Incentive Compatible and Provably Secure Blockchain Applications

by

Osman Biçer

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in

Computer Science and Engineering

KOÇ ÜNİVERSİTESİ

September 5, 2022

Incentive Compatible and Provably Secure Blockchain Applications

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a doctoral dissertation by

Osman Biçer

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assoc. Prof. Alptekin Küpçü (Advisor)

_____

Prof. Engin Erzin

_____

Assoc. Prof. Özgür Yılmaz

_____

Asst. Prof. Murat Ak

_____

Prof. Ali Aydın Selçuk

Date: _____

To my family

ABSTRACT

Incentive Compatible and Provably Secure Blockchain Applications
Osman Biçer
Doctor of Philosophy in Computer Science and Engineering
September 5, 2022

With the introduction and advent of blockchain technologies, money transfers have become easier with lower costs, no matter how far the recipient is from the sender. Moreover, the blockchain technology has not only affected the contemporary economical model, but also inspired many novel applications that have been thought impossible without the help of a trusted third party (such as governments or well-recognized companies). However, being a newly developed technology, blockchain still lacks enough research to ensure its security (in terms of mining, transactions, or its applications) and incentive compatibility (i.e., the participants are motivated for following the expected protocol steps). Cryptography and game theory are two well-known tools with traditional proving methods to help us for this task. In this thesis, we utilize game theory and cryptography based security models for building secure applications on blockchain. First, we develop a provably secure blockchain application that we name as e-donation, for secure, fair and decentralized online donations. Second, we develop a practical attribute based digital signature scheme. Third, we analyze and improve the security of blockchain by providing an incentive compatible defense solution against the selfish mining attack of Eyal and Sirer (CACM '18) on proof-of-work blockchain. Forth, we develop a network-based simulation tool for both attacks and defenses. Fifth, we propose a framework for better game-theoretical analysis of multi-player mechanisms when the sizes of the coalitions can be bounded by a threshold, via successfully combining threshold security ideas of cryptography and transferable utility of game theory. Sixth, we apply our threshold coalition notions to show the incentive compatibility of our solution to the outsourced incentivized computation problem in blockchain with multiple outsourced parties.

ÖZETÇE

Teşvik Uyumlu ve İspatlanabilir Güvenli Blokzincir Uygulamaları
Osman Biçer
Bilgisayar Bilimi ve Mühendisliği, Doktora
5 Eylül 2022

Blokzincir teknolojilerinin hayatımıza girmesi ve gelişmesi para gönderimini gönderici ile gönderen arasındaki mesafeden bağımsız olarak maliyetini düşürerek kolaylaştırmıştır. Dahası, blokzincir teknolojisi sadece günümüz ekonomik modelini etkilememiştir, aynı zamanda daha önce güvenilir bir otorite (devlet veya bilinen bir firma gibi) olmadan imkansız olduğu düşünülen birçok yeni uygulamaya da ilham kaynağı olmuştur. Fakat yeni bir teknoloji olan blokzincir; madencilik, hesap işlemleri ve uygulamaları açısından güvenliğini ve teşvik uyumluluğunu (yani katılımcıların beklenen protokol adımlarını takip etmesinin motive edilmesi) garanti edecek kadar araştırılmamıştır. Kriptografi ve oyun teorisi bize bu mevzuda yardımcı olabilecek ispat yöntemleri olan geleneksel iki araçtır. Bu tezde güvenli blokzincir uygulamaları için oyun teorisi ve kriptografik güvenlik modellerini kullanmaktayız. Birinci olarak, e-bağış adını verdiğimiz merkezi olmayan ve adil çevrimiçi bağışlar için ispatlanabilir güvenli bir blokzincir uygulamasını geliştirmekteyiz. İkinci olarak, pratik ve çok yönlü özellik tabanlı bir dijital imza protokolü önermekteyiz. Üçüncü olarak, çalışma-ispatı blokzincirine Eyal ve Sirer'in (CACM '18) bencil madencilik saldırısına karşı teşvik uyumlu bir savunma çözümü sunarak blokzincirin güvenliğini analiz etmekte ve geliştirmekteyiz. Dördüncü olarak, bu ve benzeri saldırıların ve karşı savunmalarının simüle edilebileceği ağ tabanlı bir simülasyon aracını da beraberinde tasarlamaktayız. Beşinci olarak, eşik kriptografisini ve oyun teorisinin devredilebilir yarar kavramını başarıyla birleştiren bir sistem sunmaktayız. Bu sistem sayesinde koalisyon boyutu bir eşikle sınırlandırılabildiğinde çok oyunculu mekanizmaların daha iyi oyun teorik analizi mümkün olmuştur. Altıncı olarak, bu sistemi teşvik edilmiş başkasına hesaplatma problemine çözümümüzde uygulamakta, teşvik uyumluluğunu göstermekteyiz.

ACKNOWLEDGMENTS

To start with, I would like to greatly thank my supervisor Assoc. Prof. Alptekin Küpçü from Koç University for all of his helps and support. He is indeed an inspiring researcher with lots of creativity, disciplined hardworking style, and lots of knowledge and experience. Throughout my PhD study, all of my works have been in collaboration with him, and we have discussed every single detail and possible ways forward. Without his input, I would not have achieved the quality of this thesis. Also, he is a great mentor and has always engaged with the issues that I faced along the way.

Also, I would like to thank my thesis progress and defense committee members Prof. Engin Erzin from Koç University and Assoc. Prof. Özgür Yılmaz from Koç University. They have always been interested in my work and have provided highly valuable opinions and comments. Thanks to them, I have improved myself along the PhD study. I would also thank my committee members Asst. Prof. Murat Ak from Akdeniz University and Prof. Ali Aydın Selçuk from TOBB University of Economics and Technology for interesting discussions, instructive comments, and support.

I would also thank our Cryptography, Security, and Privacy Group interns Abdullah Talayhan from Bilkent University and Levent Çelik from Koç University for their roles in the implementation of Versatile ABS, Prashanthi Ramachandran from Ashoka University and Nandini Agrawal from Ashoka University for their roles in the development of BlockSim-Net and the collaboration in the related published work, Shiwam Agarwal from Ashoka University for his role in simulation of `Fortis`, Burcu Yıldız from Koç University for the collaboration in the published work for m-stability, Onur Eren Arpacı from Koç University and Deniz

Gürarslan from İzmir Institute of Technology for their roles in the implementation of `Delegate`.

Further, I would like to thank Assoc. Prof. Mehmet Sabır Kiraz from De Montfort University, Asst. Prof. Emre Gürsoy from Koç University, Dr. Yahya Hassanzadeh-Nazarabadi from Dapper Labs (previously at Cryptography, Security, and Privacy Group of Koç University), Dr. Sanaz Taheri-Boshrooyeh from Status.im (previously at Cryptography, Security, and Privacy Group of Koç University), and Devriş İşler from IMDEA Networks (previously at Cryptography, Security, and Privacy Group of Koç University) for interesting discussions related to various security topics including the blockchain related ones.

We acknowledge support from TÜBİTAK, the Scientific and Technological Research Council of Turkey, under project number 119E088.

The last but not the least, I would like to greatly thank my parents Yıldız Biçer and Dr. Mesut Biçer for their support and love along the way. They were extremely helpful and patient. I wish my beloved Mom could see the day of my thesis defense. I would also thank Cansu Coşkun and her mother Saniye Coşkun for their support and help.

TABLE OF CONTENTS

List of Tables **xii**

List of Figures **xiv**

Chapter 1: Introduction **1**

1.1 Literature Review 2

1.1.1 Blockchain and Cryptocurrencies 2

1.1.2 Mechanism Design 5

1.1.3 Outsourced Computation Methods 6

1.2 Contributions of the Thesis 7

Chapter 2: Versatile ABS **10**

2.1 Introduction 10

2.1.1 Our Contributions 14

2.1.2 System Model 15

2.1.3 Overview of Our Techniques 17

2.2 Preliminaries 19

2.3 Definition of Versatile ABS 24

2.3.1 Operations 24

2.3.2 Security Definitions 28

2.4 Our Modular VABS 33

2.5 Security Proof of Our VABS 43

2.6 Efficiency 52

2.7 Related Work 56

2.8 Discussion 58

Chapter 3: e-Donation **60**
3.1 Introduction 60
3.2 Preliminaries 68
3.3 E-Donation Framework 75
3.3.1 Operations 76
3.3.2 Security Definitions 78
3.4 Our e-Donation Scheme 87
3.5 Related Work 89
3.6 Efficiency 93
3.7 Conclusion and Future Work 94
3.8 Security Proof of Our Scheme 95
3.8.1 Proof of Unforgeability and Balance 95
3.8.2 Proof of Fair Distribution of Donations 99
3.8.3 Proof of Donation and Recipient Privacy 101

Chapter 4: BlockSim-Net **105**
4.1 Introduction 105
4.1.1 Our contributions 106
4.1.2 Related Work 108
4.2 Preliminaries 110
4.3 Simulation 112
4.3.1 The Admin Server (AS) 112
4.3.2 The Miner 114
4.3.3 The Overview of the Simulation 114
4.3.4 Initialization 115
4.3.5 Updating local blockchain 117
4.3.6 Switching to another miner's blockchain 120
4.3.7 Consensus 120
4.4 Results 121

4.5 Conclusion 124

Chapter 5: `Fortis` **126**

5.1 Introduction 126
5.2 Related Work 130
5.3 Preliminaries 133
5.4 Oracle and Bold Mining Attacks 136
5.5 Generic Formulas For SM Attacks 141
5.6 Analyses of Oracle and Bold Mining 144
5.6.1 Oracle Mining Attack Optimization 144
5.6.2 Bold Mining Attack Optimization 146
5.7 Our SM Mitigation Algorithm 150
5.8 Simulation 153

Chapter 6: ***m*-Stability** **158**

6.1 Introduction 158
6.2 Related Work 160
6.3 Background on Game Theory and Mechanism Design 160
6.4 Shortcomings of Existing Definitions 164
6.5 Our Security Definitions Against Coalitions 166
6.5.1 Security Definitions Against a Single Coalition 166
6.5.2 Extension Against Multiple Coalitions 167
6.5.3 Comparison to Previous Definitions 168
6.6 Example Mechanisms from Literature 170
6.6.1 Example 1: Incentivized Outsourced Computation 170
6.6.2 Example 2: Forwarding Dilemma 175
6.6.3 Example 3: Strong Connectivity 178

Chapter 7: `Delegate` **181**

7.1 Introduction 181

7.1.1 Overview of Our Techniques 183
7.1.2 Our Contributions 185
7.2 Related Work 186
7.3 Preliminaries and Subprotocols 187
7.3.1 Outsourced Computation Primitives 187
7.3.2 The Universal Composability (UC) Framework 188
7.3.3 Non-Interactive and Reusable-CRS Commitment Schemes . 192
7.4 Our Response Submission Protocol 194
7.5 Our Incentivized Outsourced Computation Proposal 198
7.5.1 System Model 198
7.5.2 Our `Delegate` Protocol 200
7.6 Game Theoretical Analysis of `Delegate` 202
7.6.1 Strategies of Contractors Outside Coalition 202
7.6.2 Analysis of Leaking and *MismatchCheck* Actions 206
7.6.3 Analysis Without Coalitions 207
7.6.4 Completion of Analysis Without Coalitions 209
7.6.5 Analysis With Non-Collaborating Coalitions 210
7.6.6 Analysis With Collaborating Coalitions 211
7.7 Comparison to State-of-The-Art 213

Chapter 8: Conclusion **214**

Bibliography **217**

LIST OF TABLES

2.1 Comparison of VABS with other MA-ABS schemes. 14

2.2 Comparison of VABS with the anonymous credential scheme of [Camenisch et al., 2006]. 16

2.3 The computation times and bandwidth uses of GlobalSetup, TraceSetup, IdPJoin, AuthJoin, UserJoin, TraceIssue, and AttrIssue of our VABS scheme. 53

3.1 Comparison of traditional international charity organizations, the non-anonymous cryptocurrency schemes, the anonymous cryptocurrency schemes, the private smart contract schemes and our e-donation scheme 104

4.1 A qualitative comparison between state-of-the-art simulators. . . . 108

4.2 Block percentage for major Bitcoin miners in a BlockSim-Net simulation. 122

4.3 Block percentage for major Litecoin miners in a BlockSim-Net simulation. 124

4.4 Comparison of the performance of BlockSim and BlockSim-Net. . . 124

5.1 Table of abbreviations and notations used throughout Chapter 5 . . 134

6.1 The utilities (u_A, u_B) of Alice and Bob from honest strategies s_A and s_B, and deviant strategies s'_A and s'_B. 164

6.2 The utilities (u_A, u_B) of Alice and Bob from honest strategies s_A and s_B, and deviant strategies s'_A and s'_B. 165

6.3 The expected utility of each contractor from choosing diligent or lazy with respect to the other players' chosen strategies (where $0 < k < n$) in the outsourced computation mechanism of [Küpçü, 2017]. 170
6.4 The utility of each player from choosing forward or drop with respect to the other players' chosen strategies in the forwarding dilemma of [Naserian and Tepe, 2009]. 175

7.1 Utility matrix of a diligent contractor based on leaking. 206
7.2 Utility matrix of a contractor if no coalition exists, only honest submission and complying is allowed. 207
7.3 Detailed comparison of our work with the state-of-the-art in terms of our goals listed in Section 7.5.1. 212

LIST OF FIGURES

2.1 The UserJoin protocol. 36
2.2 The TraceIssue protocol. 37
2.3 The AttrIssue protocol. 39
2.4 The implementation results of Sign and Verify of our VABS. 54
2.5 The computation times and signature size comparison of our VABS without tracing and revocation and [Ramani et al., 2019] 55

3.1 The flow of the online donation scheme. 62
3.2 The flow of transactions in our e-donation scheme. 64
3.3 A high level description of the $\mathsf{Privacy}^{u,n,\delta}_{\mathcal{A},\Pi}(\lambda)$ game. 83
3.4 Computation time and total transaction size estimates for PrivateDonate and PrivateClaim transactions. 92

4.1 BlockSim-Net Simulation Flow. 115

5.1 An example flow of Oracle mining attack. 137
5.2 An example flow of Bold mining attack. 141
5.3 Revenue from selfish mining [Eyal and Sirer, 2018] with respect to hashing power α for network powers $\gamma = 0$, $\gamma = 0.5$, and $\gamma = 1$ calculated using our equaiton and the original equation of [Eyal and Sirer, 2018]. 145
5.4 Revenue from Oracle mining on Decentralized FP w.r.t. the hashing power α when the timestamps set optimally. Also the optimum t_m/T_c values for all α of interest are provided. 147

5.5 Revenue from Bold mining on Decentralized (or even Authority) FP vs. revenue from honest mining with respect to α. 150
5.6 The relative revenues of the optimal selfish mining of [Sapirshtein et al., 2016], our Bold mining, and our Oracle mining on our `Fortis` algorithm. 154
5.7 Simulation results of relative revenues of the Oracle and Bold miners when the honest mining algorithm is Freshness Preferred and our `Fortis` algorithm. 155

6.1 Graphs related to an instance of the strong connectivity game of [Eidenbenz et al., 2003]. 176

Chapter 1

INTRODUCTION

Blockchain may be considered among the most important inventions of the 21st century. It has not only a huge prospect to change the current economical dynamics by removal of central authorities on money or making money transfers (almost) instantaneous and cheap, but also have a variety of fascinating applications that have never been considered as possible before. However, being a newly developed technology, blockchain still lacks and requires lots of research to ensure its proven security (in terms of e.g., mining, transactions, or its applications).

Blockchain security is a crucial issue, that affects its applicability to various areas. In this online, multi-party and non-authenticated system, possible vulnerabilities are so diverse that unfortunately, it is not trivial to develop a single security framework. This contrasts to what we do in conventional cryptographic protocols where well-defined tools such as security games and functionalities. Yet, as we will clarify later, it turns out that such strong security guarantees as provided by cryptography is not necessary in many cases.

Usually, attackers are rational and clever personalities that conducts to attack for benefit. Cryptograhic assumptions tend to reflect them far more dangerous than what we have in practice. In essence, we do not object what we achieve by provable security by stating that they are useless. We just claim that if this is not achievable or efficient enough to be practical, it is possible to disincentivize rational attackers by making economics of the attack contrary to their benefits. This approach is logical and applied already by Nakamoto in his proposal of Bitcoin [Nakamoto, 2008]. We take a further step by using the conventional game theory and mechanism design for this purpose.

In what follows, we provide our starting literature review on blockchain and related security and privacy issues, mechanism design, outsourced computation (i.e., an area that we apply blockchain for security), anonymous authentication methods (for enhencing security of blockchain applications while preserving privacy).

1.1 Literature Review

1.1.1 Blockchain and Cryptocurrencies

Bitcoin [Nakamoto, 2008] has been the first decentralized crypto currency that adopts blockchain technology. Thanks to the inspiration by Bitcoin [Nakamoto, 2008], many cryptocurrencies (e.g., Ethereum [Buterin, 2013], Litecoin [Litecoin, 2011]) based on blockchain have beed proposed and gone alive. There exists various consensus methods that are used for blockchain security (e.g., proof of work [Dwork and Naor, 1993], proof of stake [King and Nadal, 2012], proof of validation [Hassanzadeh-Nazarabadi et al., 2021], or Byzantine fault tolerance [Lamport et al., 1982]). Moreover, there exists other decentralization techniques in cryptocurrencies, such as, IOTA [Popov, 2017] using "The Tangle" where the data is kept as a directed acyclic graph, Phantom [Sompolinsky and Zohar, 2018] and Spectre [Sompolinsky et al., 2017] where the data is kept as a directed acyclic graph of blocks "blockDAG".

Blockchain. A blockchain is essentially an ever-growing chain of data blocks [Nakamoto, 2008, Narayanan et al., 2016]. Each block consists of a set of recent transactions, its index from the first block, the hash of the previous block (its parent), and a nonce. The hash of each block is required to be lower than a publicly determined value called *difficulty*. The blockchain miners keep competing to become the first one to mine the next block.[1] Whenever a miner mines a new block,

[1]To mine a block means to find the nonce included in the block such that the hash of the block maps below the difficulty. The difficulty is frequently adjusted by the miners so that the expected inter-block time of the main chain would be constant.

she publishes it to the network. If the block is included in the main chain, she receives some block reward. An honest miner always mines on the head (the last block) of the longest branch (i.e., the chain that has the most blocks that is privy to her). If there exists a tie among the longest branches, various tie-breaking mechanisms can be involved[2]. The blocks that do not end up in the main chain are not rewarded and are called as "orphan blocks". We highlight that each miner invests some computational and communication resources for mining coins. The aim for fairness in a PoW blockchain is that the revenue of each miner is proportional to her investment [Garay et al., 2015]. If this is achieved, miners are better off with honest mining strategy rather than harmful ones to the system and the proper operation of the blockchain is secured.

Smart Contracts. A smart contract (SC) is utilized for establishing binding contracts in cryptocurrency environments. SC is publicly executed by the underlying consensus protocol. It collects the submitted transactions to the network and append them to the blockchain. We assume SC as trusted third party (TTP) just as in other smart contract based protocols [Kothapalli et al., 2017, Afanasev et al., 2018, Avizheh et al., 2019, Küpçü and Safavi-Naini, 2021]. We note that in Ethereum any Turing machine can be run as SC as long as the generator pays the price per instruction as "gas" [Atzei et al., 2017]. However, in Bitcoin, the programming language for SC generation is not Turing complete, and not all computer programs can be run [Atzei et al., 2018]. We refer to [Atzei et al., 2017, Atzei et al., 2018, Zheng et al., 2020] for further investigation of smart contracts. As the aim of this paper is not the privacy of the computation steps or result (but only its correctness), for generality and practicality we have not restricted ourselves with private smart contract schemes [Kosba et al., 2016, Kalodner et al., 2018, Bünz et al., 2019, Bowe et al., 2020, Eni, 2019, Kerber et al., 2020]. Yet, privacy of the outsourced incentivized computation is a quite interesting research subject to be developed.

[2]the Bitcoin application in April 2021 suggests that each miner mines on the chain that she received the first

Transaction Privacy. Bitcoin-like crypto currencies depend only on pseudonyms for user privacy and do not prevent linkability between transactions. To resolve this problem, two solutions (Zerocoin [Miers et al., 2013] and Zerocash [Sasson et al., 2014]) have been proposed to apply the idea that first a party blinds her coins, and then utilize zero-knowledge proofs or arguments for spending previously minted blinded coins without direct referencing. Another well-known method is "mixing" based schemes (e.g., CryptoNote [v. Saberhagen, 2013], Mixcoin [Bonneau et al., 2014], CoinShuffle [Ruffing et al., 2014], CoinParty [Ziegeldorf et al., 2015]). Nevertheless, a recent analysis [Möser et al., 2018] on Monero[3] (i.e., a CryptoNote based cryptocurrency) shows that these type of cryptocurrencies are vulnerable to "chain-reaction analysis". To completely avoid these type of attacks, the anonymized transactions in these protocols should use all of the previously blinded coins during mixing, which decreases their efficiency due to complicated zero-knowledge proofs.

Bitcoin Mining Security. There exists some interesting works taking into consideration the security characteristics of Bitcoin. Eyal and Sirer proposed the "selfish mining" attack [Eyal and Sirer, 2018], and showed that in the best case having 1/3 of computational power is enough for a rational miner to benefit from this attack. [Kiayias et al., 2016a] proposed and analyzed two ideal mining games, immediate release and strategic release games, both of which have two players, one honest and one rational. They conclude that if the hashing power of the rational player is fewer than about 30%, she is better off by playing the honest strategy. [Garay et al., 2015] proves the "persistence" and the "liveness" properties of Bitcoin protocol under the assumptions of honest majority, a strict number of miners, the same hashing power for each miner, and high network synchronicity. Persistence property proves that the probability of an honest player's blockchain does include an valid transaction is a negligible function of k if this transaction has a position of at least k blocks deep in the blockchain of one or more honest players. Liveness property states that an honest player's blockchain will eventually

[3] `https://www.getmonero.org`

include all valid transactions at a depth more than k blocks if they are repeatedly broadcast to the network enough. The authors extended their analysis in their later work [Garay et al., 2017] to a more realistic scenario where the number of miners and their hashing powers are not strict and the cryptographic puzzle for PoW is adjustable. An exiting work [Badertscher et al., 2018a] has showed that although there has been found many attacks why blockchain still works without any major problem in reality by using rational protocol design method which combines game theory and crpytography.

Blockchain Ledger Applications. Since blockchain based technologies also act as public ledger, many blockchain applications other than conventional cryptocurrency have been proposed. These applications include incentivization of fair and or secure mutli-party computation [Kumaresan and Bentov, 2014, Bentov and Kumaresan, 2014, Kumaresan et al., 2016, Andrychowicz et al., 2016, Kiayias et al., 2016c], online voting [Roby, 2017], e-auction [Blass and Kerschbaum, 2017], distributed PKI [Axon and Goldsmith, 2017], etc.

