Top Spy Doesn't Get: Security and Privacy Aren't Opposites". In: *Wired* (Jan. 2008). URL: http://www.wired.com/politics/security/commentary/securitymatters/2008/01/securitymatters_0124?currentPage=all (cit. on p. 5).

[144] Edward J. Schwartz, David Brumley, and Jonathan M. McCune. "Contractual Anonymity". In: NDSS 2010. The Internet Society, 2010. URL: https://www.internetsociety.org/doc/contractual-anonymity (cit. on p. 37).

[145] Andrei Serjantov and George Danezis. "Towards an Information Theoretic Metric for Anonymity". In: PETS 2002. LNCS vol. 2482. Springer, 2002, pp. 41–53. DOI: 10.1007/3-540-36467-6_4 (cit. on p. 34).

[146] Adi Shamir. "How to Share a Secret". In: *Communications of the ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176 (cit. on pp. 20, 169).

[147] Victor Shoup and Rosario Gennaro. "Securing Threshold Cryptosystems against Chosen Ciphertext Attack". In: EUROCRYPT '98. LNCS vol. 1403. Springer, 1998, pp. 1–16. DOI: 10.1007/BFb0054113 (cit. on pp. 64, 66, 94).

[148] Joseph H. Silverman and John T. Tate. *Rational Points on Elliptic Curves*. Undergraduate Texts in Mathematics. Springer, 2015. ISBN: 978-3-319-18587-3. DOI: 10.1007/978-3-319-18588-0 (cit. on p. 102).

[149] Radu Sion and Bogdan Carbunar. "On the Practicality of Private Information Retrieval". In: NDSS 2007. The Internet Society, 2007. URL: https://www.internetsociety.org/doc/practicality-private-information-retrieval (cit. on p. 168).

[150] Nigel P. Smart. *Cryptography Made Simple*. Information Security and Cryptography. Springer, 2016. ISBN: 978-3-319-21935-6. DOI: 10.1007/978-3-319-21936-3 (cit. on pp. 26, 54).

[151] Daniel J. Solove. "A Taxonomy of Privacy". In: *University of Pennsylvania Law Review* 154.3 (Jan. 2006), pp. 477–560 (cit. on pp. 2, 3).

[152] Daniel J. Solove. "'I've Got Nothing to Hide' and Other Misunderstandings of Privacy". In: *San Diego Law Review* 44 (Nov. 2007), pp. 745–772 (cit. on p. 2).

[153] Sound Intelligence. *Sigard, aggression detection*. Accessed: January 14, 2017. URL: http://www.soundintel.com/uploads/pdf/UK/Sound%20Intelligence%20Brochure%20%28EN%29.pdf (cit. on pp. 50, 51).

[154] Speed Check Services. *SPECS3 Network average speed check solutions*. Accessed: January 14, 2017. URL: http://www.speedcheck.co.uk/images/SCS_SPECS3_Brochure.pdf (cit. on p. 97).

[155] M. Stadler. "Cryptographic Protocols for Revocable Privacy". PhD thesis. Zürich: Swiss Federal Institute of Technology, 1996 (cit. on pp. 6, 37).

[156] Volker Strassen. "Gaussian elimination is not optimal". English. In: *Numerische Mathematik* 13.4 (Aug. 1969), pp. 354–356. ISSN: 0029-599X. DOI: 10.1007/BF02165411 (cit. on p. 174).

[157] Stuart G. Stubblebine, Paul F. Syverson, and David M. Goldschlag. "Unlinkable serial transactions: protocols and applications". In: *ACM Transactions on Information System Security* 2.4 (1999), pp. 354–389. ISSN: 1094-9224. DOI: 10.1145/330382.330384 (cit. on p. 166).

[158] Fred Teeven. *Beleidsvisie ANPR*. Dutch. Feb. 2013. URL: https://www.rijksoverheid.nl/documenten/kamerstukken/2013/02/13/beleidsvisie-anpr (cit. on p. 41).