1.1.2 Mechanism Design

The use of mechanism design in distributed systems and multi-party computation is an extensively studied subject with some notable works including [Gordon and Katz, 2006, Lysyanskaya and Triandopoulos, 2006, Abraham et al., 2006, Belenkiy et al., 2008, Fuchsbauer et al., 2010, Dani et al., 2011, Abraham et al., 2013, Upadhyay, J., 2015, Küpçü, 2017]. There are some proposed models such as the Byzantine (B), altruistic (A), rational (R) or BAR model [Aiyer et al., 2005, Li et al., 2006], type model [Feldman et al., 2004], and the rational protocol design (RPD) framework [Garay et al., 2013]. In this work, for simplicity, we model the parties as BAR model (without altruistic players) rather than the type model. We do not utilize RPD framework, as we are interested in internal deviations of the participants, instead of an outlier adversary taking control of the protocol participants. Another reason for this is simplicity, as we show the cryptographic subprotocol is universally composable [Canetti, 2001]. For coalition proofness,

conventionally, a line of works [Wang et al., 2012, Abraham et al., 2013, Brenguier, 2016, Halpern and Vilaça, 2016] confronts to satisfy k-resiliency [Abraham et al., 2006]. There also exists models as in [Li et al., 2006, Heller, 2008] based on simpler coalitional abilities such as "cheap talk" [Farrell and Rabin, 1996].

1.1.3 Outsourced Computation Methods

The most important requirement for outsourced computation is the correctness of the computation results. This can be checked via zero-knowledge proofs, e.g., [Goldwasser et al., 1985, Goldreich et al., 1991, Bellare and Goldreich, 1993], where the computers prove their work to the boss. In fact, here the main problem is not privacy but correctness, therefore "verifiable computation" methods can also be applied [Gennaro et al., 2010]. For a nice literature review on verifiable computation, we refer to [Walfish and Blumberg, 2015]. We can group other outsourced computation techniques as ones based on attribute based encryption [Parno et al., 2012], trusted hardware [Sadeghi et al., 2010], and interactive proofs [Goldwasser et al., 2015, Thaler et al., 2012]. Byzantine agreement or Byzantine fault tolerence based distributed computation methods [Bracha and Toueg, 1985, Aiyer et al., 2005, Li et al., 2006] can also be employed, however they are slow for generic use and they need some assumptions on the number of minimum honest computers (e.g., at least 1/3 of them needs to be honest).

The methods [Golle and Mironov, 2001, Sarmenta, 2002, Szajda et al., 2003, Du et al., 2004, Szajda et al., 2005] for verification of outsourced computation are also interesting. Inner state hash which is proposed and utilized by [Belenkiy et al., 2008] can also be included in this category. In this method computers also hands the inner state hashed to the boss in addition to the result of computation. The boss then checks all the hashes coming from all computers to detect any malicious behavior. This method is indeed very low cost and much faster than verifiable computation.

We also need to check the real-world example outsourced computation systems. Among the well-known ones, we can list SETI@Home, Rosetta@Home,

BOINC, World Community Grid, and Distributed.net projects. Unfortunately, none of these projects guarantee the correctness of the computations [Molnar, 2000]. Further, they only reward those computers that do their job correctly, but not punish the other ones. [Belenkiy et al., 2008] observes that the outsourced computation systems that do not provide punishment are problematic even for the rational computers.

In this area, the-state-of-the-art work [Küpçü, 2017] provides a solution for the payment of rewards and punishments based on fair barter. In recent years, new methods [Andrychowicz et al., 2014, Bentov and Kumaresan, 2014, Kumaresan and Bentov, 2014, Kumaresan et al., 2016, Kiayias et al., 2016c] based on blockchain are proposed for incentivization of multi-party computation. However, none of these are applicable for outsourced computation.

Use of Blockchain in Fair and Secure Computations. Blockchain has been utilized for incentivization of fair and or secure multi-party computation [Kumaresan and Bentov, 2014, Bentov and Kumaresan, 2014, Kumaresan et al., 2016, Andrychowicz et al., 2016, Kiayias et al., 2016c]. The works that have close resemblance to ours by outsourcing computation to multiple parties are [Avizheh et al., 2019, Küpçü and Safavi-Naini, 2021].

1.2 Contributions of the Thesis

In this thesis, we aim to provide a concrete methodology to resolve blockchain security issues, some of which have been mentioned above. Namely, we use cryptography and game theory as tools for achieving *incentive compatible and provably secure blockchain applications*.

In some cases, the related problems can be resolved by the direct application of conventional provable security methods of cryptography. This is exemplified by the problem of online donations (see Chapter 3). When it is practical, this approach provides security against adversaries that are capable of attacking systems for any reason via any possible way. We achieve more dependable systems with great security guarantees, as long as the adversarial computing power is bounded

by a polynomial and the underlying mathematical problems are not broken.

In other cases, we incentivize participants to follow the protocol specifications. This is exemplified by the problems of selfish mining mitigation (see Chapter 5) and incentivized outsourced computation (see Chapter 7). Incentivization involves following the protocol specification more profitable compared to a deviating strategy as in both applications, and may be combination of game theory and cryptography as in the case of the latter. This approach provides more practical and efficient solutions to well-known problems in blockchain security. Although this may come at the cost of sacrificing security compared to conventional cryptography-alone protocols (as we no longer have security against any attacks but against incetivized ones), our approach provides solutions to more diverse problems.

More concretely, our contributions in this thesis can be summarized as follows:

1. We propose a usage limited, revocable, threshold Traceable, decentralized attribute based signature scheme, useful particularly in blockchain applications (see Chapter 2),

2. We propose a novel blockchain application *e-donation*, for secure, fair and decentralized online donations on blockchain (see Chapter 3),

3. We develop a network-based simulator for selfish mining type of attacks and defenses against them (see Chapter 4),

4. We provide a defence mechanism against the selfish mining attack of [Eyal and Sirer, 2018], with self-issued timestamps (i.e. without an authority) in presence of a global synchronous clocks (see Chapter 5),

5. We provide a novel cooperative game theory and threshold security based framework for countering coalitions in online mechanisms (see Chapter 6),

6. We use our framework for analyzing our solution to outsourced incentivized computation problem in multi-party setting with blockchain and smart contract setup (see Chapter 7).

We highlight that all of our contributions are either published or submitted as separate works. Our 1-st contribution is in submission to ACM TOPS as a joint work with Alptekin Küpçü with title "Versatile ABS: Usage Limited, Revocable, Threshold Traceable, Decentralized Attribute Based Signatures" [Biçer and Küpçü, 2019]. Our 2-nd contribution is published in PETS/PoPETS '20 as a joint work with Alptekin Küpçü with title "Anonymous, Attribute Based, Decentralized, Secure, and Fair e-Donation" [Biçer and Küpçü, 2020a]. Our 3-rd contribution is published in Turkish Journal of Electrical Engineering and Computer Sciences as a joint work with Prashanthi Ramachandran, Nandini Agrawal, and Alptekin Küpçü with title "BlockSim-Net: a network-based blockchain simulator" [Ramachandran et al., 2022]. Our 4-th contribution is in submission to ACM DLT as a joint work with Alptekin Küpçü with title "FORTIS: Selfish Mining Mitigation by (FOR)geable (TI)me(S)tamps" [Biçer and Küpçü, 2020b]. Our 5-th contribution is published in ACM CCSW as a joint work with Alptekin Küpçü with title "m-Stability: Threshold Security Meets Transferable Utility" [Biçer et al., 2021]. Our 6-th contribution is submitted to EUROCRYPT as a joint work with Alptekin Küpçü with title "Delegate: Coalition, Sybil, and Copy Proof Incentivized Outsourced Computation with Smart Contracts" [Biçer and Küpçü, 2022].

Chapter 2

VERSATILE ABS: USAGE LIMITED, REVOCABLE, THRESHOLD TRACEABLE, DECENTRALIZED ATTRIBUTE BASED SIGNATURES

2.1 Introduction

Let us consider the following problem. Due to the COVID-19 outbreak, to keep as many people as possible at home, suppose that a government wants to grant free food from markets to the citizens that have monthly income less than \$1500 and are above the age 60 (i.e., those with an attribute policy "income $<$ \$1500 AND age $>$ 60). The person that benefits from this aid may prefer to remain anonymous for privacy reasons by hiding her identity during the online order. Naturally, the government may want to limit the number of each individual's orders in a given month, as its resources are limited; so the solution should allow periodic usage limitation. Also, the government may opt to revoke her grants, in case she does not deserve them anymore (e.g., when her income changes), requiring the solution to be revocable. Furthermore, if she makes an illegal order, the government would want to detect her identity for punishment, requiring traceability. This is one of the possible scenarios where a specific type of attribute-based signature scheme could be useful.

Attribute based signatures (ABS) [Maji et al., 2011, Cao et al., 2012, Okamoto and Takashima, 2013, El Kaafarani et al., 2014b, El Kaafarani et al., 2014a, Hampiholi et al., 2015, Ghadafi, 2015, Urquidi et al., 2016, Sakai et al., 2016, Sakai et al., 2018, Ramani et al., 2019, Huang et al., 2021] are developed for non-interactively proving satisfiability of a Boolean attribute policy by signing a message using some attribute tokens. For this proof, the user should have previously obtained

these tokens from attribute authorities. The goal is to disclose only suitability to the given attribute policy while protecting the privacy of the rest (e.g., identity of the signer, her other attributes, her other generated signatures). An ABS should also protect against collusion of users combining their attributes, and defend against attempts to forge signatures without having the required combination of the attribute tokens in the policy. An early ABS proposal [Maji et al., 2011] shows that ABS has many application areas including attribute based messaging [Bobba et al., 2006, Bobba et al., 2010], trust negotiation [Frikken et al., 2006], and cloud access control [Li et al., 2010]. Also, ABS may be suitable for blockchain applications, such as the e-donation system [Biçer and Küpçü, 2020a] and Monero[1]. To compare ABS with similar constructions (e.g., mesh signatures [Boyen, 2007] and anonymous credentials [Camenisch and Lysyanskaya, 2001]), we refer to [Maji et al., 2011].

Multiple Authorities. Due to the lack of support for multiple attribute authorities in the initially developed ABS schemes, a multi-authority ABS (MA-ABS) scheme was proposed [Cao et al., 2012]. However, this scheme required a central master authority, so the decentralized ABS scheme of [Okamoto and Takashima, 2013] was developed. Later, [El Kaafarani et al., 2014b] proposed the first decentralized traceable ABS scheme that allows a tracing authority to detect the identity of the signer of an ABS. They also claim that their scheme allows "proving" that identity to a judge. Unfortunately, [El Kaafarani et al., 2014b] allows each attribute authority that a signer has interacted with to obtain the user's secret token, and thus forge a signature traceable to her. [Ghadafi, 2015] elaborates on this issue, and provides "non-frameability" and "tracing soundness" notions (the latter has been previously defined in the context of group signatures by [Sakai et al.,

[1]Web site: https://www.getmonero.org/. We note that Monero currently uses ring signatures [Rivest et al., 2006, Wang et al., 2011, Liu and Wong, 2005] for anonymity to be self-sufficient without any authority. However, for high level of anonymity, these signatures should mix the public keys [Möser et al., 2018] of many account owners. This makes their usage inefficient compared to group signatures and ABS. There seems to be a trade-off between privacy and centralization.

2012]), as well as a generic construction satisfying these notions. However, these proposals still allow any malicious tracing authority to disclose the identity of the signer of any message (including honest signers).

In this chapter, we confront some of the shortcomings of existing MA-ABS schemes, and achieve the simultaneous existence of the following properties:

- ***Decentralization.*** The removal of the requirement of a single master authority. An MA-ABS scheme is not decentralized if a master authority is required for secure operation.

- ***Periodic usage limitation.*** The ability to limit the number of verifying signatures of a user (per verifier) in a given time period. This periodic usage limitation notion differs from the controllable linkability of [El Kaafarani et al., 2014a, Urquidi et al., 2016, El Kaafarani and Ghadafi, 2017], as the former merely limits the number of signatures that gets verified by a verifier to a verifier-determined bound, while the latter provides opportunity for linking between the signatures generated by the same user. For the attribute based messaging scenario, where the ABS may be used for the authentication of the sender [Maji et al., 2011], periodic usage limitation can provide spam filtering by limiting the number of messages a user can send per time period. Another use is the k-times anonymous authentication [Teranishi et al., 2004] scenario.

- ***Threshold traceability.*** The requirement that a predefined number of tracing authorities *should collaborate* to output the identity of the signer of a signature. This is important to eliminate the possibility of a corrupted tracing authority de-anonymizing a signer without a rightful purpose. This property is useful for official case resolution in practice. This notion somewhat resembles to selective traceability of [von Ahn et al., 2006], where an explicitly stated set of authorities can trace a user.

- ***Reliable traceability.*** Inclusion of the inability to forge a person's signature

(even by the authorities) and the ability of threshold-many honest tracing authorities *to always find* the original signer. Inability to forge a person's signature also supports our periodic usage limitation goal, since if a malicious party could forge a signer's signature, it would have consumed her rights to honestly generate verifying signatures. Note that this notion also covers "traceability" of [El Kaafarani et al., 2014b] and "non-frameability" of [Ghadafi, 2015].

- ***Dynamic revocation of attributes.*** A useful property for dynamically revoking the attributes of signers that no longer deserve them by the attribute authorities (e.g., if a student graduates, her student attribute can be revoked).

- ***Dynamic revocation of users.*** The ability to dynamically detach detected malicious users or criminals from the ability of generating verifiable ABSs by a specialized authority. Similarly, this may be employed, when a user's key is stolen or the user is deceased.

- ***Range proofs.*** Proof of an attribute thath falls in a given range. In many cases, this proof is important for privacy of the user by not allowing leakage of the exact value. Although in narrow ranges this can be achievable via OR proofs, the size of the proof becomes inefficent as the range gets wider.

We named our scheme that successfully combines these novel, non-trivial, and seemingly conflicting aspects as Versatile ABS, or shortly VABS. Use of VABS for signing citizens' orders would solve the Covid-19 related problem at the beginning of the chapter. Also, VABS has found an application area in a novel blockchain based e-donation system [Biçer and Küpçü, 2020a] at PETS 2020, where it is utilized for receiving donations and VABS usage limitation enables the blockchain miners to limit the receivable donations in a given time period. Table 2.1 compares the features of existing MA-ABS schemes and our VABS.

MA-ABS Scheme	Decentralization	Usage Limitation	Traceability	Threshold Traceability	Reliable Traceability	Attribute Revocation	User Revocation	Range Proofs
[Cao et al., 2012]	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
*	✓	N/A	N/A	N/A	N/A	N/A	N/A	N/A
[El Kaafarani et al., 2014b]	✓	N/A	✓	N/A	N/A	N/A	N/A	N/A
[Ghadafi, 2015]	✓	N/A	✓	N/A	✓	N/A	N/A	N/A
[Sakai et al., 2016]	✓	N/A	N/A	N/A	N/A	N/A	N/A	N/A
[Drăgan et al., 2018]	✓	N/A	N/A	N/A	N/A	N/A	N/A	N/A
VABS	✓	✓	✓	✓	✓	✓	✓	✓

Table 2.1: Comparison of VABS with other MA-ABS schemes. *, ✓, and N/A denote [Okamoto and Takashima, 2013], and the existence and non-existence of a feature, respectively.

2.1.1 *Our Contributions*

The contributions of this chapter are fourfold:

1. In Section 2.3, we provide our formal definitions of the components (GlobalSetup, TraceSetup, IdPJoin,AuthJoin, UserJoin, TraceIssue, AttrIssue, Sign, Verify, UserRevoke, AttrRevoke, Trace) of a VABS scheme. We also provide our game based security definitions for anonymity, tracing reliability, signature unforgeability, and soundness. Our security definitions are novel by including the different authority architecture and allowing the proposed VABS operations; yet they are inline with previous definitions on similar paradigms.

2. In Section 2.4, we provide the first VABS construction with modular design. The desired features can be turned on/off to efficiently achieve the requirements of the planned application. We note that the novelty of our construction is that it effectively and securely combines existing anonymous credential, threshold encryption, commitment, and zero-knowledge proof techniques to achieve a provably secure VABS scheme.

3. In Section 2.5, we formally prove the security of our VABS based on Strong RSA (via reduction to the security of the CL signatures [Camenisch and

Lysyanskaya, 2003]), the security of the TPKE scheme, DDH, and SDDHI [Camenisch et al., 2006] (via reduction to the pseudo random function construction of [Dodis and Yampolskiy, 2005]) assumptions in the random oracle model.

4. In Section 2.6, we implement our VABS scheme with and without traceability, and show the efficiency (i.e., for a VABS with up to 20 attributes, Sign and Verify operations take below 0.3 and 0.2 sec, respectively, and the generated signature size is below 89 KB).

2.1.2 System Model

In our system, there are ***users***, who can be signers or verifiers, and ***authorities***. We have three type of authorities: *identity providers*, *attribute authorities*, and *tracing authorities*, as explained below.

Identity providers (e.g., civil registry authority providing national identity numbers) are responsible for issuing/revoking global identifiers and ensuring that a user picks her secret key randomly (to prevent two users' obtaining the same key for attribute collusion).

Also, there exist ϕ ***tracing authorities*** in our model, at least θ of which can together de-anonymize the signer of a given VABS. We allow the adversary to corrupt at most Δ of the tracing authorities. Obviously, we require $\theta > \Delta$ so that malicious authorities can never de-anonymize the users themselves. Also, we require $\phi \geq \theta + \Delta$, so there exists enough honest tracing authorities that can always trace and output the correct signer of a VABS in legal cases.

There are an arbitrary number of ***attribute authorities***, each responsible for issuing various attributes to the deserving parties. We only trust these authorities for proper attribute issuance and for non-disclosure of the identities of the users that have interacted with them. Yet, in our solution, even with the information resulting from those interactions, none of the identity providers or attribute authorities can de-anonymize a signer from a given signature or can sign a message

Scheme	Multi Authority	Public Keys of an Authority i	Message Signing	Collusion Resistance	Usage Limitation	Traceability	Revocation
**	convertible	$O(\|\omega\|_i)$	convertible	N/A	per attribute	N/A	attribute
VABS	✓	$O(1)$	✓	✓	per user	✓	user & attribute

Table 2.2: Comparison of VABS with the anonymous credential scheme of [Camenisch et al., 2006] denoted as **. Also, ✓ and N/A denote the existence and non-existence of a feature. $|\omega|_i$ denotes the number of attributes that are issued by the authority i. By convertible, we refer to the fact that the scheme in the presented form do not provide the feature, but there exists generic methods to obtain it.

on behalf of a user. The only malicious actions that those authorities may take are issuing tokens to the undeserving users and revealing the identity of the users that applied to them for tokens (which are concerns in all such existing schemes, and are not related to our construction).

Threat model. Regarding *anonymity*, we allow an adversary to fully corrupt all identity providers, all attribute authorities, and Δ tracing authorities, to control all users except for two, and to obtain arbitrary number of VABSs on messages from any (honest) user that it chooses. We require the VABS scheme still to disallow an adversary from distinguishing the signer of a VABS subject to the following restrictions: Both users should be out of the adversary's control, should satisfy the related attribute policy of the VABS, and should not have signed more VABSs than the usage limitation.

Regarding *traceability*, we allow the adversary to fully corrupt all identity providers, all attribute authorities, Δ tracing authorities, and all users. We require the VABS scheme still to disallow an adversary from generating a VABS that cannot be traced to its original signer (identified by its secret key s) by θ honest tracing authorities as a result of tracing operation execution.

Regarding *unforgeability*, we allow the adversary to fully corrupt all tracing authorities, all attribute authorities except for those that can issue a particular attribute, and all users except for those that have the attribute issued (still the

adversary can ask for VABSs from these users). We require the VABS scheme still to disallow an adversary from generating a VABS with an attribute policy chosen by the adversary that requires a particular attribute for satisfiability and gets verified.

Regarding *soundness* (i.e., n-times usability), we allow the adversary to fully corrupt all identity providers, all tracing authorities, all attribute authorities, and all users. We require the VABS scheme still to disallow an adversary from generating more than the limitation number of verified VABSs within a time period for a user s.

2.1.3 Overview of Our Techniques

Initial Ideas. We start with the n-times unlinkable anonymous credential (AC) scheme of [Camenisch et al., 2006], and add the VABS requirements. We highlight that [Camenisch et al., 2006] does not propose a clear way of preventing *collusion* of attributes among users. By preventing collusion, we mean that a user with a "*income<$1500*" attribute and another user with a "*age>60*" attribute should not be able to collude and conform to a "*income<$1500* AND *age>60*" policy. Also, a generic technique for "key binding" [Bichsel et al., 2014] does not exist for obtaining collusion resistance in [Camenisch et al., 2006]. In our solution, we prevent such collusion while enabling multiple authorities, as long as identity providers honestly follow the protocol. Moreover, although [Camenisch et al., 2006] provides n-times unlinkability and credential revocation, it does not provide traceability, user revocation, and range proofs which we provide here. Also, there exists no known generic method for "inspection" (corresponding to traceability) and revocation, as described in [Bichsel et al., 2014]. For the complete comparison of [Camenisch et al., 2006] and our scheme, please see Table 2.2

Our Techniques. When joining the system, the user interacts with an identity provider, and the user's secret key s is generated as the output. The user obtains a Camenisch-Lysyanskaya (CL) signature [Camenisch and Lysyanskaya, 2003] on s and a certificate including a commitment to s and the real user identity id from

the identity provider. Afterward, the user interacts with at least $\theta + \Delta$ tracing authorities to obtain her CL tracing tokens on s.

We differentiate attributes as *descriptive attributes* (i.e., the ones that the user either have or does not have, e.g., "Stanford University student") and *numeric attributes* (i.e., the ones that the user either may have with a numeric value, e.g., "age = 64") After the interaction with the tracing authorities, the user can obtain a token for a descriptive attribute ω from the qualified authority (i.e., Stanford University). In this case, the attribute token is generated as a CL signature on $s + H(\omega)$, where $H(\cdot)$ is a hash function modeled as random oracle. This prevents collusion among users for combining their attributes, and is one of our novelties over [Camenisch et al., 2006], in addition to traceability and other properties. Regarding a numeric attribute ω with a numeric value ν, the related token is obtained from the authority as a CL signature on $s + H(\omega) + \nu$.

The user can sign a message for a policy with Boolean AND/OR combinations of attributes. In many cases, a NOT operation can easily be obtained via simple conversions (e.g., " *NOT income≥$1500*", could be converted to "*income<$1500*") or can be separately obtained as an attribute from an authority. We highlight that existing decentralized schemes of [Cao et al., 2012, Okamoto and Takashima, 2013, El Kaafarani et al., 2014b] also do not have direct NOT operation support. The user signs by generating a serial number S for the signature a commitment C_s to s, a commitment C_J to J, and a tracing tag Φ. Here S is obtained from the user's secret s and the number J of times she has generated a signature in the current period. Also, Φ is the threshold encryption of g_1^s where g_1 is a group parameter. The user also appends non-interactive zero-knowledge proofs on the message for showing that S, C_s, C_J, and Φ are constructed correctly, and that $J < n$ where n is the number uses allowed in a time period, and the CL signatures for her descriptive attributes that she obtained from the authorities are valid. For numeric attributes, the user additionally proves that the numeric value is in a given range. Verification is done by checking all proofs and the existence of S in the shared database of signatures.

For tracing a user, given a valid VABS, θ honest tracing authorities can reveal g_1^s, and check the database for the user's real identity. Our technique utilizes the threshold public key encryption (TPKE) scheme of [Canetti and Goldwasser, 1999]. Note that we have chosen this TPKE scheme for efficiency, yet it may also be possible to build a similar construction via other non-broadcast TPKE schemes (e.g., [Boneh et al., 2006, Arita and Tsurudome, 2009, Libert and Yung, 2011, Wee, 2011, Lin and Sun, 2013, Gan et al., 2013]), as long as they allow efficient zero-knowledge proof of equality of an encrypted value to a committed value and efficient validation of correct construction of a ciphertext by a third party. However, the broadcast TPKE schemes [Delerablée and Pointcheval, 2008, Sakai et al., 2015] are not suitable due to the fact that they require the signer to provide the public-secret key pairs to the tracing authorities herself, which results in disclosure of her identity even by the verifier that needs to validate tracing transcripts.

2.2 Preliminaries

Notation. Throughout this paper,

- $a \twoheadleftarrow B$: the value of a is picked from the set B uniformly at random,
- $a \leftarrow \mathsf{B}$ and $B \rightarrow a$: a is set as the output of a probabilistic polynomial time (PPT) B,
- $a := b$: the value of a is set as the value of b,
- $\mathsf{A}\{B_1(b_1), B_2(b_2)\} \rightarrow (c_1), (c_2)$: A is a protocol executed between parties B_1 with input b_1 and B_2 with input b_2. At the end, party B_1 obtains output c_1 and party B_2 obtains output c_2.
- a (i.e., a with the blue color) : a is an optional step only for achieving the user/attribute revocation feature.

- a (i.e., a with the green color) : a is an optional step only for achieving the reliable traceability,
- λ denotes a security parameter,
- n denotes the number of uses allowed in a time period, and
- $\mathcal{QR}_{\mathsf{N}}$ denotes the quadratic residue set modulo N.

Zero-Knowledge Proofs. In our protocols, we utilize non-interactive zero knowledge proofs of knowledge (NIZKPoK) obtained via the Fiat-Shamir transformation [Fiat and Shamir, 1987] of zero knowledge proof of knowledge (ZKPoK) protocols. The Fiat-Shamir transformation of ZKPoK protocols can also be used to sign messages, in which case we call them signatures of knowledge (SoK). For NIZKPoKs and SoKs, we inherit the notation of Camenisch and Stadler [Camenisch and Stadler, 1997]: NIZKPoK$\{A : C\}$ denotes a NIZKPoK of the private value set A such that the statement C is satisfied, and SoK$[m]\{A : C\}$ denotes SoK of the private value set A such that the statement C is satisfied on the public message m.

For OR proofs, we make use of zero-knowledge OR proofs realizable by the generic scheme of [Cramer et al., 1994]. This scheme provides an efficient method for proving knowledge of solutions for a-out-of-d problems, without revealing the subset of the problems whose solutions are known. The other example ZKPoK schemes that can be used in the realization of our constructions are as follows:

- [Camenisch and Michels, 1999] for proving factorization of a strong RSA modulus,
- [Camenisch and Lysyanskaya, 2001] for proving that some numbers are quadratic residues of a strong RSA modulus,
- [Boudot, 2000] for proving that a committed value is in a given range,

- [Camenisch and Lysyanskaya, 2003] for proof of a CL signature, and
- [Schnorr, 1991] for proof of a discrete logarithm and opening values of Pedersen commitments [Pedersen, 1992].

Pseudo Random Functions (PRFs). In the PRF security game $\mathsf{DistPRF}$, the challenger $\mathcal{C}$ gives to the adversary $\mathcal{A}$ a unary security parameter 1^λ and oracle access to either $F_s(\cdot)$ or $f(\cdot)$ (chosen fairly at random), where f is a random function and F is a PRF family. The adversary wins by guessing whether the oracle is $F_s(\cdot)$ or $f(\cdot)$. We say F is a PRF family, if for all PPT adversaries $\mathcal{A}$, for randomly chosen s, there exists a negligible function $\mathsf{n}(\cdot)$ such that

$$\mathsf{Pr}[\mathcal{A} \text{ wins } \mathsf{DistPRF}] \leq \frac{1}{2} + \mathsf{n}(\lambda)$$

Also, we call $F_s(\cdot)$ a PRF. Note that [Camenisch et al., 2006] shows that the function $F_{g,s}(\cdot) = \mathbf{g}^{1/(s+\cdot)}$ is a PRF (where g is a random generator of a generic group $\mathbb{G}$ of prime order q and $s \twoheadleftarrow \mathbb{Z}_q^*$), if SDDHI assumption [Camenisch et al., 2006] holds in $\mathbb{G}$. We refer the reader to [Camenisch et al., 2006] for the further details.

Digital Signatures. A digital signature scheme consists of three algorithms: (1) $\mathsf{DSKeyGen}$ for public-private key pair generation, (2) DSSign for generating a signature on a transcript with the private key, (3) $\mathsf{DSVerify}$ for verifying a signature on a given transcript with the public key. The digital signatures satisfy the conventional existential unforgeability under adaptive chosen message attack (EU-ACMA) game for signatures described as $\mathsf{Sig\text{-}forge}$ in [Katz and Lindell, 2007].

CL Key Generation. The algorithms for authorities joining the system utilizes the CL key generation procedure [Camenisch and Lysyanskaya, 2003] given in Algorithm 1. The proofs of knowledge for NIZKPoK_1, and NIZKPoK_2 can be instantiated as in [Camenisch and Michels, 1999] for knowledge of strong primes and [Camenisch and Lysyanskaya, 2001] for showing that the values are quadratic residues, respectively.

AC Scheme of [Camenisch et al., 2006]. We now briefly describe the n-times unlinkable anonymous credential scheme of [Camenisch et al., 2006] that we

Algorithm 1 CL key generation algorithm $\mathsf{CLKeyGen}$.

input: a unary security parameter 1^λ.

output: a CL public key $(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathsf{D})$, a CL secret key (e, f), and the related proofs $(\text{NIZKPoK}_1, \text{NIZKPoK}_2)$.

$\mathsf{e}, \mathsf{f} \twoheadleftarrow O(1^\lambda)$ Sophie Germain primes

$\mathsf{D} := (2\mathsf{e}+1)(2\mathsf{f}+1)$; $\mathbf{a}, \mathbf{b}, \mathbf{c} \twoheadleftarrow \mathcal{QR}_{\mathsf{D}}$

$\text{NIZKPoK}_1 :=\text{NIZKPoK}\{(\mathsf{e}, \mathsf{f}) : \mathsf{O}_i = (2\mathsf{e}+1)(2\mathsf{f}+1)\}$

$\text{NIZKPoK}_2 :=\text{NIZKPoK that } (\mathbf{a}, \mathbf{b}, \mathbf{c}) \in \mathcal{QR}_{\mathsf{D}}$

return $(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathsf{De}, \mathsf{f}, \text{NIZKPoK}_1, \text{NIZKPoK}_2)$

build upon. Let $\ell_q \in \Theta(\lambda)$, ℓ_x, ℓ_{time}, and ℓ_{cnt} be system parameters satisfying $\ell_q \geq \ell_x \geq \ell_{time} + \ell_{cnt} + 2$ and $2\ell_{cnt} - 1 > n$. The attribute issuer generates a cyclic group $\langle g \rangle = \mathbb{G}$ of prime order q such that $2^{\ell_q - 1} < q < 2^{\ell_q}$. It also generates another generator h of $\mathbb{G}$ and a cyclic group $\langle \mathbf{g} \rangle = \langle \mathbf{h} \rangle = \mathbf{G}$ of composite order $\mathsf{p}'\mathsf{q}'$ where $\mathbf{g}$ and $\mathbf{h}$ are quadratic residues modulo $\mathsf{N} = (2\mathsf{p}'+1)(2\mathsf{q}'+1)$. Moreover, the issuer also generates a CL signature [Camenisch and Lysyanskaya, 2003] key pair (p, s) within the group $\mathbf{G}$. It publishes its public key $(g, h, \mathbf{g}, \mathbf{h}, \mathbf{G}, p)$, and the zero-knowledge proof that N is a special RSA modulus [Camenisch and Michels, 1999] and that $\langle \mathbf{g} \rangle = \langle \mathbf{h} \rangle$ are quadratic residues modulo N [Camenisch and Lysyanskaya, 2001]. To obtain a credential, a user first interacts with the issuer, and they run the following protocol in a mutually authenticated channel. (1) The user generates a key pair $(s_1, p_1 := g^{s_1})$. (2) The user picks $s_2' \twoheadleftarrow \mathbb{Z}_q$ and computes the Pedersen commitment $\mathsf{C}_{s_1+s_2'}$ to $s_1 + s_2'$. [Pedersen, 1992]. She then sends $\mathsf{C}_{s_1+s_2'}$ to the issuer and proves that it is correctly formed via [Schnorr, 1991]. (3) The issuer picks $\rho \twoheadleftarrow \mathbb{Z}_q$ and sends it to the user. Both parties compute $\mathsf{C}_{s_1+s_2'+\rho} := \mathsf{C}_{s_1+s_2'} g^\rho$. The user sets $s_2 := s_2' + \rho$. (4) The parties run the signature on a committed value protocol in [Camenisch and Lysyanskaya, 2003] using $\mathsf{C}_{s_1+s_2}$. At the end, the user obtains a CL signature σ of the issuer on the user secret keys s_1 and s_2.

The user can then prove the issued credential anonymously to a verifier n-times in a given time period t via the following protocol. (1) The verifier sends to the user a value $R \twoheadleftarrow \mathbb{Z}_q^*$. (2) The user computes and sends to the verifier the serial number $S = g^{1/(s_1+t2^{\ell_{cnt}}+J))}$ and the double spending tag $E = p_1 g^{R/(s_2+2^{\ell_{cnt}+\ell_{time}}+t2^{\ell_{cnt}}+J))}$ where J is the number of times that the user has authenticated her credential in the current time period $t \geq 1$. (3) The user and the verifier run ZKPoK protocols for s_1, s_2, σ, and J such that $0 \leq J < n$, $S = g^{1/(s_1+t2^{\ell_{cnt}}+J))}$, $E = p_1 g^{R/(s_2+2^{\ell_{cnt}+\ell_{time}}+t2^{\ell_{cnt}}+J))}$, and σ verifies on the user secret s with the issuer public key p.

Threshold Public Key Encryption Scheme of [Canetti and Goldwasser, 1999]. We now briefly describe the threshold public key encryption scheme of [Canetti and Goldwasser, 1999] that we utilize for threshold traceability. We note that at least θ honest authorities can decrypt a message and δ authorities can learn anything about the plaintext from the given ciphertext with negligible probability. According to [Canetti and Goldwasser, 1999], the two thresholds have the relation. $\theta = 2\Delta + 1$ The KeyGeneration, Encryption, and Decryption operations are as provided below.

KeyGeneration. [Canetti and Goldwasser, 1999] assumes a dealer for this operation for simplicity. Yet the paper suggests that it can be removed via a multi-party protocol [Chaum et al., 1988] or more efficiently via [Pedersen, 1991a]. Let the $O(\lambda)$ prime numbers p and q, the group generators g_1 and g_2 of a group $\mathbb{Z}_q$, the thresholds θ and Δ, and the number L of decryptions before the keys need to be refreshed be determined before the protocol. Also let a polynomial $P(\xi) = \sum_{i=0}^{d} \alpha_i \xi^i$ mod q be a random polynomial for α if $\alpha_0 = \alpha$ and $\alpha_1, \ldots, \alpha_d \twoheadleftarrow \mathbb{Z}_q$. The dealer generates:

- $x_1, x_2, y_1, y_2, z \twoheadleftarrow \mathbb{Z}_q$ and degree $\theta - 1$ random polynomials $P^{x_1}(), P^{x_2}(), P^{y_1}(), P^{y_2}(), P^{z}()$ for x_1, x_2, y_1, y_2, z, respectively.

- L values $s_1, \ldots, s_L \twoheadleftarrow \mathbb{Z}_q$ and degree Δ random polynomials $P^{s_1}(), \ldots, P^{s_L}()$ for them.

- L degree $\theta - 1$ random polynomials $P^{0_1}(), \ldots, P^{0_L}()$ for the value 0.

Let $x_{j,i} = P^{x_j}(i)$. Let $y_{j,i}, z_i, s_{l,i}, 0_{l,i}$ be defined similarly. The secret key of the party i is $tes_i = (p, q, g_1, g_2, x_{1,i}, x_{2,i}, y_{1,i}, y_{2,i}, z_i, s_{1,i}, \ldots, s_{L,i}, 0_{1,i}, \ldots, 0_{L,i})$. The public key is $tep = (p, q, g_1, g_2, c, d, h)$ where $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$.

Encryption. Obtained by the operation $Enc_{tep}(m, r) = (g_1^r, g_2^r, mh^r, c^r d^{r\alpha})$, where $\alpha = H(g_1^r, g_2^r, h^r m)$.

Decryption. To decrypt l-th ciphertext (u_1, u_2, e, v), each party i in the set T of at least θ parties computes:

- $v_i' := u_1^{x_{1,i} + y_{1,i}\alpha} u_2^{x_{2,i} + y_{2,i}\alpha}$,
- $f_i = u_1^{z_i} \cdot (v / v_i') \cdot g_1^{0_{l,i}}$.

Then, each party obtains the shares f_i of the parties in T to compute $f_0 = \Pi_{j \in T} f_j^{\lambda_j}$, where λ_js satisfy that for any $\theta - 1$ degree polynomial $P()$ over $\mathbb{Z}_q$ we have $P(0) = \sum_{j \in T} \lambda_j P(j)$. Then, i outputs $m = e / f_0$.

2.3 Definition of Versatile ABS

2.3.1 Operations

$\mathsf{GlobalSetup}(1^\lambda) \rightarrow params$: This is an operation that takes place once in the setup phase of the scheme. It takes as input a unary security parameter 1^λ. It outputs global setup parameters *params* (which is assumed to include 1^λ for simplicity of the presentation). Note that depending on the application, this can be run as an algorithm by a trusted party or as a multi-party protocol to avoid single point of failures.

$\mathsf{TraceSetup}(params) \rightarrow ((tp_1, ts_1), \ldots, (tp_\phi, ts_\phi))$: This is an operation that takes place once in the setup phase. It takes as input the global setup parameters *params*. It outputs the tracing key pair (tp_i, ts_i) for each tracing authority i, such that at least θ authorities are needed to trace the signer of a given VABS.

$\mathsf{IdPJoin}(params) \rightarrow (iis, iip, irs, irp_0, sk_{id}, pk_{id})$: This is an algorithm that is executed by each identity provider when it joins the system. The algorithm takes

as input the global setup parameters $params$. It outputs an identity provider issuing secret and public key pair (iis, iip), its revocation secret key irs, its initial revocation public key irp_0, and its digital signature key pair (sk_{id}, pk_{id}).

$\mathsf{AuthJoin}(params) \rightarrow (ais, aip, ars, arp_0)$: This is an algorithm that is executed by each attribute authority when it joins the system. The algorithm takes as input the global setup parameters $params$. It outputs an authority issuing secret and public key pair (ais, aip), its revocation secret key ars, and its initial revocation public key arp_0.

$\mathsf{UserJoin}\{\text{User}(iip, irp_j, params), \text{Identity provider}(iis, iip, irp_i, sk_{id}, pk_{id}, params)\} \rightarrow (s, \sigma_{id}, cert, w_{id}), (irp_{j+1}, \tilde{\sigma}_{id})$: This is a two-party protocol between a user and an identity provider that takes place when the user joins the system. The identity provider starts by knowing the global setup parameters $params$, its issuing secret and public key pair (iis, iip), its current revocation public key irp_j, and its digital signature key pair (sk_{id}, pk_{id}), while the user knows the public values $iip, irp_j, params$. The protocol outputs to the user her secret key s, validness proof σ_{id} of s, a certificate $cert$ that includes her personal user identity uid and some additional information (e.g., validity period), and a witness w_{id} for non-revocation, and to the identity provider its next revocation public key irp_{j+1} and a revocation handle $\tilde{\sigma}_{id}$ for σ_{id}.

$\mathsf{TraceIssue}\{\text{User}(s, cert, tp, \sigma_{id}, w_{id}, iip, irp, params), \text{Tracing Authority}(ts, tp, pk_{id}, iip, irp, params)\} \rightarrow (\sigma_T), (\perp)$: This is the tracing issue protocol that a user should run with each of $\theta + \Delta$ tracing authorities before being able generate traceable signatures. The tracing authority starts by knowing the global setup parameters $params$, its tracing secret and public key pair (ts, tp), the identity provider's digital signature, issuing, and revocation public keys pk_{id}, iip, and irp (i.e., the provider with whom the user ran the $\mathsf{UserJoin}$ protocol), while the user knows her secret key s, her certificate $cert$, tp, the non-revocation witness w_{id} of her σ_{id}, and public values $irp, params$. The protocol outputs to the user her tracing token σ_T.

$\mathsf{AttrIssue}\{\text{User}(s, cert, \omega, aip, arp_j, \sigma_{id}, w_{id}, iip, irp, params),$

Attribute Authority$(ais, aip, \omega, pk_{id}, arp_j, iip, irp, params)\} \rightarrow (\sigma_\omega, w), (arp_{j+1}, \tilde{\sigma}_\omega)$: This is the attribute issuing operation executed jointly by an attribute authority and a user. As inputs, the authority starts by knowing the global setup parameters *params*, its issuing secret and public key pair (ais, aip), the attribute ω to be given to the user, its current revocation public key arp_j, the identity provider's digital signature, issuing, and revocation public keys pk_{id}, iip, and irp (i.e., the provider with whom the user ran the UserJoin protocol), while the user knows a user secret key s, her user certificate *cert*, ω, the non-revocation witness w_{id} of her σ_{id}, and the public values $aip, arp_j, irp, params$. The protocol outputs to the user an attribute token σ_ω and the non-revocation witness w_ω for the attribute, and to the authority its next revocation public key arp_{j+1} and a revocation handle $\tilde{\sigma}_\omega$ for the issued attribute token.

Sign$(m, s, \beta, \sigma_{id}, w_{id}, \Sigma_\beta, W_\beta, \Sigma_T, iip, irp, AIP, ARP, TP_{\theta+\Delta}, t, J, n, params) \rightarrow \mu$: This is the signing algorithm that is executed by a user. It takes as input a message m, a secret key s, an attribute policy β, a global id σ_{id}, a non-revocation witness w_{id} for σ_{id}, an attribute token set Σ_β that proves that the owner of s conforms to the attribute policy β, a set W_β of non-revocation witnesses of those attributes, a set Σ_T of the $\theta + \Delta$ tracing tokens, the issuing public key iip and the revocation public key set irp of the identity provider that the user has run UserJoin with, the issuing public key set AIP and the revocation public key set ARP of the authorities that the user has obtained the attributes in β from, the tracing public key set $TP_{\theta+\Delta}$ of the tracing authorities that the user has interacted, the time period indicator t, a signature counter J, the allowed number n of the legitimate signatures by a user in a time period, and the global setup parameters *params*. The output of the algorithm is a VABS μ (including the tracing tag Φ) on m. At the end of each algorithm run, the user increments the counter J for validity of her next signature.

Verify$(m, \mu, \beta, iip, irp, AIP, ARP, TP_{\theta+\Delta}, t, SDB_j, n, params) \rightarrow (b, SDB_{j+1})$: This is the verification algorithm that is executed by a verifier for a given signature. It takes as input a message m, a VABS μ, an attribute policy β, the issuing public key iip and the revocation public key set irp of the identity provider,

the issuing public key set AIP and the revocation public key set ARP of the authorities for the attributes in β, the tracing public key set $TP_{\theta+\Delta}$ of the tracing authorities, the time period indicator t, the current signature database SDB_j, the allowed number n of the legitimate signatures by a user in a time period, and the global setup parameters *params*. The output of the algorithm is a bit b. b is defined as 1, if the user's global id is still valid (not revoked), the user has executed the TraceIssue operation with $\theta + \Delta$ tracing authorities, signed attributes in μ conform to β under AIP and has not been revoked in ARP, and there are no more than $n - 1$ signatures produced with the same key that is used to sign m in the time period according to the SDB_j. Otherwise, it is defined as 0. If the algorithm output is 1, the signature database is updated by the addition of μ (i.e., $SDB_{j+1} := SDB_j || \mu$). We usually make the database update output implicit.

$\mathsf{UserRevoke}(irs, irp_j, \tilde{\sigma}_{id}, params) \rightarrow irp_{j+1}$: This is the user revocation algorithm that is executed by an identity provider for revoking a user from the system completely by making her global id invalid. It takes as input the revocation secret key *irs* and the current revocation public key irp_j, the user's revocation handle $\tilde{\sigma}_{id}$, and the global setup parameters *params*. It outputs the new revocation public key irp_{j+1}.

$\mathsf{AttrRevoke}(ars, arp_j, \tilde{\sigma}_\omega, params) \rightarrow arp_{j+1}$: This is the revocation algorithm that is executed by an authority for revoking an attribute of a user. The inputs are the revocation secret key *ars* and the current revocation public key arp_j, the user's revocation handle $\tilde{\sigma}_\omega$ for the attribute ω, and the global setup parameters *params*. It outputs the new revocation public key arp_{j+1}.

$\mathsf{Trace}\{\textbf{for } i = 1, \ldots, \theta + \Delta, \text{ Tracing Authority}_i(\mu, ts_i, tp_i, TDB, params)\} \rightarrow uid$: This is the tracing protocol that is executed by $\theta + \Delta$ tracing authorities to reveal the signer of a VABS. Each tracing authority i starts by knowing the global setup parameters *params*, a VABS μ, its own tracing secret and public key pair (ts_i, tp_i), and the shared tracing database TDB. The protocol outputs to each honest tracing authority the *uid* of the signer of μ.