[159] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. "BLAC: Revoking Repeatedly Misbehaving Anonymous Users without Relying on TTPs". In: *ACM Transactions on Information Systems Security* 13.4 (2010), 39:1–39:33. ISSN: 1094-9224. DOI: 10.1145/1880022.1880033 (cit. on pp. 32, 50, 53, 58, 93, 164, 165).

[160] Patrick P. Tsang, Apu Kapadia, Cory Cornelius, and Sean W. Smith. "Nymble: Blocking Misbehaving Users in Anonymizing Networks". In: *IEEE Transactions on Dependable and Secure Computing* 8.2 (2011), pp. 256–269. ISSN: 1545-5971. DOI: 10.1109/TDSC.2009.38 (cit. on pp. 32, 93).

[161] Eric R. Verheul. *Practical backward unlinkable revocation in FIDO, German e-ID, Idemix and U-Prove*. Cryptology ePrint Archive, Report 2016/217. 2016. URL: http://eprint.iacr.org/2016/217 (cit. on p. 152).

[162] Pim Vullers and Gergely Alpár. "Efficient Selective Disclosure on Smart Cards Using Idemix". In: IDMAN 2013. IFIP AICT vol. 396. Springer, 2013, pp. 53–67. DOI: 10.1007/978-3-642-37282-7_5 (cit. on pp. 9, 135, 136, 160).

[163] Lawrence Wright. "The Spymaster". In: *The New Yorker* (Jan. 2008). URL: http://www.newyorker.com/magazine/2008/01/21/the-spymaster (cit. on p. 5).

[164] Ennan Zhai, David Isaac Wolinsky, Ruichuan Chen, Ewa Syta, Chao Teng, and Bryan Ford. "AnonRep: Towards Tracking-Resistant Anonymous Reputation". In: NSDI 2016. USENIX, Mar. 2016, pp. 583–596. ISBN: 978-1-931971-29-4. URL: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/zhai (cit. on p. 93).

[165] Dong Zheng, Xiangxue Li, Changshe Ma, Kefei Chen, and Jianhua Li. *Democratic Group Signatures with Threshold Traceability*. Cryptology ePrint Archive, Report 2008/112. 2008. URL: http://eprint.iacr.org/2008/112 (cit. on p. 93).

NOTATION AND SYMBOLS

This chapter provides a brief overview of the frequently used notation and symbols. In a few situations we deviate from the common notation to prevent symbol collisions.

NOTATION

Sets		
$a \in_R A$	choose a uniformly at random from the set A	Section 2.1
$[n]$	the set $\{1, \ldots, n\}$	Section 2.1
$\lvert A \rvert$	the cardinality of A	Section 2.1
Strings		
$\{0,1\}^*$	the set of all binary strings	Section 2.1
$\{0,1\}^\ell$	the set of all strings binary strings of length ℓ	Section 2.1
$\lvert x \rvert$	the length in bits of a string x	Section 2.1
$x \parallel y$	the concatenation of the strings x and y	Section 2.1
Matrices and vectors		
$\mathbf{D}, \mathbf{Q}, \mathbf{R}, \ldots$	matrices (set in bold)	Section 2.1
$\mathbf{0}$	all-zero matrix	Section 7.2
$\mathbf{I}$	identity matrix	Section 7.2
e_i	the ith basis vector	Section 7.1
$\oplus$	component wise XOR on bits, strings, vectors and matrices	Section 2.1
Algorithms and adversaries		
$\mathcal{A}, \mathcal{B}, \ldots$	algorithms and adversaries	Section 2.1
$x \leftarrow \mathcal{A}(y)$	random variable output x of $\mathcal{A}$ when run on y	Section 2.1
Logarithms		
$\lg a$	the base-2 logarithm of a	Section 2.1
$\ln a$	the natural logarithm of a	Section 2.1