2.3.2 *Security Definitions*

A VABS scheme $\Pi = (\mathsf{GlobalSetup}, \mathsf{TraceSetup}, \mathsf{IdPJoin}, \mathsf{AuthJoin}, \mathsf{UserJoin}, \mathsf{TraceIssue}, \mathsf{AttrIssue}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{UserRevoke}, \mathsf{AttrRevoke}, \mathsf{Trace})$ must satisfy anonymity, tracing reliability, signature unforgeability, and soundness as security requirements. In all of our game-based definitions, given a VABS scheme Π, a PPT adversary $\mathcal{A}$, a challenger $\mathcal{C}$, a unary security parameter 1^λ, and a limit n for a time period duration δ, the first 2 steps of the games are as follows:

1. $\mathcal{C}$ runs $\mathsf{GlobalSetup}$ and obtains the global setup parameters *params*. $\mathcal{C}$ then gives 1^λ, n, δ, and *params* to $\mathcal{A}$. $\mathcal{A}$ is allowed to generate Δ malicious tracing authorities, but is required to give their public keys to $\mathcal{C}$. $\mathcal{C}$ generates the rest of the tracing authorities such that $\phi - \Delta$ of them honestly follow the protocol. All tracing authorities together run $\mathsf{TraceSetup}$. If $\mathcal{C}$ can detect any malicious behaviour by the tracing authorities under $\mathcal{A}$'s control during this setup phase (i.e., not outputting values from what would be expected by an honest tracing authority), the game terminates and the game's output is defined as 0. The time period counter t is initialized as $t := 1$, and is started.

2. At any step in the games,

   (a) $\mathcal{A}$ can generate polynomially-many identity provider and attribute authority keys, but is required to give the public keys to $\mathcal{C}$. $\mathcal{A}$ can also ask $\mathcal{C}$ to generate identity providers or attribute authorities. Then, $\mathcal{C}$ would honestly generate them, and share their public keys with $\mathcal{A}$.

   (b) $\mathcal{A}$ can generate polynomially-many users under its control or can ask $\mathcal{C}$ to generate honest users, and has the ability to run $\mathsf{UserJoin}$ with any identity provider or $\mathsf{AttrIssue}$ with any attribute authority for those users and any attribute ω. For all of the users generated, $\mathcal{A}$ can request $\mathsf{TraceIssue}$ to be run. $\mathcal{A}$ can demand revocation of identity or any attribute belonging to any of the generated users.

(c) $\mathcal{A}$ can adaptively request $\mathcal{C}$ to sign any message as any user under her control with any attribute policy that the user satisfies.

(d) $\mathcal{A}$ can adaptively request jointly running the Trace protocol with tracing auhorities under the control of $\mathcal{C}$ given any VABS as input.

(e) If $\mathcal{A}$ outputs any string to $\mathcal{C}$ other than what is explicitly expected from it in the protocol, then $\mathcal{C}$ just ignores it.

Below we provide the individual security games. Our signature unforgeability and anonymity definitions are as strong as the ones of [El Kaafarani et al., 2014b] (i.e., "strong full unforgeability" and "anonymity"), and the differences are due to having a different authority architecture and adding the revocation feature.

Anonymity of a user is defined via the game $\mathsf{VABSAnonym}_{\mathcal{A},\Pi}(\lambda)$:

3. $\mathcal{C}$ generates two secret keys s_0 and s_1. $\mathcal{C}$ runs UserJoin, TraceIssue, AttrIssue, and AttrRevoke for s_0 and s_1 with authorities of $\mathcal{A}$'s choice. We note that $\mathcal{A}$ can choose only one identity provider for both s_0 and s_1 to run the UserJoin with, as otherwise de-anonymization would be trivial.

4. $\mathcal{A}$ is given access to the signature oracles of both users: $\mathsf{Sign}(\tilde{m}, s_0, \tilde{\beta}, \sigma_{id,0}, w_{id,0}, \tilde{\Sigma}_{\tilde{\beta},0}, \tilde{W}_{\tilde{\beta},0}, \Sigma_{T,0}, iip, irp, \tilde{AIP}, \tilde{ARP}, TP_{\theta+\Delta}, t, J_0, n, params)$ and $\mathsf{Sign}(\tilde{m}, s_1, \tilde{\beta}, \sigma_{id,1}, w_{id,1}, \tilde{\Sigma}_{\tilde{\beta},1}, \tilde{W}_{\tilde{\beta},1}, \tilde{\Sigma}_{T,1}, iip, irp, \tilde{AIP}, \tilde{ARP}, TP_{\theta+\Delta}, t, J_1, n, params)$; where $(\tilde{m}, \tilde{\beta})$ is chosen by $\mathcal{A}$ as part of its queries; $\sigma_{id,b}$, $w_{id,b}$, and $\Sigma_{T,b}$ are the global id, its non-revocation witness, and the tracing tokens for s_b; iip, irp, and $TP_{\theta+\Delta}$ are the issuing and revocation public keys of the identity provider and the public key set of the tracing authorities (respectively); the value t is the current time, and $0 \leq J_0, J_1 < n$ are the signature counters (incremented with each signing within t and reset between time periods) for the corresponding signing key. Other values with tilde sign are chosen by $\mathcal{C}$ as an honest signer would do.

   Essentially, $\mathcal{C}$ sets $\tilde{AIP}$ and $\tilde{ARP}$ as the public keys of the authorities that can issue attributes in $\tilde{\beta}$, $\tilde{\Sigma}_{\tilde{\beta},0}$ and $\tilde{W}_{\tilde{\beta},0}$ as the attribute tokens and non-

revocation witnesses that are issued to s_0 related to $\tilde{\beta}$, $\tilde{AIP}$ and $\tilde{ARP}$ as issuing and revocation public keys of all authorities known to $\mathcal{C}$ that can issue the attributes in $\tilde{\beta}$. Each oracle stops responding to queries during a time period when its counter J_0/J_1 reach $n-1$.

5. $\mathcal{A}$ gives $\mathcal{C}$ an attribute policy β, and two messages m_0 and m_1 to $\mathcal{C}$, when both $J_0 < n-1$ and $J_1 < n-1$. If any of the counters is equal to $n-1$, $\mathcal{A}$ loses. If there exists any missing attribute tokens on s_0 and s_1 for conforming to β, $\mathcal{C}$ obtains them. For each attribute in β, $\mathcal{C}$ makes sure both users have obtained an attribute token from the same authority.

6. $\mathcal{C}$ picks a random bit b. $\mathcal{C}$ signs both messages as s_b by running $\mu_0 \leftarrow \mathsf{Sign}(m_0, s_b, \beta, \sigma_{id,b}, w_{id,b}, \Sigma_{\beta,b}, W_{\beta,b}, \Sigma_{T,b}, iip, irp, AIP, ARP, TP_{\theta+\Delta}, t, J_b, n, params)$ and $\mu_1 \leftarrow \mathsf{Sign}(m_1, s_{\bar{b}}, \beta, \sigma_{id,\bar{b}}, w_{id,\bar{b}}, \Sigma_{\beta,\bar{b}}, W_{\beta,\bar{b}}, \Sigma_{T,\bar{b}}, iip, irp, AIP, ARP, TP_{\theta+\Delta}, t, J_{\bar{b}}, n, params)$, where AIP and ARP are issuing and revocation public keys of all authorities known to $\mathcal{C}$ that have issued the attributes in β, and Σ_β and $W_{\beta,b}$ are the set of attribute tokens their non-revocation witnesses for s_b for proving β. $TP_{\theta+\Delta}$ is required to be the same to prevent trivial de-anonymization. The other values are set as before. $\mathcal{C}$ then gives μ_0 and μ_1 to $\mathcal{A}$.

7. $\mathcal{A}$ eventually returns a bit b'. The output of the game is defined as 1 (i.e., $\mathcal{A}$ wins), if $b = b'$ and $\mathcal{A}$ has not asked any tracing authority under $\mathcal{C}$'s control for participation in Trace of μ_0 or μ_1. Otherwise, the output of the game is defined as 0 (i.e., $\mathcal{A}$ loses).

Definition 1 (Anonymity)**.** *A VABS scheme* Π *provides* anonymity, *if* $\forall n, \delta \in poly(\lambda)$, *PPT adversary* $\mathcal{A}$, *there exists a negligible function* $\mathsf{n}(\cdot)$ *such that*

$$\Pr[\mathsf{VABSAnonym}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{n}(\lambda)$$

Our **reliable traceability** definition ensures that tracing is done correctly as long as the adversary controls at most Δ tracing authorities. Our definition covers

"traceability" of [El Kaafarani et al., 2014b] and "non-frameability" of [Ghadafi, 2015], as long as $\Phi-$ tracing authorities are honest. Our notion implies "tracing soundness" of [Ghadafi, 2015] by requiring a single original signer per VABS that is deterministically traced. We note that this is the first work that provides a compact tracing definition in the context of MA-ABS in the presence of multiple tracing authorities, although in the context of group signatures "traceability", "non-frameability", and "tracing soundness" definitions have been previously provided for distributed tracing by [Ghadafi, 2014].[2] Consider the following game $\mathsf{VABSTrace}_{\mathcal{A},\Pi}(\lambda)$:

3. The output of the game is defined as 1 (i.e., $\mathcal{A}$ wins), if for any VABS that verifies, at least θ tracing authorities (therefore necessarily including honest ones) do *not* output the *uid* that belongs to the original signer of the comprising VABS during any run of the Trace protocol. Otherwise, the output of the game is defined as 0 (i.e., $\mathcal{A}$ loses).

Definition 2 (Reliable Traceability)**.** *A VABS scheme* Π *provides* reliable traceability*, if* $\forall n, \delta \in poly(\lambda)$*, for each PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{n}(\cdot)$ *such that*

$$\Pr[\mathsf{VABSTrace}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \mathsf{n}(\lambda)$$

The **signature unforgeability** definition that we provide is similar to the unforgeability definition of [Maji et al., 2011], and is even stronger than that by allowing the adversary to issue any attribute to a user while the latter does not allow this. For our **signature unforgeability** definition, consider the following signature unforgeability game $\mathsf{VABSForge}_{\mathcal{A},\Pi}(\lambda)$:

3. $\mathcal{A}$ is allowed to fully corrupt all ϕ tracing authorities.

[2]Distributed tracing of [Ghadafi, 2014] and our threshold tracing are different in that the former enforces all tracing authorities to join the Trace operation, while the latter enables a settable threshold-many of them to execute Trace

4. $\mathcal{A}$ returns an attribute ω to $\mathcal{C}$. ω must not be queried before as $\mathsf{AttrIssue}$ to the authorities under $\mathcal{C}$'s control for users under $\mathcal{A}$'s control.

5. $\mathcal{A}$ is restricted in a way that if $\mathcal{A}$ queries to an authority oracle as $\mathsf{AttrIssue}$ with ω for users under $\mathcal{A}$'s control, ω is issued but is *immediately revoked*[3].

6. $\mathcal{A}$ eventually generates a message m (not queried for signing to the users under $\mathcal{C}$'s control for the attribute policy $\beta \wedge \omega$ where β is any policy) and a VABS μ. The output of the game is defined as 1 (i.e., $\mathcal{A}$ wins), if $\mathsf{Verify}(m, \mu, \beta \wedge \omega, IIP, IRP, AIP, ARP, TP_\phi, t, SDB, n, params) = 1$, where IIP, IRP, AIP, ARP, and TP_ϕ are the issuing and revocation public keys of all identity providers, the issuing and revocation public keys of all attribute authorities that can issue the attributes in $\beta \wedge \omega$, and the public keys of all tracing authorities known to $\mathcal{C}$. Otherwise, the output of the game is defined as 0 (i.e., $\mathcal{A}$ loses). Note that this step enforces $\mathcal{A}$ to ask to $\mathcal{C}$ for generation of at least one authority oracle to check for ω.

Definition 3 (Signature Unforgeability)**.** *A VABS scheme* Π *provides* signature unforgeability*, if* $\forall n, \delta \in poly(\lambda)$*, for each PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{n}(\cdot)$ *such that*

$$\Pr[\mathsf{VABSForge}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \mathsf{n}(\lambda)$$

The **soundness** definition that we provide is similar to the soundness definition of [Camenisch et al., 2006], but differs slightly due to usage, i.e., ours limits the number of signatures by a party that is verified, while the latter limits credential proofs *per attribute*. Formally, consider the following soundness game $\mathsf{VABSSound}_{\mathcal{A},\Pi}(\lambda)$:

3. $\mathcal{A}$ is allowed to fully corrupt all tracing authorities.

[3]This is required to make sure that a user cannot use her revoked attributes

4. The output of the game is defined as 1 (i.e., $\mathcal{A}$ wins), if $\mathcal{A}$ generates at least $n+1$ message and VABS pairs within some time period for the same user s such that honest executions of the Verify algorithm on at least $n+1$ of those pairs within the same time period output 1. Otherwise, the output of the game is defined as 0 (i.e., $\mathcal{A}$ loses).

Definition 4 (Soundness). *A VABS scheme* Π *provides* soundness, *if* $\forall n, \delta \in poly(\lambda)$, *for each PPT adversary* $\mathcal{A}$, *there exists a negligible function* $\mathsf{n}(\cdot)$ *such that*

$$\Pr[\mathsf{VABSSound}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \mathsf{n}(\lambda)$$

2.4 Our Modular VABS

In this section, we present our VABS solution in a modular manner. In the protocol and algorithm descriptions, *blue* parts that are are used for user/attribute revocation, and *green* parts are employed for reliable threshold traceability. If these properties are not required for the application, they can be omitted for better efficiency, without affecting security.

Global Setup. In the start of a VABS system, in GlobalSetup, with any number of k volunteers a protocol is run for obtaining global setup parameters that are used in the algorithms and protocols later. To do so, a cyclic group $\langle g \rangle = \mathbb{G}$ of prime order q such that $2^{\ell_q - 1} < q < 2^{\ell_q}$ is generated, where $\ell_q \in \Theta(\lambda)$. Another generator h of $\mathbb{G}$ is also generated in a distributed computation [Pedersen, 1991b, Gennaro et al., 2007] so that $\log_g h$ would be intractable. Essentially, each party i (a user or an authority) that wants to join the generation process of h picks a random value $x_i \in \mathbb{Z}_q$, and then publishes $h_i := g^{x_i}$ and NIZKPoK$\{(x_i) : h_i = g^{x_i}\}$ with authentication tags.[4] At the end of the setup, all of the k parties involved compute $h := \prod_{i \in \{1,\ldots,k\}} h_i$ and output the parameters $(q, \mathbb{G}, g, h)$.

Tracing Setup. After the global setup, ϕ tracing authorities run the protocol TraceSetup for generating tracing parameters. First, two generators g_1 and

[4]The parties may be required to publish them on a public ledger, to maintain the consistency among parties and to thwart equivocation attempts.

g_2 of the group $\mathbb{G}$ are generated in a distributed fashion by all ϕ tracing authorities. Then, the authorities together generate the encryption public key $tep := (p, q, g_1, g_2, c, d, h_1)$ and their decryption secret key shares tes_i such that θ of them can decrypt a ciphertext (i.e., the KeyGeneration protocol of [Canetti and Goldwasser, 1999]). Each authority i runs Algorithm 1:

$$(\mathbf{g}_i'', \mathbf{h}_i'', \mathbf{j}_i'', \mathsf{O}_i, \mathsf{p}_i''', \mathsf{q}_i''', \mathrm{NIZKPoK}_1^i, \mathrm{NIZKPoK}_2^i) \leftarrow \mathsf{CLGen}(1^\lambda)$$

Then, it publishes its public key $tp_i := (tep, \mathbf{g}_i'', \mathbf{h}_i'', \mathbf{j}_i'', \mathsf{O}_i)$, $\mathrm{NIZKPoK}_1^i$, and $\mathrm{NIZKPoK}_2^i$, keeping its secret key $ts_i = (tes_i, \mathsf{p}_i''', \mathsf{q}_i''')$. Each honest authority checks the transcripts of every other tracing authority, and signals an error in case of detection of any malicious behaviour.

Authority Join. To join the system, each identity provider or attribute authority runs:

$$(\mathbf{g}_i, \mathbf{h}_i, \mathbf{j}_i, \mathsf{N}_i, \mathsf{p}_i', \mathsf{q}_i', \mathrm{NIZKPoK}_1^i, \mathrm{NIZKPoK}_2^i) \leftarrow \mathsf{CLGen}(1^\lambda)$$

$$(\mathbf{u}_{i,0}, \mathbf{g}_i', \mathbf{h}_i', \mathsf{M}_i, \mathsf{p}_i'', \mathsf{q}_i'', \mathrm{NIZKPoK}_3^i, \mathrm{NIZKPoK}_4^i) \leftarrow \mathsf{CLGen}(1^\lambda)$$

for generating the public keys that will be used by the authority for issuing tokens to user. Then, it publishes its generated issuing public key $ip_i := (\mathbf{g}_i, \mathbf{h}_i, \mathbf{j}_i, \mathsf{N}_i)$ and revocation public key $rp_{i,0} := (\mathbf{u}_{i,0}, \mathbf{g}_i', \mathbf{h}_i', \mathsf{M}_i)$ together with the generated proofs $\mathrm{NIZKPoK}_1^i$, $\mathrm{NIZKPoK}_2^i$, $\mathrm{NIZKPoK}_3^i$, and $\mathrm{NIZKPoK}_4^i$ for honest performing of the join operation. Here ip and rp correspond to iip and irp for an identity provider and to aip and arp for an attribute authority, respectively. We note that the value $\mathbf{u}_{i,0}$ of the revocation public key will keep being updated as an RSA accumulator[5] and will accumulate the revocation handles [Camenisch and Lysyanskaya, 2002]. The authority keeps its generated issuing secret keys $iis_i/ais_i := (\mathsf{p}_i', \mathsf{q}_i')$ and revocation secret keys $irs_{i,0}/ars_{i,0} := (\mathsf{p}_i'', \mathsf{q}_i'')$. Here is and rs correspond to iis and irs for an identity provider and to ais and

[5]Here RSA accumulator preferred due to its dynamic addition and revocation abilities. Also, the signer uses the proof protocol [Camenisch and Lysyanskaya, 2002] for showing she is not revoked without disclosing her secret.

ars for an attribute authority, respectively. For efficiency, each attribute authority only has a fixed-length public key, no matter how many different attributes it can issue, in contrast to [Camenisch et al., 2006]. Additionally, each identity provider runs the DSKeyGen algorithm of a conventional digital signature scheme (DSKeyGen, DSSign, DSVerify) to obtain a public-private key pair (pk_{id}, sk_{id}) and publishes the public key. The obtained keys are used for signing the ceritficates of users.

User Join. To join the system, a user interacts with an identity provider through an authenticated channel, where they run the UserJoin protocol in Figure 2.1, i.e., an extended version of the "Signature on a Committed Value" protocol of [Camenisch and Lysyanskaya, 2003]. Note that we remove the Pedersen commitment input from the original protocol of [Camenisch and Lysyanskaya, 2003], as it runs on Fujisaki-Okamoto commitment [Fujisaki and Okamoto, 1997] after proving the equality of the committed values. At the end of the protocol, the user obtains a certificate *cert*, which is composed of user identification information, the commitment C_s to her secret key, possibly expiration date and other information, together with the signature of the identity provider on them. The user uses the certificate *cert* given by the identity provider to obtain tokens from attribute authorities using the same secret key. The user also obtains σ_{id} as a CL signature on her secret key s, which is used each time she signs a message for the purpose of limiting the number of signatures within a time period. The secret key has two random components as $s = s_1 + s_2$ where s_1 is contributed by the user and s_2 is contributed by the identity provider, to ensure randomness and uniqueness. In contrast to [Camenisch et al., 2006], in our solution, a user has only one secret key s due to the removal of the double spending tag that was used in e-cash solutions.

Tracing Issue. Upon running the protocol in Figure 2.1 with an identity provider, the user interacts with at least $\theta + \Delta$ tracing authorities, with each of which she runs the TraceIssue protocol given in Figure 2.2 to obtain her tracing tokens $\sigma_{T,i}$, i.e., a CL signature on s. The tracing authorities together register the tuple (uid, g_1^s) into the shared tracing database TDB, which only permits update by

The UserJoin **Protocol**

User $(iip_i, irp_{ij}, params)$

$s_1, \rho \twoheadleftarrow \mathbb{Z}_q^*; \mathsf{C}_{s_1} := \mathbf{g}_i^{s_1}\mathbf{h}_i^{\rho}$
$\mathrm{NIZKPoK} := \mathrm{NIZKPoK}\{(s_1, \rho) : \mathsf{C}_{s_1} = \mathbf{g}_i^{s_1}\mathbf{h}_i^{\rho}\}$

$\xrightarrow{\mathsf{C}_{s_1}, \mathrm{NIZKPoK}}$

Identity Provider i $(iis_i, iip_i, pk_{id}, sk_{id}, irp_{ij}, params)$

if NIZKPoK does not verify
abort
$s_2 \twoheadleftarrow \mathbb{Z}_q^*; \mathsf{C}_{s_1+s_2} := \mathsf{C}_{s_1}\mathbf{g}_i^{s_2}; \tilde{\sigma}_{id} \twoheadleftarrow O(1^{\lambda})$ prime number
$r \twoheadleftarrow O(1^{\lambda}); v := (\mathsf{C}_{s_1+s_2}\mathbf{h}_i^{r}\mathbf{j}_i)^{\frac{1}{\tilde{\sigma}_{id}}}$
$cert_{info} := (uid, \mathsf{C}_{s_1+s_2}, \ldots); \mathbf{u}_{i,j+1} := \mathbf{u}_{i,j}^{\tilde{\sigma}_{id}}$
$cert := (cert_{info}, \sigma \leftarrow \mathsf{DSSign}_{sk_{id}}(cert_{info}))$

$\xleftarrow{s_2, \tilde{\sigma}_{id}, r, v, cert}$

$s := s_1 + s_2; \sigma_{id} := (\rho + r, \tilde{\sigma}_{id}, v); w := \mathbf{u}_{i,j}$

Figure 2.1: The UserJoin protocol between a new user and an identity provider. Note that ... denote the fact that $cert_{info}$ may include additional information for identification of the user or the validity period of the $cert$.

the consensus of θ tracing authorities. Note that there are generic ways of maintaining such a consistent and consensus-based database (e.g., the authenticated Byzantine Fault Tolerance [Lamport et al., 1982, Veronese et al., 2013] protocol). We highlight that if an identity provider does not randomize a user secret key s via picking s_2 randomly and generating the CL signature on $s = s_1 + s_2$ as in honest execution of UserJoin, and thereby let two different users have the same secret key s, then the tracing authorities can detect it at the last step of their TraceIssue protocol execution.