GENERAL SYMBOLS

c	ciphertext	
$\hat{e}$	bilinear map from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$	Section 2.2.2
f	polynomial, usually a secret-sharing polynomial	
g	generator of $\mathbb{G}$ or $\mathbb{G}_1$	Section 2.2
g_T	generator of $\mathbb{G}_T$	Section 2.2.2
h	generator of $\mathbb{G}_2$	Section 2.2.2
i, j, i_j	indices	
k	threshold	Section 2.5
ℓ	security parameter	Section 2.4
m	message	
n	number of parties	
p	the order of the cyclic group	Section 2.2
r_A	additive share for subset A	Section 2.5.2
s_i	Shamir secret share	Section 2.5
$C(a_1, a_2)$	anonymous credential over the attributes a_1 and a_2	Section 2.7
$H, H', \ldots$	Cryptographic hash functions	
ϵ	epoch	
$\lambda_i^{\mathcal{I}}$	Lagrange coefficient for an index i in a coalition $\mathcal{I}$	Section 2.5
$\lambda_i^{\mathcal{I}}(X)$	Lagrange polynomial for an index i in a coalition $\mathcal{I}$	Section 2.5
π	a (signature) proof of knowledge	
$\mathbb{F}$	field	
$\mathbb{G}$	generic cyclic group, usually of prime order p	
$\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$	cyclic groups in bilinear setting of prime order p	
$\mathbb{Z}_p$	the integers modulo p, usually a field	Section 2.1
$\mathbb{Z}_p^*$	the units of $\mathbb{Z}_p$	Section 2.1
$\top$	denotes success	Section 2.1
$\bot$	denotes failure or error	Section 2.1

SYMBOLS USED IN CHAPTER 4

c_1, c_2	Elgamal ciphertext components	Scheme 4.7
$c, \hat{c}, \tilde{c}$	ciphertext pairs of shuffled and encrypted decryption shares	Scheme 4.16

d	response of ZK-proof in TDH2 ciphertext	Scheme 4.1
d_i	response of ZK-proof in TDH2 decryption share	Scheme 4.1
e	challenge of ZK-proof in TDH2 ciphertext	Scheme 4.1
e_i	challenge of ZK-proof in TDH2 decryption share	Scheme 4.1
$\bar{g}$	TDH2 extra generator	Scheme 4.1
g_ϵ	generator for epoch ϵ	Scheme 4.8
r	linking token	Scheme 4.8
t	transaction record	Scheme 4.8
t_1, t_2	linking information	Scheme 4.8
u	part of the TDH2 ciphertext	Scheme 4.1
u_i	part of the TDH2 decryption share	Scheme 4.1
v	part of the TDH2 ciphertext	Scheme 4.1
w	TDH2 and moderators public key	Scheme 4.1
w_i	TDH2 public key corresponding to κ_i	Scheme 4.1
y	ElGamal secret key	Scheme 4.7
y_i	private ElGamal key of anonymous moderator i	Scheme 4.16
x	user secret key	Scheme 4.8
z	zero-sharing polynomial	Scheme 4.11
H	hash function from $\{0,1\}^*$ to $\mathbb{G}_1$	Scheme 4.8
H'	hash function from $\{0,1\}^*$ to $\mathbb{Z}_p$	Scheme 4.1
H''	hash function from $\{0,1\}^*$ to $\{0,1\}^\ell$	Scheme 4.11
L	TDH2 label	Scheme 4.1
Y	ElGamal public key	Scheme 4.7
Y_i	public ElGamal key of anonymous moderator i	Scheme 4.16
Y'	random user's public key	Scheme 4.16
$\hat{Y}_i$	shuffled and randomized moderator public key	Scheme 4.16
T	encrypted linking token	Scheme 4.8
VK	TDH2 verification key	Scheme 4.1
α_i	a public key randomizer used by the SP	Scheme 4.16
δ_i	TDH2 decryption key of party i	Scheme 4.1
δ_i	voting key of moderator i	Scheme 4.8
ζ	secret for zero-sharing polynomial	Scheme 4.11
θ	permutation key	Scheme 4.11

κ	TDH2 private key	Scheme 4.1
κ_i	TDH2 secret share of private key	Scheme 4.1
π	transaction proof	Scheme 4.8
π_a, π_b, π_c	user shuffle proofs	Scheme 4.16
σ	permutation of moderator indices	Scheme 4.11
σ_{SP}, σ_U	SP and user permutations of moderator keys	Scheme 4.16
τ	transaction	Scheme 4.8
ψ	TDH2 ciphertext	Scheme 4.1
ψ_i	TDH2 decrypted ciphertext share	Scheme 4.1

SYMBOLS USED IN CHAPTER 5

c_{im}	component in vector C_i for message m	Scheme 5.25
e	HGDE secret key	Scheme 5.10
f	the cubic function defining the elliptic curve	Scheme 5.3
h	the cofactor of the group on the elliptic curve	Scheme 5.3
h_1	hash function from $\{0,1\}^{\ell_h}$ to $\{0,1\}^{\ell_h}$	Section 5.4.1
h_1	hash function from $\{0,1\}^{\ell_h}$ to $\mathbb{F}$	Section 5.4.1
ℓ_m	number of bits used for plaintext	Scheme 5.3
ℓ_h	number of bits of hash in EZS scheme	Section 5.4.1
ℓ_H	desired number of bits used for hash function	Scheme 5.3
ℓ'_H	actual number of bits used for hash function	Scheme 5.3
ℓ_c	number of bits used for counter	Scheme 5.3
r	number of corrupted senders	Section 5.3.5
$\bar{r}_{\epsilon,A}$	base additive share for set A in epoch ϵ	Section 5.4.1
s	number of epochs	Syntax 5.5
$s_{\epsilon,i}$	secret share of sender i in epoch ϵ	Scheme 5.21
$s_{m,i}$	ideal secret share for message m and sender i	Section 5.3.5
x	x-coordinate on the elliptic curve	Scheme 5.3
y	y-coordinate on the elliptic curve	Scheme 5.3
$z_{\epsilon,i}$	zero-share of sender i in epoch ϵ	Section 5.4.1
$z_{m,i}$	ideal zero-share for message m and sender i	Section 5.5.3
C_i	vector of ciphertext shares of sender i	Syntax 5.23
E	elliptic curve	Scheme 5.3

E	HGDE public key	Scheme 5.10
E'	subset of curve into which RIM maps	Scheme 5.3
H_1	hash function from $\{0,1\}^{\ell_m+\ell_c}$ to $\{0,1\}^{\ell'_H}$	Scheme 5.3
H_2	hash function from $\{0,1\}^{\ell'_H}$ to $\{0,1\}^{\ell_m+\ell_c}$	Scheme 5.3
K	HGDE derived symmetric key	Scheme 5.10
M	set of messages, subset of $\mathcal{M}$	Syntax 5.23
$S_{\epsilon,i}$	encryption key of sender i in epoch ϵ	Syntax 5.5
$Z_{\epsilon,i}$	zero-sharing key of sender i in epoch ϵ	Syntax 5.16
α_i	part of DE ciphertext share from sender i	Scheme 5.12
γ_i	HGDE ciphertext share component of sender i	Scheme 5.10
κ	index in hybrid argument	Section 5.3.5
ρ_i	HGDE ciphertext share component of sender i	Scheme 5.10
σ_i	HGDE ciphertext share component of sender i	Scheme 5.10
ψ	redundant injective map	Definition 5.1
ψ^{-1}	inverse redundant injective map	Definition 5.1
$\mathcal{M}$	message space	Syntax 5.5

SYMBOLS USED IN CHAPTER 6

$g_{\epsilon,V}$	generator for epoch ϵ and verifier V	Section 6.1
m	number of generators per epoch	Section 6.5
r	revocation value	Section 6.1
r_i	ith revocation value	Section 6.1
t_s, t_e	start and end time of an epoch	Section 6.7.2
t^*	estimated current time	Section 6.7.2
H_i	hash function i of Bloom filter	Section 6.7.5
MRL	master revocation list containing revocation values	Section 6.3
P	false positive probability of Bloom filter	Section 6.7.5
R	revocation token	Section 6.1
$RL_{\epsilon,V}$	revocation list for epoch ϵ and verifier V	Section 6.1
RV	list of credential identifiers and revocationvalue tuples	Section 6.3
κ	size of the Bloom filter	Section 6.7.5