Attribute Issue. To obtain the token for a descriptive attribute ω from an authority i, the user and the authority run the protocol in Figure 2.3 through a secure and authenticated channel (i.e., an extended version of the "Signature on a Committed Value" protocol of [Camenisch and Lysyanskaya, 2003]), where the hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ is modeled as a random oracle. At a high level, the user obtains a CL signature on $s + H(\omega)$, after proving in zero knowledge that she holds the identity s to tie her attributes to her identity blindly. To obtain the token for a numeric attribute ω with value ν from an authority i, the authority runs the same steps with $\mathcal{C}'_{s+H(\omega)+\nu}$ instead of $\mathcal{C}'_{s+H(\omega)}$ to generate a CL

The TraceIssue Protocol

User $(s, cert, tp_i, \sigma_{id}, w_{id}, iip, irp, params)$		**Tracing Authority** i $(ts_i, tp_i, pk_{id}, iip, irp, , params)$
$\rho' \twoheadleftarrow \mathbb{Z}_q^*; \mathsf{C}'_s := \mathbf{g}_i^s \mathbf{h}_i^{\rho'}$ From $cert$ extract $\mathsf{C}_s = \mathsf{C}_{s_1+s_2}$ $\mathrm{NIZKPoK}_1 :=$ NIZKPoK that C_s and C'_s are commitments to the same value s $\mathrm{NIZKPoK}_2 :=$ NIZKPoK that C'_s is a commitment to the exponent of g_1^s $\mathrm{NIZKPoK}_3 :=$ NIZKPoK$\{(s, \rho, \sigma_{id}, w_{id}) : \sigma_{id}$ is a σ_{CL} on s committed in C_s verifiable with iip, and w_{id} is a witness that $\tilde{\sigma}_{id}$ is accumulated in $irp\}$	$\xrightarrow{\mathsf{C}'_s, g_1^s, cert, \mathrm{NIZKPoK}_1, \mathrm{NIZKPoK}_2, \mathrm{NIZKPoK}_3}$	**if** σ or any NIZKPoK does not verify abort $\tilde{\sigma}_T \twoheadleftarrow O(1^\lambda)$ prime number, $r \twoheadleftarrow O(1^\lambda)$ $v := (\mathsf{C}'_s \mathbf{h}_i^r \mathbf{j}_i)^{\frac{1}{\tilde{\sigma}_T}}$ **if** (g_1^s, uid') is already in TDB **and** $uid' \neq uid$ detect malicious identity provider; abort **else** record (g_1^s, uid) into TDB
$\sigma_{T,i} := (\rho' + r, \tilde{\sigma}_T, v)$	$\xleftarrow{\tilde{\sigma}_T, r, v}$	

Figure 2.2: The TraceIssue protocol between a user and a tracing authority.

signature on $s + H(\omega) + \nu$.We enforce randomization of the attributes by $H(\omega)$ to prevent malicious collusion of users. Unlike the ABS schemes of [Cao et al., 2012, Okamoto and Takashima, 2013, El Kaafarani et al., 2014b], we employ a hash function (modeled as a random oracle) for randomization, instead of trusting authorities for that purpose.

Sign. We present our Sign algorithm with Boolean AND policy $\omega_1 \wedge \ldots \wedge \omega_k$ in Algorithm 2, where ℓ_{cnt} is a system parameter such that $\ell_q - 2 \geq \ell_{cnt} > n + 1/2$. The algorithm outputs a serial number S for the VABS, commitments C_J and C_s to the number J of generated signatures in the current time period and the user's secret key s, and a number of proofs for the correct construction of the signature and knowledge of CL signatures on s plus the randomized attribute. The tracing tag Φ is the ciphertext obtained by encrypting g_1^s with TPKE [Canetti and Goldwasser, 1999]. To sign with an example OR policy $(\omega_1 \wedge \omega_2) \vee (\omega_3 \wedge \omega_4)$, the subpolicies $(\omega_1 \wedge \omega_2)$ and $(\omega_3 \wedge \omega_4)$ can be combined with an OR proof in an

Algorithm 2 Our Sign algorithm with AND policy.

input: a message m, a secret key s, an AND policy $\beta = \omega_1 \wedge \ldots \wedge \omega_k$ (where if ω_i is a numeric attribute, it has value ν_i), a global id σ_{id}, a non-revocation witness w_{id} for σ_{id}, a set $\Sigma_\beta = (\sigma_{\omega_1}, \ldots, \sigma_{\omega_k})$ of the attribute tokens for β, a set $W_\beta = (w_1, \ldots, w_k)$ of non-revocation witness of those attributes, a set $\Sigma_T = (\sigma_{T,1}, \ldots, \sigma_{T,\theta+\Delta})$ of the tracing tokens, the issuing public key iip and the revocation public key $irp$of the provider of the σ_{id}, the issuing public key set AIP and the revocation public key set $ARP$of the authorities that have issued Σ_β, the tracing public key set TP of the tracing authorities that have issued Σ_T, the current time period $t \geq 1$, the number J of ABSs generated by the signer in the current period, the allowed number n of the legitimate signatures by a user in a time period, the signature and global setup parameters $params = (q, \mathbb{G}, g, h)$.

output: an ABS $\sigma = (S, \mathsf{C}_J, \mathsf{C}_s, \Phi, \mathsf{SoK}_1, \ldots, \mathsf{SoK}_{3+k}, \mathsf{SoK}_{4+k}, \ldots, \mathsf{SoK}_{5+k+\theta+\Delta})$.

$\rho_1, \rho_2, \rho_3 \twoheadleftarrow \mathbb{Z}_q$; $S := g^{1/(s+t2^{\ell_{cnt}}+J)}$

$\mathsf{C}_J := g^J h^{\rho_1}$; $\mathsf{C}_s := g^s h^{\rho_2}$

$\Phi := (g_1^{\rho_3}, g_2^{\rho_3}, g_1^s h_1^{\rho_3}, c^{\rho_3} d^{\rho_3 \kappa})$ where $\kappa := H(g_1^{\rho_3}, g_2^{\rho_3}, g_1^s h_1^{\rho_3})$

SoK_1 :=$\mathsf{SoK}[m]\{(J, \rho_1) : J \in \{0, \ldots, n-1\} \wedge \mathsf{C}_J = g^J h^{\rho_1}\}$

SoK_2 :=$\mathsf{SoK}[m]\{(\alpha, \gamma) : S = g^\alpha \wedge g = (\mathsf{C}_s g^{t2^{\ell_{cnt}}} \mathsf{C}_J)^\alpha h^\gamma\}$

SoK_3 :=$\mathsf{SoK}[m]\{(s, \rho_2, \sigma_{id}, w_{id}) : \sigma_{id}$ is a σ_{CL} on s committed in C_s verifiable with iip, and w_{id} is a witness that $\tilde{\sigma}_{id}$ is accumulated in $irp\}$

for $i = 1, \ldots, k$ **do**

  if ω_i is a descriptive attribute **then**

    SoK_{3+i} :=$\mathsf{SoK}[m]\{(s, \rho_2, \sigma_{\omega_i}, w_i) : \sigma_{\omega_i}$ is a σ_{CL} on $s + H(\omega_i)$ committed in $\mathsf{C}_{s+H(\omega_i)} = \mathsf{C}_s g^{H(\omega_i)}$ verifiable with

    $aip_i \in AIP$, and w_i is a witness that $\tilde{\sigma}_{\omega_i}$ is accumulated in $arp_i \in ARP\}$

  if ω_i is a numeric attribute **then**

    $\rho_{3+i} \twoheadleftarrow \mathbb{Z}_q$; $\mathsf{C}_{\nu_i} := g^{\nu_i} h^{\rho_{3+i}}$

    SoK'_{3+i} :=$\mathsf{SoK}[m]\{(\nu_i, \rho_{3+i}) : \nu_i$ is in the given range defined in $\beta\}$

    SoK''_{3+i} :=$\mathsf{SoK}[m]\{(s, \rho_2, \sigma_{\omega_i}, w_i) : \sigma_{\omega_i}$ is a σ_{CL} on $s + H(\omega_i) + \nu_i$ committed in $\mathsf{C}_{s+H(\omega_i)+\nu_i} = \mathsf{C}_s g^{H(\omega_i)} \mathsf{C}_{\nu_i}$

    verifiable with $aip_i \in AIP$, and w_i is a witness that $\tilde{\sigma}_{\omega_i}$ is accumulated in $arp_i \in ARP\}$

    $\mathsf{SoK}_{3+i} = (\mathsf{C}_{\nu_i}, \mathsf{SoK}'_{3+i}, \mathsf{SoK}''_{3+i})$

SoK_{4+k} :=$\mathsf{SoK}[m]$ that Φ is constructed correctly

  SoK_{5+k} :=$\mathsf{SoK}[m]$ that C_s and $g_1^s h_1^{\rho_3}$ are commitments to the same value

  for $i = 1, \ldots, \theta + \Delta$ **do**

    SoK_{5+k+i} :=$\mathsf{SoK}[m]\{(s, \rho_2, \sigma_{T,i}) : \sigma_{T,i}$ is a σ_{CL} on s committed in C_s verifiable with $tp_i \in TP\}$

The AttrIssue **Protocol**

User $(s, cert, \omega, aip_i, arp_{i,j}, \sigma_{id}, w_{id}, iip, irp, params)$		**Attribute Authority** i $(ais_i, aip_i, \omega, pk_{id}, arp_{i,j}, iip, irp, params)$
$\rho' \twoheadleftarrow \mathbb{Z}_q^*; \mathsf{C}'_s := \mathbf{g}_i^s \mathbf{h}_i^{\rho'}$ From $cert$ extract $\mathsf{C}_s = \mathsf{C}_{s_1+s_2}$ $\mathrm{NIZKPoK}_1 :=$ NIZKPoK that C_s and C'_s are commitments to the same value s $\mathrm{NIZKPoK}_2 :=$ NIZKPoK$\{(s, \rho, \sigma_{id}, w_{id}) : \sigma_{id}$ is a σ_{CL} on s committed in C_s verifiable with iip, and w_{id} is a witness that $\tilde{\sigma}_{id}$ is accumulated in $irp\}$		
	$\xrightarrow{\mathsf{C}'_s, cert, \mathrm{NIZKPoK}_1, \mathrm{NIZKPoK}_2}$	
		if σ or any NIZKPoK does not verify abort $\tilde{\sigma}_\omega \twoheadleftarrow O(1^\lambda)$ prime number, $r \twoheadleftarrow O(1^\lambda)$; $\mathbf{u}_{i,j+1} := \mathbf{u}_{i,j}^{\tilde{\sigma}_\omega}$ $\mathsf{C}'_{s+H(\omega)} := \mathsf{C}'_s H(\omega)$; $v := (\mathsf{C}'_{s+H(\omega)} \mathbf{h}_i^r \mathbf{j}_i)^{\frac{1}{\tilde{\sigma}_\omega}}$
	$\xleftarrow{\tilde{\sigma}_\omega, r, v}$	
$\sigma_\omega := (\rho' + r, \tilde{\sigma}_\omega, v); w := \mathbf{u}_{i,j}$		

Figure 2.3: The AttrIssue protocol between a user and an attribute authority for a descriptive attribute ω.

SoK using [Cramer et al., 1994].

SoKs of Sign **given in Algorithm 2.** The proof of knowledge schemes for SoK_1 and SoK_2 can be realized using [Schnorr, 1991] and [Boudot, 2000] by converting them into non-interactive signatures on m, thereby showing that the signature was not produced more than n times in the current time period (also to be verified against a database to ensure the serial number is used only once). For SoK_3, the signer computes the commitments $C_s := g^s h^{\rho_1}$, $C_{\rho+r} := g^{\rho+r} h^{\rho_2}$, $C_{\tilde{\sigma}_{id}} := g^{\tilde{\sigma}_{id}} h^{\rho_3}$, $C_v := v g^{\rho_4}$, $C_{\rho_4} := g^{\rho_4} h^{\rho_5}$, $C_{\rho_4 \tilde{\sigma}_{id}} := g^{\rho_4 \tilde{\sigma}_{id}} h^{\rho_6}$, and $C := (C_v)^{\tilde{\sigma}_{id}} h^{\rho_7}$ using $\sigma_{id} = (\rho + r, \tilde{\sigma}_{id}, v)$ and picking $\rho_1, \ldots, \rho_7$ at random as in "Proof of Knowledge of a Signature" protocol of [Camenisch and Lysyanskaya, 2003]. We note that the commitments here are listed in the same order as in [Camenisch and Lysyanskaya, 2003]. She then generates zero-knowledge *OR proofs* of all listed proofs in the mentioned proof scheme of [Camenisch and Lysyanskaya, 2003] as SoKs on m. Moreover, she computes the commitments and values listed in "Efficient Proof That a Committed Value Was Accumulated" protocol in [Camenisch and Lysyanskaya, 2002] to prove that the committed value in $C_{\tilde{\sigma}_{id}}$ is accumulated in the part

$\mathbf{u}_{i,j+1}$ of $irp_{i,j}$ (i.e., the current revocation public key of the authority i). For an SoK$\in$(SoK$_{3+1}$, ..., SoK$_{3+k}$), if the attribute is descriptive, the signer also follows the same method as SoK$_3$ by replacing s with $s + H(\omega)$ and id with ω. Thus, these proofs enable the signer to prove that she has a valid id from an *id* provider, and the required attributes from the attribute authorities, without disclosing her identity. SoK$_{4+k}$ is generated showing ρ_3 values are the same in all of $g_1^{\rho_3}$, $g_2^{\rho_3}$, $g_1^s {h_1}^{\rho_3}$ as a proof on m and $c^{\rho_3} d^{\rho_3 \kappa}$. SoK$_{5+k}$ is composed of the conventional proof of equality of committed values on m. Overall, these proofs show that the user provided her own tracing tag as part of the VABS. If the attribute is numeric, for SoK$'$ the user follows the same method as SoK$_3$ by replacing s with $s + H(\omega) + \nu$ and id with ω. For SoK$''$, the user generates the range proof given in Bulletproofs [Bünz et al., 2018]. SoK$_{5+k+1}$, ..., SoK$_{5+k+\theta+\Delta}$ are again generated by the same technique as in the previous ones for the knowledge of CL signatures for identity and attribute tokens, showing that she obtained at least θ CL signatures from tracing authorities on her same identity. Note that instead of encrypting s, the signer encrypts g_1^s within the tracing tag Φ, which has two advantages: secrecy of s and ease of SoK$_{5+k}$.

Remark: We emphasize that our scheme has some important improvements over what would have been obtained if one had just converted [Camenisch et al., 2006] to a non-interactive signature scheme. First, we provide a method for attribute authorities to generate a signature on the same secret key of the user to eliminate collusion among the users. That is, an authority issues an attribute ω to a party with secret key s by generating a CL signature on $s + H(\omega)$, which makes it intractable for another party to use that CL signature with its secret key s', since it requires that $s + H(\omega) = s' + H(\omega')$ where $s \neq s'$ and H is a random oracle. Second, regarding efficiency, we require computation of the serial number S and the commitments C_J and C_s only once per signature, since they are used in counting the number of signatures a person has generated (but not how many times a particular one of her attributes is used), and the commitment $C_{s+H(\omega)}$ can be obtained efficiently from C_s. Third, our algorithm effectively combines the attribute

proving with the threshold encryption scheme for traceability. Fourth, due to the fact that we only require the more than n signing attempts to be detectable but not de-anonymized (which is done differently during tracing), we remove the double spending tags of [Camenisch et al., 2006] for efficiency.

Verify. Algorithm 3 shows our Verify algorithm for an AND policy. It essentially returns 1, if all the SoKs in the signature verify, and the serial number S has not appeared in a signature before (to ensure n-times limited use). Otherwise, it returns 0.

Algorithm 3 Our Verify algorithm for AND policy

input: a message m, an ABS μ, an AND policy $\beta = \omega_1 \wedge \ldots \wedge \omega_k$, the issuing public key iip and the revocation public key irp of the provider of the σ_{id}, the issuing public key set AIP and the revocation public key set ARP of the authorities that have issued Σ_β, the tracing public key set TP of the tracing authorities that have issued Σ_T, the current time period t, a signature database SDB_j, the allowed number n of the legitimate signatures by a user in a time period, and global setup parameters $params = (q, \mathbb{G}, g, h)$.

output: a bit b and implicitly an updated signature database SDB_{j+1}.

$(S, \mathsf{C}_J, \mathsf{C}_s, \Phi, \mathrm{SoK}_1, \ldots, \mathrm{SoK}_{3+k}, \mathrm{SoK}_{4+k}, \ldots, \mathrm{SoK}_{5+k+\theta+\Delta}) := \mu$
if S is a part of any signature in SDB **then**
  $b := 0$; $SDB_{j+1} := SDB_j$; **abort**
for $i = 1, \ldots, k$ **do** $\mathsf{C}_{s+H(\omega_i)} := \mathsf{C}_s g^{H(\omega_i)}$
  for $i = 1, \ldots, 3+k$ **do**
    if SoK_i does not verify **then**
      $b := 0$; $SDB_{j+1} := SDB_j$; **abort**
  for $i = 4+k, \ldots, 5+k+\theta$ **do**
    if SoK_i does not verify **then**
      $b := 0$; $SDB_{j+1} := SDB_j$; **abort**
$b := 1$; $SDB_{j+1} := SDB_j || \mu$

Revocation. In our solution, the identity providers may revoke dishonest users, and attribute authorities may revoke the attributes that they assigned (e.g., when a student graduates). During UserRevoke, to revoke an issued σ_{id} using the related revocation handle $\tilde{\sigma}_{id}$, an identity provider i updates its current revocation

public key $irp_{i,j} = (\mathbf{u}_{i,j}, \mathbf{g}'_i, \mathbf{h}'_i, M_i)$ as $irp_{i,j+1} = (\mathbf{u}_{i,j+1}, \mathbf{g}'_i, \mathbf{h}'_i, M_i)$ where $u_{i,j+1} := u_{i,j}^{\tilde{\sigma}_{id}^{-1} \bmod 4\mathsf{p}''_i\mathsf{q}''_i} \bmod \mathsf{M}_i$. Note that according to [Camenisch and Lysyanskaya, 2002], after each issuing or revocation, the identity provider may publish $\tilde{\sigma}_{id}$, so that the other users can update their witnesses. Instead, it is also possible to periodically issue or revoke users in a batch as $\psi := \prod_{i \in revoked} \tilde{\sigma}_{id,i} \Big/ \prod_{i \in issued} \tilde{\sigma}_{id,i} \bmod 4\mathsf{p}''_j\mathsf{q}''_j$. Then, a user k with $gcd(\psi, \tilde{\sigma}_{id,k}) = 1$ downloads ψ and efficiently updates her witness by first computing a and b such that $a.\tilde{\sigma}_{id,k} + b.\psi = 1$ via the extended Euclidean algorithm, and then setting $w_{id,j+1} := w_{id,j}^b.irp_{i+1}^a$. This is an optimization we propose over [Camenisch and Lysyanskaya, 2002], details of which are given in Section 2.6. We note that this optimization does not violate anonymity and signature unforgeability, since even if an attacker obtains $\tilde{\sigma}_{id}$ and w_{id} of a user, the knowledge of these values are proven in zero-knowledge in Sign algorithm, and one cannot generate the SoK_3 without knowing (s, ρ_2, σ_{id}). AttrRevoke algorithm is the same as UserRevoke, where the attribute revocation handle $\tilde{\sigma}_\omega$ is used instead of $\tilde{\sigma}_{id}$, and the attribute revocation public key arp is used instead of irp.

Trace. In the Trace protocol, given a valid VABS with the tracing tag Φ, tracing authorities run the TPKE decryption [Canetti and Goldwasser, 1999] as an authenticated Byzantine Fault Tolerance [Lamport et al., 1982, Veronese et al., 2013] protocol, so that at the end the ones that follow the protocol would come to a consensus[6] in the decryption of Φ as g_1^s, and they find the associated uid in TDB via searching g_1^s. In our case, we assume at most Δ tracing authorities are malicious, resulting in at least θ honest tracing authorities always coming to the consensus on the correct signer uid. We note that "tracing soundness" of [Ghadafi, 2015] is also ensured, since Φ can only be decrypted to a single value, assuming θ of the tracing authorities joining the operation are honest.

[6] "Interactive proofs of validity of partial decryptions" method in [Canetti and Goldwasser, 1999] should be utilized to obtain the correct plaintext.

2.5 Security Proof of Our VABS

We formally prove the security of our scheme in this section based on Strong RSA, DDH, and SDDHI assumptions in the random oracle model. Our security proof takes into account only the full-fledged VABS. Regarding exclusion of the blue parts, the games and the proof show that VABS without revocation feature still has reliable traceability, anonymity, and soundness, without any change. It requires a minor change to the signature unforgeability game, as we cannot revoke ω upon issuing. Instead, the attribute is never issued by the challenger. This updated definition would still be satisfied by our solution, as it is more restrictive for the adversary. Regarding exclusion of the green parts, it is straightforward to show that we obtain the complete VABS security except for reliable traceability.

Theorem 1. *If the Strong RSA assumption [Camenisch et al., 2006, Barić and Pfitzmann, 1997] holds in the groups $\mathbb{Z}^*_{N_i}$ and $\mathbb{Z}^*_{M_i}$ of each identity provider/attribute authority i and in the group $\mathbb{Z}^*_{O_i}$ of each tracing authority i, the underlying SoK schemes are secure (i.e., they satisfy completeness, soundness, and zero-knowledge properties of zero-knowledge proofs of knowledge), $F_{g,s}(x) = g^{1/s+x}$ is a PRF with input $x \in \mathbb{Z}^*_q$, the digital signature scheme* $(\mathsf{DSKeyGen}, \mathsf{DSSign}, \mathsf{DSVerify})$ *satisfies EU-ACMA [Katz and Lindell, 2007], the TPKE scheme realizes the ideal encryption model of [Canetti and Goldwasser, 1999], $H(\cdot)$ is modeled as a random oracle, the trust assumptions on the authorities hold, i.e., the identity providers/attribute authorities issue the tokens only to deserving users and out of $\phi \geq 2\theta - 1$ tracing authorities at most $\theta - 1$ are malicious; then our VABS scheme is secure (i.e., it satisfies anonymity, reliable traceability, signature unforgeability, and soundness).*

Remark 1. *The security of the recommended SoK schemes and the pseudorandomness of the function $F_{g,s}(x) = g^{1/s+x}$ are proven to be reducible to the Strong RSA assumption and the SDDHI assumption [Camenisch et al., 2006] holding in $\langle g \rangle$. Further, the security of the TPKE scheme of [Canetti and Goldwasser, 1999] is reducible to the DDH assumption holding in $\langle g_1 \rangle$. In what follows, we provide full proof of Theorem 1.*

Lemma 1. *If the digital signature scheme* $(\mathsf{DSKeyGen}, \mathsf{DSSign}, \mathsf{DSVerify})$ *satisfies EU-*

ACMA, the underlying SoK schemes are zero-knowledge, and $F_{g,s}(x) = g^{1/s+x}$ is a pseudorandom function (PRF) for $x \in \mathbb{Z}_q^*$, and the TPKE scheme realizes the ideal encryption model of [Canetti and Goldwasser, 1999] (due to DDH assumption holding in $\langle g_1 \rangle$); then our VABS scheme achieves anonymity.