λ	number of hash functions in Bloom filter	Section 6.7.5
ν	length of the revocation list	Section 6.7
$\mathcal{T}_C$	used generators for credential C	Section 6.3

SYMBOLS USED IN CHAPTER 7

ℓ	size of split in subcube batch code	Section 7.1.2
	subset batch code parameter	Section 7.1.2
	depth of three representing ν certificates	Section 7.3
m	number of subdatabases in batch code	Section 7.1.2
n	number of PIR servers	Section 7.1
q	number of queries	Section 7.1
r	number of rows in the database	Section 7.1
r'	subset batch code parameter	Section 7.1.2
s	number of field elements per row in the database	Section 7.1
t	subcube batch code recursion depth	Section 7.1.2
v_i	the ith PIR query	Section 7.1
w	subset batch code parameter	Section 7.1.2
$\mathbf{D}$	the PIR database	Section 7.1
$\mathrm{GF}(2^8)$	Galois field of 256 elements	
$\mathbf{K}_i$	batch code subdatabase i	Section 7.2
$\mathbf{M}$	matrix representation of batch code	Section 7.2
N	total number of blocks in all subdatabases in batch code	Section 7.1.2
$\mathbf{Q}$	matrix of queries	Section 7.1
α	subset batch code parameter	Section 7.1.2
ν	total number of certificates	Section 7.3

GLOSSARY

ABC	attribute-based credential
AE	authenticated encryption
ANPR	automatic number plate recognition
BKDE	batched key-evolving distributed encryption
BLAC	blacklistable anonymous credentials
CA	certificate authority
CCA	chosen ciphertext attack
CPIR	computational PIR
CRL	certificate revocation list
DAA	direct anonymous attestation
DBDH	decisional bilinear Diffie-Hellman
DDH	decisional Diffie-Hellman
DE	distributed encryption
DL	discrete logarithm
ECC	elliptic curve cryptography
GDPR	general data protection regulation
HGDE	Hoepman and Galindo's DE scheme
IBE	identity-based encryption
IP	internet protocol
IT-PIR	information-theoretic PIR
KDE	key-evolving distributed encryption
KLPD	koninklijke landelijke politie dienst (Dutch)
MPC	multi-party computation
OCSP	online certificate status protocol
PIR	private information retrieval
PPT	probabilistic polynomial time
SCT	signed certificate timestamp
SDH	strong Diffie-Hellman
TTP	trusted third party
VLR	verifier-local revocation
VSS	verifiable secret-sharing

INDEX

ABOUT THE AUTHOR

Wouter Lueks was born in Emmen, the Netherlands, on December 28, 1986. He attended secondary education at the Esdal College in Emmen, where he graduated in 2005. He then started a double-degree program in Mathematics and Computing Science at the University of Groningen, the Netherlands. In 2008, he received his bachelor's degrees, while in 2011 he received his master's degrees from the same university. All four were awarded the distinction cum laude.

From 2011 to 2016, he was a PhD student at the University of Nijmegen, the Netherlands, where he worked on this thesis under the supervision of Jaap-Henk Hoepman and Bart Jacobs. During this time, he also worked for one and a half years on the IRMA project, which aims at making attribute-based credentials practical. In the spring of 2014, he made a research visit to the University of Waterloo, Canada to work with Ian Goldberg on private information retrieval.

In February 2017, he started working as a post-doctoral researcher at the IMDEA Software Institute in Madrid. There, he collaborates with Carmela Troncoso on electronic voting systems and privacy-enhancing technologies.

LIST OF PUBLICATIONS

The following is a list of academic publications that Wouter authored or co-authored (in reverse-chronological order):

1. Wouter Lueks, Gergely Alpár, Jaap-Henk Hoepman, and Pim Vullers. "Fast revocation of attribute-based credentials for both users and verifiers". In: *Computers & Security* 67 (2017), pp. 308–323. ISSN: 0167-4048. DOI: 10.1016/j.cose.2016.11.018.

2. Wouter Lueks, Maarten H. Everts, and Jaap-Henk Hoepman. "Vote to Link: Recovering from Misbehaving Anonymous Users". In: WPES 2016. ACM, 2016, pp. 111–122. DOI: 10.1145/2994620.2994634.

3. Wouter Lueks, Gergely Alpár, Jaap-Henk Hoepman, and Pim Vullers. "Fast Revocation of Attribute-Based Credentials for Both Users and Verifiers". In: IFIP SEC 2015. IFIP AICT vol. 455. Springer, 2015, pp. 463–478. DOI: 10.1007/978-3-319-18467-8_31.

4. Wouter Lueks, Maarten H. Everts, and Jaap-Henk Hoepman. "Revocable Privacy: Principles, Use Cases, and Technologies". In: APF 2015. LNCS vol. 9484. Springer, 2016, pp. 124–143. DOI: 10.1007/978-3-319-31456-3_7.

5. Jaap-Henk Hoepman, Wouter Lueks, and Sietse Ringers. "On Linkability and Malleability in Self-blindable Credentials". In: WISTP 2015. LNCS vol. 9311. Springer, 2015, pp. 203–218. DOI: 10.1007/978-3-319-24018-3_13.

6. Wouter Lueks and Ian Goldberg. "Sublinear Scaling for Multi-Client Private Information Retrieval". In: FC 2015. LNCS vol. 8975. Springer, 2015, pp. 168–186. DOI: 10.1007/978-3-662-47854-7_10.

7. Jaap-Henk Hoepman, Bert-Jaap Koops, and Wouter Lueks. "Anoniem misdaad melden via Internet: technische en juridische risico's". Dutch. In: *Nederlands Juristenblad* 43 (Dec. 2014), pp. 3056–3063.

8. Wouter Lueks, Jaap-Henk Hoepman, and Klaus Kursawe. "Forward-Secure Distributed Encryption". In: PETS 2014. LNCS vol. 8555. Springer, 2014, pp. 123–142. DOI: 10.1007/978-3-319-08506-7_7.

9. Bassam Mokbel, Wouter Lueks, Andrej Gisbrecht, and Barbara Hammer. "Visualizing the quality of dimensionality reduction". In: *Neurocomputing* 112 (2013), pp. 109–123. DOI: 10.1016/j.neucom.2012.11.046.

10. Gergely Alpár, Lejla Batina, and Wouter Lueks. "Designated Attribute-Based Proofs for RFID Applications". In: RFIDSEC 2012. LNCS vol. 7739. Springer, 2013, pp. 59–75. DOI: 10.1007/978-3-642-36140-1_5.

11. Bassam Mokbel, Wouter Lueks, Andrej Gisbrecht, Michael Biehl, and Barbara Hammer. "Visualizing the quality of dimensionality reduction". In: ESANN 2012. 2012. URL: https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2012-99.pdf.

12. Andrej Gisbrecht, Wouter Lueks, Bassam Mokbel, and Barbara Hammer. "Out-of-sample kernel extensions for nonparametric dimensionality reduction". In: ESANN 2012. 2012. URL: https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2012-25.pdf.

13. Wouter Lueks, Ivan Viola, Matthew van der Zwan, Henk Bekker, and Tobias Isenberg. "Spatially Continuous Change of Abstraction in Molecular Visualization". In: BIOVIS 2011. Extended abstract and poster. 2011.

14. Matthew van der Zwan, Wouter Lueks, Henk Bekker, and Tobias Isenberg. "Illustrative Molecular Visualization with Continuous Abstraction". In: *Computer Graphics Forum* 30.3 (2011), pp. 683–690. DOI: 10.1111/j.1467-8659.2011.01917.x.