Proof. Assuming that the digital signature scheme (DSKeyGen, DSSign, DSVerify) satisfies EU-ACMA, and the underlying SoK schemes are zero-knowledge, and the threshold encryption of [Canetti and Goldwasser, 1999] realizes the ideal encryption model of [Canetti and Goldwasser, 1999]; we now reduce the anonymity of our VABS scheme to the pseudorandomness of $F_{g,s}(x) = g^{1/s+x}$. If a PPT adversary $\hat{\mathcal{A}}$ wins the anonymity game VABSAnonym with non-negligible advantage, then we can use $\hat{\mathcal{A}}$ to construct a PPT algorithm $\hat{\mathcal{B}}$ that distinguishes $F_{g,s}$ from a random function $f : \mathbb{Z}_q^* \to \langle g \rangle$ with non-negligible advantage. In the VABSAnonym game, $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ play the roles of $\mathcal{A}$ and $\mathcal{C}$, respectively. In the DistPRF game, $\hat{\mathcal{B}}$ plays the role of $\mathcal{A}$ against an honest challenger $\hat{\mathcal{C}}$. In VABSAnonym, since we assume the security of the SoKs and the realization of the ideal encryption model by the TPKE scheme, we give the control of the simulator for non-interactive proofs and ideal functionality of the TPKE scheme to $\hat{\mathcal{B}}$; i.e., $\hat{\mathcal{B}}$ simulates SoKs for $\hat{\mathcal{A}}$. Note that in the TPKE scheme of [Canetti and Goldwasser, 1999], $\hat{\mathcal{B}}$ can create a ciphertext by picking a random group element for each of its four parts, and during the decryption, if $\hat{\mathcal{B}}$ knows a plaintext m, using at least one authority under its control, it can simulate the partial decryption of that authority so that overall the decryption outputs m.

1. In VABSAnonym, $\hat{\mathcal{B}}$ picks arbitrary $h \in \langle g \rangle$ and $n, \delta \in poly(\lambda)$, and gives to $\hat{\mathcal{A}}$ the values 1^λ, n, δ, and $params = (q, \langle g \rangle, g, h)$. $\hat{\mathcal{B}}$ and $\hat{\mathcal{A}}$ then follow TraceSetup and publish the outputs as described. The time counter t is initialized as 1, and is started (Step 1 of VABSAnonym).

2. In VABSAnonym, $\hat{\mathcal{B}}$ follows Step 2 of the game with $\hat{\mathcal{A}}$ in the exact same way as an honest $\mathcal{C}$ would do.

3. In VABSAnonym, $\hat{\mathcal{B}}$ generates two user secret keys s_0 and s_1, and computes commitments to these values using the authority public keys. $\hat{\mathcal{B}}$ gives to $\hat{\mathcal{A}}$ all the commitments to s_0 and s_1. (Step 3 of VABSAnonym).

4. In VABSAnonym, a bit b is randomly picked by $\hat{\mathcal{B}}$ (normally an honest challenger picks this bit in Step 6 of the game).

5. $\hat{\mathcal{A}}$ is given access to the Sign oracles for s_0 and s_1. However, the oracle outputs are arranged by $\hat{\mathcal{B}}$ as follows. If the Sign oracle for s_b is queried with $(m', \tilde{\beta}')$, $\hat{\mathcal{B}}$ queries the oracle in DistPRF with $t'2^{\ell_{cnt}} + J_b$, obtains the oracle output S', and then generates $\sigma \leftarrow (S', \mathsf{C}_{J_b}, \varsigma', \text{the simulation of } \Phi, \text{the simulated SoKs on } m')$, where $\varsigma' \twoheadleftarrow \langle g \rangle$ is the commitment simulation, and all the SoKs and the ciphertext Φ (so that it always traces to the signer) are simulated. If $\mathsf{Sign}(\cdot, s_{\bar{b}}, \tilde{\beta}, \tilde{\Sigma}_\Omega, \tilde{AP}, t, J_{\bar{b}})$ is queried with $(m', \tilde{\beta}')$, $\hat{\mathcal{B}}$ itself picks $S' \twoheadleftarrow \langle g \rangle$ instead of querying the oracle in DistPRF, and generates the other parts of the signature in the same way as in s_b. $\hat{\mathcal{B}}$ gives the oracle outputs to $\hat{\mathcal{A}}$ (Step 4 of VABSAnonym).

6. In VABSAnonym, $\hat{\mathcal{A}}$ generates β, m_0 and m_1, when both $J_0 < n-1$ and $J_1 < n-1$. $\hat{\mathcal{A}}$ generates attribute tokens on s_0 and s_1 to prove that both users conform to β and gives those signatures to $\hat{\mathcal{B}}$ (Step 5 of VABSAnonym).

7. In DistPRF, $\hat{\mathcal{B}}$ queries the oracle with $t2^{\ell_{cnt}} + J_b$, and obtains the oracle output S_b. In VABSAnonym, $\hat{\mathcal{B}}$ sets $\sigma_0 := (S_b, \mathsf{C}_{J_b}, \varsigma_1, \text{the simulation of } \Phi, \text{the simulated SoKs on } m_0, A)$ and $\sigma_1 := (S_{\bar{b}}, \mathsf{C}_{J_{\bar{b}}} \varsigma_2, \text{the simulation of } \Phi, \text{the simulated SoKs on } m_1, A)$ where $S_{\bar{b}} \twoheadleftarrow \langle g \rangle$, $\varsigma_1, \varsigma_2 \twoheadleftarrow \langle g \rangle$ are commitment simulations, and all the SoKs are simulated (Step 6 of VABSAnonym).

8. In VABSAnonym, $\hat{\mathcal{A}}$ eventually outputs a bit b' (Step 7 of VABSAnonym). If $b = b'$, in DistPRF, $\hat{\mathcal{B}}$ outputs $F_{g,s}$. Otherwise, $\hat{\mathcal{B}}$ outputs f.

In both σ_0 and σ_1, the only values where $\hat{\mathcal{A}}$ can make the differentiation are S_b and $S_{\bar{b}}$, since the Pedersen commitment scheme is information theoretically hid-

ing, and SoKs and Φ are simulated. Moreover, if the oracle in DistPRF is the random function f, then S_b and $S_{\bar{b}}$ are also perfectly indistinguishable, which implies that $\hat{\mathcal{A}}$ would not obtain non-negligible advantage from these values. If $\hat{\mathcal{A}}$ obtains non-negligible advantage from these values, then the oracle is the pseudorandom function $F_{g,s}$. Let ϵ_1 and ϵ_2 denote the advantages of the adversaries in VABSAnonym and DistPRF, respectively. Then, we have

$$\mathsf{Pr}[\hat{\mathcal{A}} \text{ wins VABSAnonym (i.e., } b = b')] = \frac{1}{2} + \epsilon_1$$

$$\mathsf{Pr}[\hat{\mathcal{B}} \text{ outputs } F_{g,s} | \text{ oracle is } F_{g,s}] - \mathsf{Pr}[\hat{\mathcal{B}} \text{ outputs } \mathsf{F_{g,s}} | \text{ oracle is } f] = \epsilon_2$$

Via the total probability law, we can write the first equality as

$$\mathsf{Pr}[b = b' | \text{ oracle is } F_{g,s}].\mathsf{Pr}[\text{ oracle is } F_{g,s}] + \mathsf{Pr}[b = b' | \text{ oracle is } f].\mathsf{Pr}[\text{oracle is } f] = \frac{1}{2} + \epsilon_1$$

Also, since if $b = b'$, $\hat{\mathcal{B}}$ outputs $F_{g,s}$, and otherwise, it outputs f, we can convert the second equality as

$$\mathsf{Pr}[b = b' | \text{ oracle is } F_{g,s}] - \mathsf{Pr}[b = b' | \text{ oracle is } f] = \epsilon_2$$

Combining the above two resulting equations, we obtain

$$(\mathsf{Pr}[b = b' | \text{ oracle is } f] + \epsilon_2).\mathsf{Pr}[\text{oracle is } F_{g,s}] + \mathsf{Pr}[b = b' | \text{ oracle is } f].\mathsf{Pr}[\text{ oracle is } f] = \frac{1}{2} + \epsilon_1$$

Substituting $1/2$ for both $\mathsf{Pr}[\text{oracle is } F_{g,s}]$ and $\mathsf{Pr}[\text{oracle is } f]$,

$$\frac{1}{2}(\mathsf{Pr}[b = b' | \text{oracle is } f] + \epsilon_2) + \frac{1}{2}\mathsf{Pr}[b = b' | \text{oracle is } f] = \frac{1}{2} + \epsilon_1$$

$$\mathsf{Pr}[b = b' | \text{ oracle is } f] + \frac{1}{2}\epsilon_2 = \frac{1}{2} + \epsilon_1$$

If the oracle is f, $\hat{\mathcal{A}}$ can win VABSAnonym with exactly $1/2$ probability, since its views for both $b = 0$ and $b = 1$ are statistically identical. Thus, $\frac{1}{2}\epsilon_2 = \epsilon_1$ and if $\hat{\mathcal{A}}$ wins VABSAnonym with non-negligible advantage ϵ_1, then $\hat{\mathcal{B}}$'s advantage ϵ_2 in DistPRF is also non-negligible. Since ϵ_2 is negligible if the PRF is indistinguishable from random, so must ϵ_1 be, showing our VABS provides anonymity. □

Lemma 2. *If the digital signature scheme* (DSKeyGen, DSSign, DSVerify) *satisfies EU-ACMA, the Strong RSA assumption holds in the group $\mathbb{Z}^*_{O_i}$ of each tracing authority i, the TPKE scheme realizes the ideal encryption model of [Canetti and Goldwasser, 1999] (due to DDH assumption holding in $\langle g_1 \rangle$), and the underlying SoK schemes are sound; then our VABS scheme achieves reliable traceability.*

Proof. Assuming that the digital signature scheme (DSKeyGen, DSSign, DSVerify) satisfies EU-ACMA and all the underlying SoK schemes satisfies soundness and the DDH assumption holds in $\langle g_1 \rangle$, we now reduce reliable traceability of our VABS scheme to the Strong RSA assumption that should hold in all $\mathbb{Z}^*_{O_i}$ of tracing authorities. If a PPT adversary $\hat{\mathcal{A}}$ wins the reliable traceability game VABSTrace with non-negligible advantage, then we can use $\hat{\mathcal{A}}$ to construct a PPT algorithm $\hat{\mathcal{B}}$ that breaks unforgeability of the CL signature of at least one authority with non-negligible advantage. Since [Camenisch and Lysyanskaya, 2003] already showed that their signature scheme is unforgeable under the Strong RSA assumption, $\hat{\mathcal{B}}$ can be used to break the Strong RSA assumption in at least one tracing authority group. In the game VABSTrace, $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ play the roles of $\mathcal{A}$ and $\mathcal{C}$, respectively. In the CLSignForge game[7], $\hat{\mathcal{B}}$ plays the role of $\mathcal{A}$ against an honest challenger $\hat{\mathcal{C}}$. $\hat{\mathcal{B}}$ extracts SoKs of $\hat{\mathcal{A}}$ so that if $\hat{\mathcal{A}}$ needs to generate $\mathrm{SoK}[m]\{\xi : \xi \text{ satisfies } \zeta\}$, $\hat{\mathcal{B}}$ learns the witness ξ in addition to the message m and condition ζ, and only checks whether ξ satisfies ζ.

1. In CLSignForge, $\hat{\mathcal{B}}$ obtains 1^λ.

2. In VABSTrace, $\hat{\mathcal{B}}$ runs GlobalSetup on 1^λ and obtains $params = (q, \mathbb{G}, g, h)$. Also, it picks arbitrary $n, \delta \in poly(\lambda)$, and gives 1^λ, n, δ, $params$ to $\hat{\mathcal{A}}$. $\hat{\mathcal{B}}$ and $\hat{\mathcal{A}}$ then follow TraceSetup and publish the outputs as described. The time counter t is initialized as 1, and is started (Step 1 of VABSTrace).

[7]Briefly, in CLSignForge, given a security parameter λ, a CL public key p and access to the related CL signing oracle, the adversary wins by computing a message m and a signature σ_{CL} on it that gets verified by the public key p, subject to the restriction that m cannot be previously queried to the oracle.

3. In VABSTrace, $\hat{\mathcal{B}}$ follows Step 2 of the game with $\hat{\mathcal{A}}$ in the exact same way as an honest $\mathcal{C}$ would do, except for one of the tracing authorities (picked randomly by $\hat{\mathcal{B}}$) under $\hat{\mathcal{B}}$'s control. For that authority only, $\hat{\mathcal{B}}$ acts as follows. In CLSignForge, $\hat{\mathcal{B}}$ obtains the public key p. $\hat{\mathcal{B}}$ gives p to $\hat{\mathcal{A}}$ as the CL public key part of the public key of that authority (TPKE public key part is prepared as in the honest way). If $\hat{\mathcal{A}}$ asks to involve the tracing authority with public key p in a TraceIssue execution, then $\hat{\mathcal{B}}$ queries to the signing oracle in CLSignForge, receives the output and returns it to $\hat{\mathcal{A}}$. Otherwise, $\hat{\mathcal{B}}$ signs the message with the secret key of the queried authority itself.

4. In VABSTrace, $\hat{\mathcal{A}}$ eventually gives a VABS to $\hat{\mathcal{B}}$ (Step 3 of VABSTrace). If $\hat{\mathcal{A}}$ wins, $\hat{\mathcal{B}}$ extracts the SoK_{6+k} query for that VABS to obtain the CL signature σ_{CL} that $\hat{\mathcal{A}}$ used.

5. In CLSignForge, $\hat{\mathcal{B}}$ outputs σ_{CL}.

Clearly, the only way that $\hat{\mathcal{A}}$ wins VABSTrace is via forging a CL signature with non-negligible probability. Since one of the generated signatures would correspond to the authority p with non-negligible probability (i.e., at least $1/K$ where $K \in poly(\lambda)$ is the number of the tracing authorities), $\hat{\mathcal{B}}$ also wins CLSignForge with non-negligible probability. □

Lemma 3. *If the identity providers are honestly randomize the secret keys, and $H(\cdot)$ is modeled as a random oracle; then the probability that different users can combine their attribute tokens is negligible.*

Proof. In our scheme, users i and j with secret keys s_i and s_j can combine their attribute tokens for the attributes ω_k and ω_l, only in three different cases: (1) $s_i = s_j$, (2) the equality $s_i = s_j + H(\omega_l)$ holds for some $s_i \neq s_j$ and the user attribute authority is permitted to issue attribute ω_l, and (3) the equality $s_i + H(\omega_k) = s_j + H(\omega_l)$ holds for some $s_i \neq s_j$ and some $\omega_k \neq \omega_l$. Our scheme does not allow any collusion in any other case, since CL signatures are only generated on these values and do not permit combining. Therefore, it will be sufficient to show that

the probability that there exists two different users with secret keys s_i and s_j, and two different attributes ω_k and ω_l, such that at least one of the cases (1), (2), and (3) holds is negligible. Let Γ, Ψ, η and φ denote the set of the secret keys s in the system, the set of attributes ω in the system, the number of elements of Γ, and the number of elements of Ψ, respectively. Note that the number of users and attributes can be considered as bounded by a polynomial $poly(\lambda)$. If the identity providers are honest, and randomize the user secret keys, each user key s_i, s_j, and their difference $s_i - s_j$ is statistically identical to picking randomly from $\mathbb{Z}_q$. Also, as $H(\cdot)$ is modeled as a random oracle, each randomized attribute $H(\omega_k)$, $H(\omega_l)$, and their difference $H(\omega_k) - H(\omega_l)$ is statistically identical to picking randomly from $\mathbb{Z}_q$. Since the number of elements in $\mathbb{Z}_q$ is q, for cases (1), (2), and (3), we calculate

$$\mathsf{Pr}[\exists s_i, s_j \in \Gamma \text{ s.t. } s_i = s_j, i \neq j] = 1 - \prod_{x=2}^{\eta} \left(1 - \frac{x-1}{q}\right) = O(\eta^2/q),$$

$$\mathsf{Pr}[\exists s_i, s_j \in \Gamma, \exists H(\omega) \in \Psi \text{ s.t. } s_i = s_j + H(\omega), i \neq j] \leq 1 - \left(1 - \frac{\varphi}{q}\right)^{2\cdot\binom{\eta}{2}} = O(\eta^2\varphi/q),$$

$$\mathsf{Pr}[\exists s_i, s_j \in \Gamma, \exists H(\omega_k), H(\omega_l) \in \Psi \text{ s.t. } s_i + H(\omega_k) =$$

$$s_j + H(\omega_l), i \neq j, \omega_k \neq \omega_l] \leq 1 - \left(1 - \frac{\binom{\varphi}{2}}{q-1}\right)^{2\cdot\binom{\eta}{2}} = O(\eta^2\varphi^2/q).$$

From the above, we calculate the probability ϵ of at least one of the above three events occuring as at most the addition of them, i.e., $\epsilon = O(\eta^2/q) + O(\eta^2\varphi/q) + O(\eta^2\varphi^2/q) = O(\eta^2\varphi^2/q)$. Since $q \in \Theta(2^\lambda)$ and $\eta, \varphi \in poly(\lambda)$, we deduce that ϵ is a negligible function of λ. □

Lemma 4. *If the Strong RSA assumption [Camenisch et al., 2006, Barić and Pfitzmann, 1997] holds in the groups $\mathbb{Z}^*_{N_i}$ and $\mathbb{Z}^*_{M_i}$ of each identity provider/attribute authority i, $\log_g h$ is not deducible by any party (due to the DDH assumption in $\langle g \rangle$), the underlying SoK schemes are sound, and the underlying hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ is modeled as a random oracle; then our VABS scheme achieves signature unforgeability.*

Proof. Assuming that $\log_g h$ is not deducible by any party, and that all the underlying SoK schemes satisfy soundness, and that H is modeled as a random ora-

cle, we now reduce signature unforgeability of our VABS scheme to the Strong RSA assumption that should hold in all $\mathbb{Z}^*_{N_i}$ and $\mathbb{Z}^*_{M_i}$ of identity providers and attribute authorities. We also assume that non-revocation witnesses are unforgeable due to the Strong RSA assumption [Camenisch and Lysyanskaya, 2002]. If a PPT adversary $\hat{\mathcal{A}}$ wins the signature unforgeability game VABSForge with non-negligible advantage, then we can utilize $\hat{\mathcal{A}}$ to construct a PPT algorithm $\hat{\mathcal{B}}$ that breaks the unforgeability of CL signature of at least one authority non-negligible advantage. Since forging CL signatures imply breaking the Strong RSA assumption as shown in [Camenisch and Lysyanskaya, 2003], $\hat{\mathcal{B}}$ can trivally be utilized for constructing a PPT algorithm that breaks the Strong RSA assumption in at least one attribute authority group. In the game VABSForge, $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ play the roles of $\mathcal{A}$ and $\mathcal{C}$, respectively. In the CLSignForge game (the CL signature equivalent of the conventional EU-ACMA game for signatures as described as Sig-forge in [Katz and Lindell, 2007]), $\hat{\mathcal{B}}$ plays the role of $\mathcal{A}$ against an honest challenger $\hat{\mathcal{C}}$. Briefly, in CLSignForge, given a security parameter λ, a CL public key p and access to the related CL signing oracle, the adversary wins by computing a message m and a signature σ_{CL} on it that gets verified by the public key p, subject to the restriction that m cannot be previously queried to the oracle.

We start by replacing H with a random oracle under control of $\hat{\mathcal{B}}$ and accessible by $\hat{\mathcal{A}}$. Also, $\hat{\mathcal{B}}$ extracts SoKs of $\hat{\mathcal{A}}$ such that if $\hat{\mathcal{A}}$ needs to generate $\mathrm{SoK}[m]\{\xi : \xi \text{ satisfies } \zeta\}$, it needs to give the input a message m, the information ξ, and condition ζ to $\hat{\mathcal{B}}$, who only checks whether ξ satisfies ζ.

1. In CLSignForge, $\hat{\mathcal{B}}$ obtains 1^λ.

2. In VABSForge, $\hat{\mathcal{B}}$ runs GlobalSetup on 1^λ and obtains $params = (q, \mathbb{G}, g, h)$. Also, it picks arbitrary $n, \delta \in poly(\lambda)$, and gives 1^λ, n, δ, and $params$ to $\hat{\mathcal{A}}$. $\hat{\mathcal{B}}$ and $\hat{\mathcal{A}}$ then follow TraceSetup and publish the outputs as described. The time counter t is initialized as 1, and is started (Step 1 of VABSForge).

3. In VABSForge, $\hat{\mathcal{B}}$ follows Step 2 of the game with $\hat{\mathcal{A}}$ in the exact same way as an honest $\mathcal{C}$ would do, except for one of the attribute authorities (picked

randomly by $\hat{\mathcal{B}}$) that can issue ω under $\hat{\mathcal{B}}$'s control generated after $\hat{\mathcal{A}}$'s request. Note that according to the game definition $\hat{\mathcal{A}}$ must request at least one attribute authority (see Step 5 of VABSForge) for verification of ω. For that authority only, $\hat{\mathcal{B}}$ acts as the Steps 6 and 8 of this proof below. If there are multiple such authorities, $\hat{\mathcal{B}}$ picks one randomly reducing its chance only inverse polynomially.

4. $\hat{\mathcal{A}}$ is allowed to corrupt all ϕ tracing authorities (Step 3 of VABSForge.

5. In VABSForge, $\hat{\mathcal{A}}$ returns an attribute ω (that is not queried to the authorities under $\hat{\mathcal{B}}$'s control for users under $\hat{\mathcal{A}}$'s control (Step 4 of VABSForge).

6. In CLSignForge, $\hat{\mathcal{B}}$ obtains the public key p. $\hat{\mathcal{B}}$ gives to $\hat{\mathcal{A}}$ as one of the public keys of authorities under $\hat{\mathcal{B}}$'s control (Step 5 of VABSForge).

7. In VABSForge, for each s_i queried for the attribute ω by $\hat{\mathcal{A}}$ that verifies in the checks of the previous step, $\hat{\mathcal{B}}$ itself picks a random value ϱ, sets it as the query output to the random oracle for ω, and sets $m_i = s_i + \varrho$.

8. In VABSForge, if $\hat{\mathcal{A}}$ queries the authority oracle for p, then $\hat{\mathcal{B}}$ queries m_i to the signing oracle in CLSignForge, receives and gives the output to $\hat{\mathcal{A}}$. Otherwise, $\hat{\mathcal{B}}$ signs the message with the secret key of queried authority itself.

9. In VABSForge, $\hat{\mathcal{A}}$ eventually outputs $(m, \beta \wedge \omega, \sigma)$ to $\hat{\mathcal{B}}$ (Step 6 of VABSForge). If $\hat{\mathcal{A}}$ wins, $\hat{\mathcal{B}}$ extracts the SoK_{3+i} query for the winning μ, where i is the index of ω. $\hat{\mathcal{B}}$ then parses the related SoK query from $\hat{\mathcal{A}}$ and using the SoK extractor $\hat{\mathcal{B}}$ obtains the CL signature σ_{CL} that $\hat{\mathcal{A}}$ utilized.