TITLES IN THE IPA DISSERTATION SERIES SINCE 2014

J. van den Bos. *Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics*. Faculty of Science, UvA. 2014-01

D. Hadziosmanovic. *The Process Matters: Cyber Security in Industrial Control Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-02

A.J.P. Jeckmans. *Cryptographically-Enhanced Privacy for Recommender Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-03

C.-P. Bezemer. *Performance Optimization of Multi-Tenant Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2014-04

T.M. Ngo. *Qualitative and Quantitative Information Flow Analysis for Multi-threaded Programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-05

A.W. Laarman. *Scalable Multi-Core Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-06

J. Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. Faculty of Science, Mathematics and Computer Science, RU. 2014-07

W. Meulemans. *Similarity Measures and Algorithms for Cartographic Schematization*. Faculty of Mathematics and Computer Science, TU/e. 2014-08

A.F.E. Belinfante. *JTorX: Exploring Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-09

A.P. van der Meer. *Domain Specific Languages and their Type Systems*. Faculty of Mathematics and Computer Science, TU/e. 2014-10

B.N. Vasilescu. *Social Aspects of Collaboration in Online Software Communities*. Faculty of Mathematics and Computer Science, TU/e. 2014-11

F.D. Aarts. *Tomte: Bridging the Gap between Active Learning and Real-World Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2014-12

N. Noroozi. *Improving Input-Output Conformance Testing Theories*. Faculty of Mathematics and Computer Science, TU/e. 2014-13

M. Helvensteijn. *Abstract Delta Modeling: Software Product Lines and Beyond*. Faculty of Mathematics and Natural Sciences, UL. 2014-14

P. Vullers. *Efficient Implementations of Attribute-based Credentials on Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2014-15

F.W. Takes. *Algorithms for Analyzing and Mining Real-World Graphs*. Faculty of Mathematics and Natural Sciences, UL. 2014-16

M.P. Schraagen. *Aspects of Record Linkage*. Faculty of Mathematics and Natural Sciences, UL. 2014-17

G. Alpár. *Attribute-Based Identity Management: Bridging the Cryptographic Design of ABCs with the Real World*. Faculty of Science, Mathematics and Computer Science, RU. 2015-01

A.J. van der Ploeg. *Efficient Abstractions for Visualization and Interaction*. Faculty of Science, UvA. 2015-02

R.J.M. Theunissen. *Supervisory Control in Health Care Systems*. Faculty of Mechanical Engineering, TU/e. 2015-03

T.V. Bui. *A Software Architecture for Body Area Sensor Networks: Flexibility and Trustworthiness*. Faculty of Mathematics and Computer Science, TU/e. 2015-04

A. Guzzi. *Supporting Developers' Teamwork from within the IDE*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-05

T. Espinha. *Web Service Growing Pains: Understanding Services and Their Clients*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-06

S. Dietzel. *Resilient In-network Aggregation for Vehicular Networks*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-07

E. Costante. *Privacy throughout the Data Cycle*. Faculty of Mathematics and Computer Science, TU/e. 2015-08

S. Cranen. *Getting the point — Obtaining and understanding fixpoints in model checking*. Faculty of Mathematics and Computer Science, TU/e. 2015-09

R. Verdult. *The (in)security of proprietary cryptography*. Faculty of Science, Mathematics and Computer Science, RU. 2015-10

J.E.J. de Ruiter. *Lessons learned in the analysis of the EMV and TLS security protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2015-11

Y. Dajsuren. *On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems*. Faculty of Mathematics and Computer Science, TU/e. 2015-12

J. Bransen. *On the Incremental Evaluation of Higher-Order At-*

tribute Grammars. Faculty of Science, UU. 2015-13

S. Picek. *Applications of Evolutionary Computation to Cryptology*. Faculty of Science, Mathematics and Computer Science, RU. 2015-14

C. Chen. *Automated Fault Localization for Service-Oriented Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-15

S. te Brinke. *Developing Energy-Aware Software*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-16

R.W.J. Kersten. *Software Analysis Methods for Resource-Sensitive Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2015-17

J.C. Rot. *Enhanced coinduction*. Faculty of Mathematics and Natural Sciences, UL. 2015-18

M. Stolikj. *Building Blocks for the Internet of Things*. Faculty of Mathematics and Computer Science, TU/e. 2015-19

D. Gebler. *Robust SOS Specifications of Probabilistic Processes*. Faculty of Sciences, Department of Computer Science, VUA. 2015-20

M. Zaharieva-Stojanovski. *Closer to Reliable Software: Verifying functional behaviour of concurrent programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-21

R.J. Krebbers. *The C standard formalized in Coq*. Faculty of Science, Mathematics and Computer Science, RU. 2015-22

R. van Vliet. *DNA Expressions – A Formal Notation for DNA*. Faculty of Mathematics and Natural Sciences, UL. 2015-23

S.-S.T.Q. Jongmans. *Automata-Theoretic Protocol Programming*. Faculty of Mathematics and Natural Sciences, UL. 2016-01

S.J.C. Joosten. *Verification of Interconnects*. Faculty of Mathematics and Computer Science, TU/e. 2016-02

M.W. Gazda. *Fixpoint Logic, Games, and Relations of Consequence*. Faculty of Mathematics and Computer Science, TU/e. 2016-03

S. Keshishzadeh. *Formal Analysis and Verification of Embedded Systems for Healthcare*. Faculty of Mathematics and Computer Science, TU/e. 2016-04

P.M. Heck. *Quality of Just-in-Time Requirements: Just-Enough and Just-in-Time*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2016-05

Y. Luo. *From Conceptual Models to Safety Assurance – Applying Model-Based Techniques to Support Safety Assurance*. Faculty of Mathematics and Computer Science, TU/e. 2016-06

B. Ege. *Physical Security Analysis of Embedded Devices*. Faculty of

Science, Mathematics and Computer Science, RU. 2016-07

A.I. van Goethem. *Algorithms for Curved Schematization*. Faculty of Mathematics and Computer Science, TU/e. 2016-08

T. van Dijk. *Sylvan: Multi-core Decision Diagrams*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2016-09

I. David. *Run-time resource management for component-based systems*. Faculty of Mathematics and Computer Science, TU/e. 2016-10

A.C. van Hulst. *Control Synthesis using Modal Logic and Partial Bisimilarity – A Treatise Supported by Computer Verified Proofs*. Faculty of Mechanical Engineering, TU/e. 2016-11

A. Zawedde. *Modeling the Dynamics of Requirements Process Improvement*. Faculty of Mathematics and Computer Science, TU/e. 2016-12

F.M.J. van den Broek. *Mobile Communication Security*. Faculty of Science, Mathematics and Computer Science, RU. 2016-13

J.N. van Rijn. *Massively Collaborative Machine Learning*. Faculty of Mathematics and Natural Sciences, UL. 2016-14

M.J. Steindorfer. *Efficient Immutable Collections*. Faculty of Science, UvA. 2017-01

W. Ahmad. *Green Computing: Efficient Energy Management of Multiprocessor Streaming Applications via Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-02

D. Guck. *Reliable Systems – Fault tree analysis via Markov reward automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-03

H.L. Salunkhe. *Modeling and Buffer Analysis of Real-time Streaming Radio Applications Scheduled on Heterogeneous Multiprocessors*. Faculty of Mathematics and Computer Science, TU/e. 2017-04

A. Krasnova. *Smart invaders of private matters: Privacy of communication on the Internet and in the Internet of Things (IoT)*. Faculty of Science, Mathematics and Computer Science, RU. 2017-05

A.D. Mehrabi. *Data Structures for Analyzing Geometric Data*. Faculty of Mathematics and Computer Science, TU/e. 2017-06

D. Landman. *Reverse Engineering Source Code: Empirical Studies of Limitations and Opportunities*. Faculty of Science, UvA. 2017-07

W. Lueks. *Security and Privacy via Cryptography – Having your cake and eating it too*. Faculty of Science, Mathematics and Computer Science, RU. 2017-08