10. In CLSignForge, $\hat{\mathcal{B}}$ outputs σ_{CL}.

As shown in [Camenisch et al., 2006], $\hat{\mathcal{A}}$ cannot achieve a forgery in commitments with non-negligible probability, since it cannot deduce $\log_g h$. Also, since

SoKs are sound, $\hat{\mathcal{A}}$ cannot prove with non-negligible probability without knowing the CL signatures. Since the non-revocation witnesses are also assumed to be unforgeable, the only way that $\hat{\mathcal{A}}$ wins VABSForge is via forging a CL signature. Since the generated signature would correspond to the authority p with non-negligible probability (i.e., at least $1/K$ where $K \in poly(\lambda)$ is the number of the authorities), $\hat{\mathcal{B}}$ also wins CLSignForge with non-negligible probability. □

Lemma 5. *If Strong RSA assumption holds in the group $\mathbb{Z}^*_{N_i}$ of each identity provider i, and the underlying SoK schemes are sound, then our VABS scheme achieves soundness.*

Proof Intuition. The soundness proof of [Camenisch et al., 2006] covers our soundness proof, since we achieve this property by limiting the use of the user secret key s via S, C_s, C_J and three SoKs ($\mathrm{SoK}_1, \ldots, \mathrm{SoK}_3$) related to these serial number and commitments. The serial number S, and commitments C_s and C_J are structured exactly the same as in [Camenisch et al., 2006]. The minor difference is that instead of ZKPoK protocols of [Camenisch et al., 2006], the user proves the knowledge of the same information via SoKs (which provides the same security guarantees in the random oracle mode) on the signed messages. □

2.6 Efficiency

Observe that GlobalSetup, TraceSetup, IdPJoin, AuthJoin, UserJoin, TraceIssue, AttrIssue, AttrRevoke, and Trace operations take place infrequently, and require a constant number of public key operations. Therefore, the main efficiency concern is the more frequently used operations, Sign and Verify. Asymptotically, for an attribute policy β without any numeric attributes, Sign algorithm's computational cost and the signature size is proportional to the included attributes, and hence is $O(|\beta| + \theta)$ SoK costs. Similarly, Verify algorithm requires the verifier to execute $O(|\beta| + \theta)$ SoKs. Regarding numeric attributes, complexities of Sign and Verify are increased by the logarithm of the range of each such attribute.

Implementation Results. We implemented our scheme as proof-of-concept using the CL signature and zero-knowledge proof primitives implemented in the

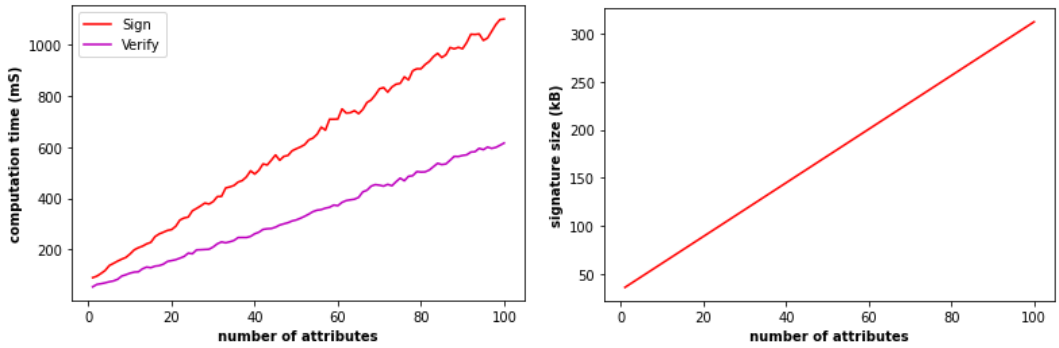
	GlobalSetup	TraceSetup	IdPJoin	AuthJoin	UserJoin	TraceIssue	AttrIssue
Computation Time	69.2	$507.6 + 72.4 \times \phi$	519.4	593.8	11.3, 21.3	13.5, 48.1	21.2, 9.9
Bandwidth	N/A	N/A	N/A	N/A	0.6, 3.5	6.3, 0.5	5.7, 0.5

Table 2.3: The computation times (in milliseconds) and bandwidth uses (in kilobytes) of GlobalSetup, TraceSetup, IdPJoin, AuthJoin, UserJoin, TraceIssue, and AttrIssue of our VABS scheme. The boxes with two values (separated by comma) are actually protocol results with the first value belonging to the user and the second value belonging to the authority. The RSA and prime-order group modulus lengths are set as 2048 bits, and SHA256 and DSA are used as the random oracle and the digital signature algorithm, respectively. N/A denotes "not applicable". GlobalSetup and TraceSetup run locally with one authority. We highlight that in an application these values are unlikely to be significant, as they will occur rarely.

C++ Cashlib cryptographic library[8] [Meiklejohn et al., 2010]. Using our implementation, we conducted efficiency tests on a computer with Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz and 16GB RAM. The RSA and prime-order group modulus lengths are set as 2048 bits, and SHA256 and DSA are used for instantiating the random oracle and the digital signature algorithm, respectively. Also, we use the hash-and-sign paradigm. Therefore, we conducted our tests on constant message size of 256 bits. Our tests were repeated 10 times and we report average numbers. We tested GlobalSetup and TraceSetup locally. The latter is also conducted with a dealer for a varying number ϕ of trace authorities. In practice they will take place only once in a distributed fashion, yet the complexity should differ only slightly due to the communication. Similarly, IdPJoin and AuthJoin are run only once per authority. Table 2.3 shows the average results for setup, joining, and issuing operations.

Figures 2.4a and 2.4b and Table 2.3 show the results of the implementation with respect to various attribute policy sizes (the number of attributes in the Boolean AND policy) with VABS traceability. The results are provided as the

[8] `https://github.com/brownie/cashlib`

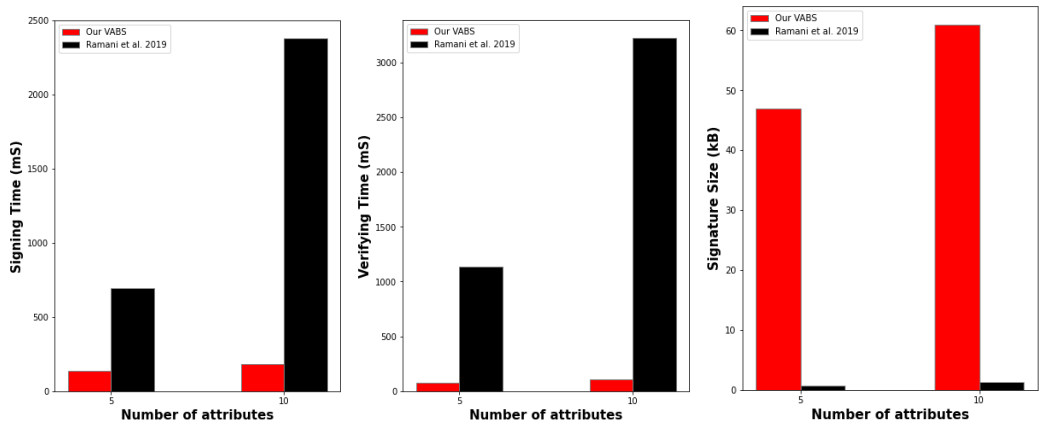


(a) Computation times of Sign and Verify (b) VABS size

Figure 2.4: The implementation results of Sign and Verify of our VABS scheme for various attribute policy sizes. The RSA and prime-order group modulus lengths are set as 2048 bits, and SHA256 and DSA are used as the random oracle and the digital signature algorithm, respectively.

mean values of 10 experiments for each value; and our observation is a very small standard deviation. On average, our Sign algorithm implementation takes 10.2 ms per attribute that needs to be proven, on top of a 80.1 ms fixed cost, with traceability. Also, on average, the length of the generated ABS is 2.8 KB per attribute, on top of the 32.8 KB fixed size, with traceability. Further, on average, our Verify algorithm requires 5.6 ms per attribute, in addition to the 48.6 ms fixed cost, with traceability. The additional costs of traceability are 66.6 ms on Sign, 20.0 KB on VABS size, and 24.3 ms on Verify, plus the costs for proving CL signature from each tracing authority (the same as the additional costs per attribute).

Regarding efficiency comparison, in Figure 2.5, we compare our scheme with the implementation results given in [Ramani et al., 2019]. We can see that in terms of computation times, our scheme is more efficient, whereas [Ramani et al., 2019] results in smaller signatures. We stress that our VABS has multiple features added on ABS, while [Ramani et al., 2019] is an ABS scheme designed for Named Data Networking without traceacility, revocation, etc. Regarding the other MA-ABS schemes, a direct comparison would be misleading, since they are based on span programs corresponding to attribute policies, and their computation times and



(a) Signing (b) Verification (c) Size

Figure 2.5: The computation times and signature size comparison of our VABS without tracing and revocation and [Ramani et al., 2019]. Note that our results are obtained on Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz and 16GB RAM, whereas the results of [Ramani et al., 2019] were obtained on Intel T2300 1.66GHz, 2.4 GB RAM.

signature sizes depend on the number of columns and widths of span programs, while ours depend on the number of attributes proven. Moreover, we note that these schemes did not provide implementation results. We could not find any open-source ABS implementation either. In contrast, we shall release our code as open-source upon acceptance.

Bulk Update of Revocation Accumulators. We can optimize revocation due to the observation that in "adding or deleting several values at once" recommended in [Camenisch and Lysyanskaya, 2002], the costly operation needs to be done only if $gcd(\psi, \tilde{\sigma}_{id,k}) \neq 1$, which may occur in case ψ is a multiple of $\tilde{\sigma}_{id,k}$ since $\tilde{\sigma}_{id,k}$ is prime. Let us approximate to the average frequency F of the costly operation to see the efficiency improvement. Let M_j be $\Upsilon(\lambda)$ bits where $\Upsilon(\cdot)$ is a polynomial, (for ease of calculation) ψ be randomly picked from $[0, 2^{\Upsilon(\lambda)})$, and each $\tilde{\sigma}_{id,k}$ be a prime in the range $[2, 2^{\Upsilon(\lambda)})$. In this range there exists roughly $\vartheta = 2^{\Upsilon(\lambda)} / \ln 2^{\Upsilon(\lambda)}$ prime numbers, and the i-th prime number can be approxi-

mated as $i \ln i$. Hence,

$$F \approx \frac{1}{\vartheta}\left(\frac{2^{\Upsilon(\lambda)}/2}{2^{\Upsilon(\lambda)}} + \sum_{i=2}^{\vartheta} \frac{2^{\Upsilon(\lambda)}/(i \ln i)}{2^{\Upsilon(\lambda)}}\right) < \frac{1}{\vartheta}\sum_{i=1}^{\vartheta}\frac{1}{i} < \frac{\ln \vartheta + 1}{\vartheta} < \frac{(\Upsilon(\lambda)\ln 2)^2}{2^{\Upsilon(\lambda)}} = \mathsf{n}(\lambda)$$

where we have applied the Maclaurin-Cauchy test on the harmonic series. Thus, the check for $gcd(\psi, g\tilde{id}_k) = 1$ can be omitted by the users.

2.7 Related Work

Attribute based signature (ABS) schemes [Shahandashti and Safavi-Naini, 2009, Maji et al., 2011, Cao et al., 2012, Okamoto and Takashima, 2013, El Kaafarani et al., 2014b, El Kaafarani et al., 2014a, Ghadafi, 2015, Hampiholi et al., 2015, Urquidi et al., 2016, Sakai et al., 2016, El Kaafarani and Ghadafi, 2017, Drăgan et al., 2018, Sakai et al., 2018, Ramani et al., 2019, Huang et al., 2021] are non-interactive signature solutions for anonymous attribute policy proving. An early ABS proposal [Maji et al., 2011] enabled multi-authority attribute issuing, but required a single master authority for operation. [Okamoto and Takashima, 2013] allowed decentralization by eliminating the need for a single master authority. Further, [El Kaafarani et al., 2014b] enabled a single tracing authority. [Ghadafi, 2015] improved tracing notions together with schemes satisfying them, but still the tracing depends on a single tracing authority. Therefore, the existing multi-authority schemes of [Cao et al., 2012, Okamoto and Takashima, 2013, El Kaafarani et al., 2014b, Ghadafi, 2015] lack our mentioned goals for VABS as shown in Table 2.1. The techniques provided in this work can be combined with [Sakai et al., 2016] to achieve revocation or threshold traceability. However, [Sakai et al., 2016] does not provide usage limitation, which is hard to achieve on top. There exist some revocation techniques [Tsang et al., 2007, Nguyen, 2005] that can possibly be applied to an ABS scheme, however achieving other VABS requirements are non-trivial. Some other ABS works worth mentioning include [El Kaafarani et al., 2014a, El Kaafarani and Ghadafi, 2017] that allow a verifier to link the ABSs signed by the same signer. Although this notion seems useful for usage limitation, our work is different by not linking for limitation and preserving the signer's

anonymity. [Urquidi et al., 2016] allows an entity with a special key to link the ABSs signed by the same signer. It is different than traceability by allowing only linking but not detecting the signer. [Drăgan et al., 2018, Gardham and Manulis, 2022] enables authorities to delegate attribute issuing tasks. [Huang et al., 2021] is also an interesting work that deals with secure outsourcing of ABS signing to the cloud.

Anonymous credential (AC) schemes [Camenisch et al., 2006, Camenisch and Groß, 2012, Garman et al., 2014, Bichsel et al., 2014, Lueks et al., 2015, Derler et al., 2015, Camenisch et al., 2016, Fuchsbauer et al., 2019, Sanders, 2020] are developed for proving attributes while keeping the anonymity of the user. [Camenisch et al., 2006, Camenisch et al., 2016] provide n-times unlinkability, which is useful for a VABS construction. Note that although by definition AC schemes are not expected to be non-interactive, both [Camenisch et al., 2006] and [Camenisch et al., 2016] can be converted to a non-interactive version via the Fiat-Shamir transformation [Fiat and Shamir, 1987]. Although some techniques provide attribute policy signing and collusion resistance [Fuchsbauer et al., 2019, Sanders, 2020], they cannot be trivially used for ABS purposes. In our solution, we build on top of [Camenisch et al., 2006] and improve it, in particular, by adding collusion resistance and traceability to obtain VABS. However, we could not use [Camenisch et al., 2016] for this purpose, as it allows the authorities to obtain the secret keys of the users. The k-times anonymous authentication scheme of [Au et al., 2013] also provides a similar usage limiting functionality, yet it lacks revocation feature as in [Camenisch et al., 2006].

Functional credential schemes. The functional credential scheme of [Deuber et al., 2018] can be utilized for anonymously proving conformity to an attribute policy to the third parties. Unfortunately, the number of authentication attempts in this scheme cannot be bounded by a fixed value for limited use. Moreover, there seems no effective way of multi-authority attribute issuing.

Group and ring signatures. Group signatures [Camenisch and Groth, 2005, Bellare et al., 2005, Yang et al., 2011, Slamanig et al., 2014, Hwang et al.,

2015, Ghadafi, 2014, Nisansala et al., 2017, Emura et al., 2017, Camenisch et al., 2020] are non-interactive constructions for proving that the signer of a message belongs to some group (sharing a particular attribute). In particular, revocable [Nisansala et al., 2017, Emura et al., 2017], traceable [Chow, 2009], or distributed traceable [Ghadafi, 2014], or fully dynamic model of [Bootle et al., 2016] may seem useful in construction of a VABS. On the other hand, the use of keys in those schemes cannot easily be bounded by a fixed value. Moreover, they allow group managers to identify the signers. Ring signatures [Rivest et al., 2006, Wang et al., 2011, Liu and Wong, 2005] also provide non-interactive anonymous self-proving. However, they require the signer of the message to know all the group members' public keys and use them in the signing algorithm, resulting in computation on the order of the number of members.

2.8 Discussion

Our VABS scheme achieves usage limitation, revocability, threshold traceability, and decentralization in one scheme by extending anonymous credential scheme of [Camenisch et al., 2006]. It can be argued that such a scheme can be obtained by similar tweaks on another anonymous credential scheme in a black-box manner. However, anonymous credential schemes are not necessarily obtained by the same structure as [Camenisch et al., 2006], i.e. combination of a blind signature scheme and zero-knowledge proof scheme of authority signature. Yet, our methods could be applied to a blind signature with a generic proof system such as Groth-Sahai proofs [Groth and Sahai, 2012]. Therefore, we leave obtaining an ABS scheme from anonymous credential schemes or blind signature schemes in a black-box manner as a future work. Further, the line [Drăgan et al., 2018, Gardham and Manulis, 2022] of ABS schemes for delegating issuing of attributes in a hierarchical manner is also interesting. In particular, [Gardham and Manulis, 2022] achieves revocability feature. We also leave as a future work to obtain other features of our VABS scheme in an hierarchical ABS or, alternatively, to extend our VABS scheme for delegation of attribute issuing. Authority hiding feature for

allowing multiple authorities to issue a given attribute is also another interesting extension. This would increase user anonymity in cases such as a "university student" attribute is good enough instead of "Harvard University student". While it is trivial to obtain this in our scheme via OR proofs, a solution sublinear to the number of authorities is left as a future work.

Chapter 3

ANONYMOUS, ATTRIBUTE BASED, DECENTRALIZED, SECURE, AND FAIR E-DONATION

3.1 Introduction

Since the first secure e-cash scheme was proposed by Chaum [Chaum, 1982], digital currencies have been a center of attention in applied cryptography. One development of practical importance has been the invention of Bitcoin [Nakamoto, 2008], which combines the ideas of conventional e-cash, proof of work [Dwork and Naor, 1993], distributed systems, and game theory for a decentralized electronic payment scheme. The emergence of Bitcoin has not only contributed to state-of-the-art cryptocurrency schemes, but also brought to attention the blockchain, which is a decentralized and unalterable public ledger. Informally, a public ledger is an append-only bulletin board, where new data can be added only if it satisfies a predetermined criterion. The proposed Bitcoin and blockchain mechanism depends on only proof of work (PoW) [Dwork and Naor, 1993] instead of requiring a trusted third party for maintenance. PoW can be briefly described as a race among blockchain miners for computing brute-force solutions to a cryptographic puzzle with ever-changing parameters. Recent blockchain and Bitcoin research has focused on novel applications [Kumaresan and Bentov, 2014, Bentov and Kumaresan, 2014, Kumaresan et al., 2016, Andrychowicz et al., 2016, Kiayias et al., 2016c, Roby, 2017, Blass and Kerschbaum, 2017, Axon and Goldsmith, 2017], improved security [Eyal and Sirer, 2018, Kiayias et al., 2016a, Garay et al., 2015, Garay et al., 2017], and enhanced privacy [Miers et al., 2013, Sasson et al., 2014]. In this chapter, we focus on the novel problem of anonymous decentralized online donations as a public ledger application.

The context that we consider is similar to the "helping a stranger" described in CAF 2018 World Giving Index [CAF, 2018]. There, international charity organizations (ICO) match the donor and recipients based on the former's attribute choices. The size of the charity organizations is usually underestimated. In fact, they have a grand sector of its own with numbers as high as more than 180,000 organizations only in United Kingdom [Breeze, 2013] and more than 2 out of 3 people donate in countries like Indonesia, Australia, New Zealand, America, Ireland, United Kingdom [CAF, 2018]. We acknowledge this high demand for charity, and differentiate our work from this traditional setting by being *decentralized*, i.e., by eliminating ICOs to carry out the donation process. Our work enables donors to choose recipients based on their attributes in one compact system, providing fairness[1] of donation distributions by allowing each recipient to obtain a limited amount in each time period.

In an online donation scheme, a donor (Alice) chooses a recipient (Bob) based on his attributes without meeting him in person. Alice picks an attribute policy (e.g., the recipient should be "a high school student" AND "a person whose family's monthly income is lower than $750 per month"), and publishes it in a publicly visible venue (the Ledger) along with the donation amount. She might go offline afterwards. Assume that Bob meets this criteria, and has obtained attribute tokens from the authorities (e.g., student certificate from his school, and low income certificate from the ministry). Upon seeing this donation on the Ledger, Bob proves his conformity to this attribute policy via the obtained attribute tokens to claim the donation. Figure 3.1 shows the flow of the online donation scheme. The identities of all the parties involved remain anonymous and their actions remain unlinkable during the whole process. Further, the protocol ensures that Bob will not be able to claim more donations (even if he satisfies their policies) than a publicly set amount from the system in a time period for fairness to the other people. The system also ensures that someone (who does not meet the attribute policy that Alice picked), cannot claim any money from her donation. We list the

[1]For a further discussion on fairness, see Section 3.7.

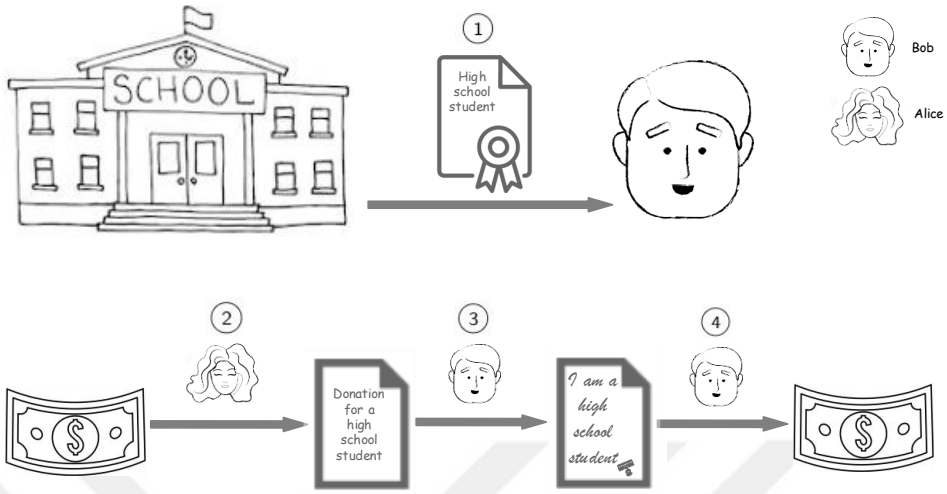


Figure 3.1: The flow of the online donation scheme that we propose. ① Bob obtains his attribute token from an attribute authority. ② Alice publishes the donation and the attribute policy on a publicly seen medium. She might go offline afterwards. ③ Upon seeing the donation, Bob claims the donation by showing the proof of his conformity to the attribute policy that he generated via the attribute token. ④ Bob then spends the donation amount, according to his wish.

aspects that an e-donation scheme needs to satisfy as follows:

- **donor privacy:** it should prevent the leakage of Alice's identity and linkage between multiple donations by her.
- **recipient privacy:** it should prevent linking between donations that Bob has received and leakage about his identity.
- **fair distribution of donations:** it should not allow Bob to receive more than a publicly determined amount of donation from the system in a given time period for fairness to other potential recipients.
- **decentralization:** it should not require trusted third parties (e.g., charity organizations) for the donation process.[2]

[2]We observe that trusted authorities are inevitable for checking recipient attributes (e.g., being

- **always-on:** it should not require Alice and Bob to remain and interact online, in particular, it should allow Alice to go offline after sending her donation money and the recipient attribute policy.

- **security:** it should protect the overall balance of the involved parties.

- **scalability:** it should be supporting multiple attribute authorities for issuing attributes to multiple recipients and multiple donors, and Bob should be able to receive a donation from any geographical location as long as he satisfy its policy.

- **accountability:** donations should be binding such that Bob should indeed receive a donation if he proves his conformity to the associated policy.[3]

- **revocability:** it should allow revocation of attributes or users.

We note that some of these aspects may not be necessary in all cases. For example, some donors may choose not to be anonymous or may like to see the recipients of their donations. In our solution, we choose to provide the most capable and privacy-preserving solution that we could, and some aspects of our solution is presented in a modular way such that they can be removed for obtaining a more efficient but less capable solution. Table 3.1 provides a comparison of various possible solutions with respect to our e-donation requirements. We highlight that the traditional ICOs can only provide donor and recipient privacy in a limited way as they do learn the identities of the parties. Also, fair distribution of donations in this scheme is limited, as recipients may obtain donations from various ICOs. Furthermore, their security, being always-on, scalability, accountability, and revocability depend on the honesty of a single organization for all

a student), but they are not required for monetary transactions or for each donation (they are used only for issuing attributes per user not per transaction).

[3]Without this property, the scheme would allow an unethical opportunity of data collection via fake donations, without the attacker losing any money.

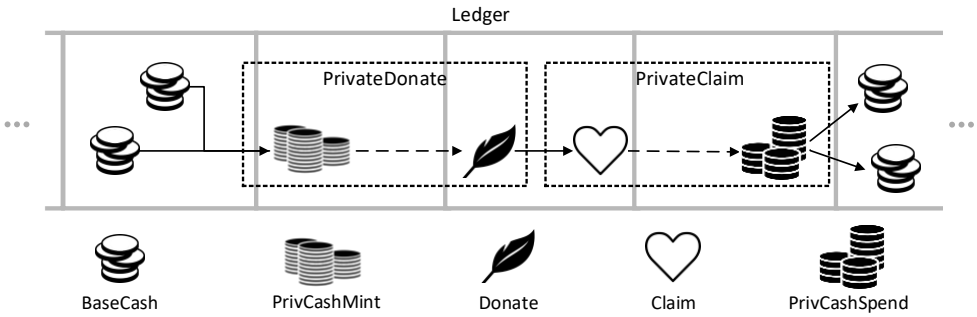


Figure 3.2: The flow of transactions in our e-donation scheme. The PrivCashMint and Donate transactions are made by the donor (together called PrivateDonate), while the Claim and PrivCashSpend transactions are made by the recipient of the donation (together called PrivateClaim). The BaseCash tuples referenced by the PrivCashMint transaction (leftmost ones) belong to the donor, whereas the output of the PrivCashSpend transaction belongs to the final payee(s). The straight arrows represent direct referencing of the transaction (linkable), while the dashed arrows represent the use of the outputs (unlinkable). We note that the block boxes are placed arbitrarily.

these tasks to be carried out. The motivation behind our proposed profile relies on the report of [Breeze, 2013], showing that the donor behavior tends to be simplistic with few attribute choices in recipients and caring more about fast delivery and ease of the donation process than finding the best fitting recipient.

System Model. In our framework, in each state or region there is a top level authority *identity provider* (e.g., Census Bureau or State Authority) that user interacts when he joins the system to generate his user token. Thus, upon joining the system, each user has an identity attribute (e.g., national identity number or social security number). This is required for preventing Sybil attacks [Schreiberr, 1973]. There are also attribute authorities for issuing attributes of users. An attribute authority can provide tokens for various attributes, and an attribute can be issued by more than one authority. We highlight that the system should prevent different users from combining their attributes. For a further discussion on possible authority architecture improvements, see Section 3.7.

We trust identity providers and attribute authorities (e.g., school districts) for honest issuing of user and attribute tokens by checking identities and eligibility of the users for attributes (e.g., high school student). The only malicious actions that authorities can take are issuing tokens to the undeserving users and revealing the identity of the users that applied to them for tokens, which can happen in any donation system. Yet, none of the authorities can donate or claim on behalf of a user, nor can they carry out the whole donation process as an ICO. They do not have monetary transaction abilities, and are not involved in each donation; rather, they are required only once per person while obtaining the attribute. Even the authorities collectively cannot identify a user from her transactions. Other than those potential malicious actions, no authority can deduce a user's secret key or sign on behalf her. Besides, even the authorities cannot identify a user from her signature.

Failed Approaches. One might imagine a trivial online solution to the secure donation problem as follows. First, the donor publishes the attribute policy that she prefers the recipient to conform to. Next, a potential recipient and the donor together run an anonymous credential protocol (e.g., [Camenisch and Lysyanskaya, 2003]), where the recipient proves his eligibility to the donor. Then, the donor directly pays the donation money to the recipient using an anonymous e-cash scheme to preserve privacy. However, although this scheme seems applicable, it can hardly satisfy some of the above-mentioned requirements, namely, online decentralization (due to the need for a trusted third party in the e-cash scheme), scalability (since it is not easy for every donor to always have secure access to all valid authority certificates and potential recipients), revocability (again, since secure access to all revocation lists or keys may be hard for every donor), and accountability (since once the recipient proves his conformity, the donor can still choose not to send the e-cash). Moreover, this trivial solution requires the recipients and the donors to remain online and to interact directly, not allowing decentralization in its full sense as described above. Lastly, in such a setting, enforcing fair distribution seems impossible. In our solution, we employ a dis-

tributed ledger together with proper novel cryptographic tools to deal with these issues, and provide the first definition and instantiation for the anonymous fair e-donation paradigm.

The anonymous decentralized e-cash schemes [Miers et al., 2013, Sasson et al., 2014] cannot be directly applied for the e-donation purpose, as they do not provide a mechanism for anonymous attribute-based money transfer between donors and recipients who are non-privy to each other. Also, it is non-trivial to directly combine these with the existing anonymous attribute proving techniques (e.g., anonymous credentials [Camenisch and Lysyanskaya, 2003, Camenisch et al., 2006, Camenisch and Groß, 2012, Garman et al., 2014, Lueks et al., 2015, Derler et al., 2015, Camenisch et al., 2016]), attribute based signatures [Maji et al., 2011, Cao et al., 2012, Okamoto and Takashima, 2013, El Kaafarani et al., 2014b, Hampiholi et al., 2015, Urquidi et al., 2016], group signatures [Camenisch and Groth, 2005, Bellare et al., 2005, Yang et al., 2011, Slamanig et al., 2014, Hwang et al., 2015, Nisansala et al., 2017, Emura et al., 2017]), while satisfying the above-mentioned requirements. We note that even letting ICOs use anonymous cryptocurrencies (e.g., Zerocash [Sasson et al., 2014]) does not help with all of our e-donation requirements. While such a trivial scheme may increase donor and recipient privacy, it can hardly distribute donations fairly. Further, the above-mentioned shortcomings of ICOs still prevail.

An existing private smart contract scheme (e.g., Hawk [Kosba et al., 2016], Arbitrum [Kalodner et al., 2018], Zether [Bünz et al., 2019], ZEXE [Bowe et al., 2020], and Enigma [Eni, 2019]) would also not solve the online donation problem itself, since it does not achieve fair distribution of donations. We show that combining a decentralized e-cash scheme or a private contract scheme with an attribute based signature scheme (e.g, VABS of [Biçer and Küpçü, 2019]) would achieve e-donation.

Overview of Our Techniques. In this work, we employ a public ledger for decentralization and accountability. We highlight that according to the flowchart presented by NIST [Yaga et al., 2018, Fig. 6], the use of blockchain in our ap-

plication is appropriate. We highlight that although e-donation fits better to blockchain-based ledger applications, there is no limitation for using it with conventional e-cash schemes, as we abstract out the ledger implementation. The ledger can be maintained by a trusted party (e.g., the issuer of the e-cash), and all its security guarantees still hold.

Our scheme is based on three cryptographic building blocks: an underlying BaseCash scheme (i.e., our abstraction of decentralized e-cash such as Bitcoin [Nakamoto, 2008]) that provides the amount of donations, an anonymized PrivCash scheme (i.e., our abstraction of decentralized anonymous e-cash such as Zerocash [Sasson et al., 2014]) and an attribute-based signature scheme called VABS [Biçer and Küpçü, 2019] that provides usage limitation, revocability, and decentralization features. PrivCash scheme basically has two operations, PrivCashMint for minting PrivCash (e.g., in Zerocash a Mint operation for converting the amounts in t-addresses into the ones z-addresses) and PrivCashSpend for spending minted PrivCash (e.g., in Zerocash a Pour operation by converting the amounts in z-addresses into the ones in t-addresses). A PrivateDonate transaction made by the donor consists of two main actions: one for minting some PrivCash (i.e., PrivCashMint) and one for sending it to some attribute policy (i.e., Donate) as a spending transaction. Similarly, a PrivateClaim transaction made by the recipient consists of two main actions: one for claiming a previous donation (i.e., Claim) signed with a VABS for proving attribute policy conformity and one for spending it to a separate payee (i.e., PrivCashSpend). Once the donor makes a donation to an attribute policy, the donation is fixed on the Ledger as a PrivateDonate transaction tuple, and the donor no longer needs to be involved in its delivery. The donations on the Ledger are delivered in a first-come-first-served basis. Any qualified recipient (adhering to the attribute policy) can claim a donation with a PrivateClaim transaction tuple, as long as the donation (partially) remains unclaimed on the Ledger. Furthermore, for fair distribution of the donations, we limit the total amount of donations that a recipient can claim in a given time period. Figure 3.2 shows the flow of transactions in our e-donation scheme.

Our Contributions. We list the overall contributions of this chapter as follows:

1. In Section 3.3, we propose a novel, secure, and decentralized e-donation framework. Our generic framework is ledger-based, therefore, it can be realized on top of a distributed blockchain. We scrupulously define the component algorithms and transaction tuples of the framework. We further provide formal game-based definitions for its security properties, (1) unforgeability and balance, (2) fair distribution of donations, (3) donation and recipient privacy.

2. In Section 3.4, we instantiate our e-donation framework with a practical and provably secure scheme. Our scheme is based on two cryptographic building blocks: an anonymized PrivCash scheme (i.e., our abstraction for decentralized anonymous e-cash such as Zerocash [Sasson et al., 2014]) and a recent attribute-based signature scheme called VABS [Biçer and Küpçü, 2019]. In Section 3.8, we formally prove that our proposed scheme satisfies the security requirements of the e-donation framework, if the underlying building blocks are secure.

3. In Section 3.6, we provide efficiency information for our e-donation scheme, showing that our operations have reasonable computation and communication costs (i.e., for many cases, donating takes less than 1.7 min and 6 kB, while receiving donations takes around 1.7 min and less than 90 kB).

3.2 Preliminaries

Notation. Throughout this chapter,

- $a \twoheadleftarrow B$ denotes that a gets a value from the set B sampled uniformly at random,

- $a \leftarrow \mathsf{B}$ denotes that a gets the output of a probabilistic polynomial time algorithm (PPT) B,

- $a := b$ denotes that a gets the value of b,
- $\mathsf{A}(a) \rightarrow b$ denotes that A is a PPT taking as input a and outputting b (A may output values non-deterministically), and
- $\mathsf{A}\{B_1(b_1), B_2(b_2)\} \rightarrow (c_1), (c_2)$ denotes that B_1 and B_2 are two parties executing the protocol A on their inputs b_1 and b_2, respectively, and A outputs c_1 and c_2 to parties B_1 and B_2, respectively (A may output values non-deterministically to both parties).

Ideal Public Ledger. We now briefly describe an ideal public ledger functionality based on the public ledger functionality definition of [Kiayias et al., 2016c]. Any party can securely read the current $\mathsf{Ledger}_\mathsf{C}$, while only the Ledger writer is allowed to write on it by appending some data. Any transaction[4] handed to the Ledger writer is written to the Ledger, if it satisfies a validation function f, i.e., a commonly determined polynomial time function taking as input a transaction, and outputting either 1 (satisfied) or 0 (unsatisfied).

Two commonly known realizations of the Ledger functionality are via a blockchain or via a trusted party. In [Garay et al., 2015] and [Garay et al., 2017], the authors show that the Bitcoin protocol provides the requirements of a public ledger, based on the assumptions that the number of miners are constant, and that the majority of the hashing power belongs to the honest miners. However, existing blockchains, in practice, have issues such as connectivity problems or Eclipse attacks on the peer-to-peer network of blockchain. Therefore, when realizing the Ledger functionality as a blockchain, such issues should be considered. Further, blockchains do not offer network layer anonymity themselves as required by the Ledger functionality, but this can be provided via an additional resource such as using the TOR network [Syverson et al., 2004].

BaseCash scheme. Our e-donation framework requires an underlying

[4]We use the term "transaction" to refer to any information string that conforms to some predefined rules.

BaseCash scheme that functions as the value of the donations. A BaseCash is a tuple (v, p), where v is a value determining the amount and p is a public key of the owner who also has a related secret key s. We note that each BaseCash tuple has a unique reference number r given by the Ledger writer when it is written on the Ledger. In Zerocash, BaseCash corresponds to Bitcoin or a coin belonging to t-addresses in Zerocash [Sasson et al., 2014]. Yet, we do not separate these tuples and refer both as BaseCash. We note that BaseCash may have setup, transaction generation, and verification algorithms, but are abstracted out for simplicity. The key generation algorithm for (s, p) is part of a digital signature scheme that is assumed to satisfy existential unfor- geability under an adaptive chosen-message attack [Katz and Lindell, 2007].

PrivCash scheme. Based on the decentralized e-cash definitions of Zerocash [Sasson et al., 2014], we now describe a generic decentralized anonymous payment scheme PrivCash, which we utilize for donor and recipient privacy in our e-donation scheme. Yet, our description is simplified as including three types of operations PrivCashKeyGen, PrivCashMint, and PrivCashSpend. Also, it differs from the one in [Sasson et al., 2014] by inclusion of a unit BaseCash amount u of each transaction (we require this for fair distribution of donations in our e-donation scheme) and exclusion of details (e.g., Merkle root inputs to spending operations) that are not generalizable. We note that similar ideas exist in [Miers et al., 2013] or other "mixing" based currencies [v. Saberhagen, 2013, Bonneau et al., 2014, Ruffing et al., 2014, Ziegeldorf et al., 2015]. At the start of the PrivCash scheme execution, the Ledger writer runs the setup algorithm, obtains and publishes the global setup parameters $pparams$. The operations PrivCashKeyGen, PrivCashMint, and PrivCashSpend of the PrivCash scheme are defined as follows.

$\mathsf{PrivCashKeyGen}(pparams) \rightarrow (\mathsf{s}, \mathsf{p})$: This is executed by a user for generating the user's private/public key pair (s, p). The user can publish the public key p, but needs to keep s as secret.

$\mathsf{PrivCashMint}(r, s, \mathsf{s}, \mathsf{p}, \mathsf{Ledger_C}, pparams) \rightarrow (c,$
$\mathsf{TX_{PrivCashMint}})$: This is executed by a user (with at least u amount of BaseCash ref-

erenced as r with the corresponding secret key s) for hiding the coins that will be spent. It takes as input the user's private/public key pair (s, p), the current ledger $\mathsf{Ledger}_\mathsf{C}$, and the setup parameters $pparams$. The algorithm outputs the minted PrivCash c and a transaction $\mathsf{TX}_{\mathsf{PrivCashMint}}$. The user sends the output $\mathsf{TX}_{\mathsf{PrivCashMint}}$ to the Ledger writer and keep c as secret spending information.

$\mathsf{PrivCashSpend}(c, \mathsf{s}, \mathsf{p}, \mathsf{info}, p, \mathsf{Ledger}_\mathsf{C}, pparams) \rightarrow \mathsf{TX}_{\mathsf{PrivCashSpend}}$: This is executed by a user for spending her private coin. It takes as input a previously minted PrivCash c, the user's private/public key pair (s, p), a public information info the receiver BaseCash public key p, the current ledger $\mathsf{Ledger}_\mathsf{C}$, and the setup parameters $pparams$. The user sends the output $\mathsf{TX}_{\mathsf{PrivCashSpend}}$ (including info as plaintext) to the Ledger writer. As a result, u amount of PrivCash is deducted from the user's balance and added to the recipient's one as BaseCash.

Each transaction TX_X also has a corresponding verification algorithm (X.Vrfy) so that the Ledger writer can verify that they are structured correctly. The Ledger writer also keeps a serial number database to prevent double-spending. We note that each PrivCashSpend transaction has a unique reference number r given by the Ledger writer when it is written on the Ledger, whereas a PrivCashMint transaction is only referenced implicitly (by spending the minted coin by it). We also note that in case Zerocash is used as PrivCash, PrivCashKeyGen, PrivCashMint and PrivCashSpend correspond to their CreateAddress, Mint, and Pour with the following slight modifications for generality, respectively. s of PrivCashMint is used for signing the output transaction of Mint for verification with the related public key in BaseCash. PrivCashSpend is adjusted so that the output address is BaseCash public key, and p_1 and info of PrivCashSpend are written to info part of the Pour transaction. We assume that BaseCash tuples and PrivCash transactions existing on a single Ledger for simplicity, yet in reality they may be on separate ledgers (e.g., Bitcoin and Zerocash). We assume that PrivCash satisfies correctness, balance and non-malleability and ledger indistinguishability as below.

Balance and Non-Malleability of PrivCash. The definition we provide here combines the "balance" definition of [Miers et al., 2013] (we also make this adap-

tive) and the "transaction non-malleability" definition of [Sasson et al., 2014]. Also it is simplified, as we require a fixed unit amount u for each transaction and by omitting Receive operation (for generality, as Receive does not result in a transaction on the Ledger) and nested PrivCash spending (i.e., the output of Pour cannot be some PrivCash but some BaseCash). Given a PPT adversary $\mathcal{A}$, a challenger $\mathcal{C}$, a PrivCash scheme, and a security parameter λ, consider the following BalanceTNM game:

1. $\mathcal{A}$ is given 1^λ, the unit donation amount u, read access to the Ledger, and the global setup parameters *pparams*. $\mathcal{A}$ is also given query access to a Ledger writer oracle $\mathcal{O}$ for which the allowed query types by $\mathcal{A}$ and the oracle's consequent actions are as follows:

   - PrivCashKeyGen: $\mathcal{O}$ generates a user with private/public key pair s, p by running the algorithm PrivCashKeyGen. It gives p to $\mathcal{A}$ and keeps s private.
   - $(\mathsf{PrivCashMint}, r, s, \mathsf{p})$: r is the referenced BaseCash, s is the related private key to the referenced BaseCash, and p is the public key of a user under $\mathcal{C}$'s control generated by PrivCashKeyGen. If r is unspent, $\mathcal{O}$ generates a $\mathsf{TX}_{\mathsf{PrivCashMint}}$ transaction referencing r.
   - $(\mathsf{PrivCashSpend}, r, p)$: r is a reference to the related $\mathsf{TX}_{\mathsf{PrivCashMint}}$ and p is the public key of the recipient. If no previous transaction has referenced r, $\mathcal{O}$ generates a $\mathsf{TX}_{\mathsf{PrivCashSpend}}$ transaction referencing r spent to the recipient p.
   - $(\mathsf{Insert}, \mathsf{TX}/\mathsf{BaseCash})$ where TX is any type of transaction among $\mathsf{TX}_{\mathsf{PrivCashMint}}$ and $\mathsf{TX}_{\mathsf{PrivCashSpend}}$.If the query input includes BaseCash or the verification algorithm verifies the transaction, $\mathcal{O}$ accepts.

   $\mathcal{O}$ generates transactions as a result of $\mathcal{A}$'s BaseCash, PrivCashMint, and PrivCashSpend type of queries and writes them to the Ledger. If the type

of a query is Insert and the result of $\mathcal{O}$'s action is accept, the related BaseCash or transaction is written to the Ledger.

2. Eventually, $\mathcal{A}$ returns to $\mathcal{C}$ a transaction $\mathsf{TX}'_{\mathsf{PrivCashSpend}}$. $\mathcal{A}$ wins (i.e., the game output is defined as 1), if one of the following two conditions holds: (1) the verification algorithm verifies $\mathsf{TX}'_{\mathsf{PrivCashSpend}}$ with the current $\mathsf{Ledger}_{\mathsf{C}}$ and the union of $\mathsf{TX}'_{\mathsf{PrivCashSpend}}$ and inserted $\mathsf{TX}_{\mathsf{PrivCashSpend}}$ transactions (via Insert queries) on the Ledger exceeds the total amount of inserted $\mathsf{TX}_{\mathsf{PrivCashMint}}$ transactions (via Insert queries) on the Ledger, or (2) $\mathsf{TX}'_{\mathsf{PrivCashSpend}}$ spends a previously spent PrivCash of a user under $\mathcal{C}$'s control by a transaction $\mathsf{TX}''_{\mathsf{PrivCashSpend}}$ such that $\mathsf{TX}''_{\mathsf{PrivCashSpend}} \neq \mathsf{TX}'_{\mathsf{PrivCashSpend}}$, and the verification algorithm verifies $\mathsf{TX}'_{\mathsf{PrivCashSpend}}$ with the portion of Ledger preceding $\mathsf{TX}'_{\mathsf{PrivCashSpend}}$.

Definition 5 (Balance and Non-Malleability)**.** *A* PrivCash *scheme satisfies the balance and non-malleability property, for all PPT adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{n}(\cdot)$ *such that*

$$\Pr[\mathsf{BalanceTNM} = 1] < \mathsf{n}(\lambda)$$

For the sake of simplicity, our e-donation proof utilize the definition given here.

Ledger Indistinguishability of PrivCash. Regarding ledger indistinguishability, we use the game of [Sasson et al., 2014]. Yet, we have simplified the definition of [Sasson et al., 2014], since some features (e.g., Receive, and nested Pour operations, and referencing multiple coins in spending) come by Zerocash are not generalizable to other decentralized anonymous e-cash schemes, and are not utilized in our definitions. Yet, the complete Zerocash scheme satisfies our core security proofs as well. Given a PrivCash scheme, a challenger $\mathcal{C}$, a PPT adversary $\mathcal{A}$, and a security parameter λ, consider the following L-IND game:

1. $\mathcal{C}$ picks a bit $b \twoheadleftarrow \{0,1\}$. It also initializes two ledger writer oracles $\mathcal{O}_0$ and $\mathcal{O}_1$ which writes to ledgers Ledger_0 and Ledger_1, respectively.

2. $\mathcal{A}$ is given 1^λ, the global setup parameters *pparams*, and the unit PrivCash amount $u \in poly(\lambda)$. $\mathcal{A}$ is also given read access to the ledgers $\mathsf{Ledger}_{\mathsf{left}}$ and $\mathsf{Ledger}_{\mathsf{right}}$, where $\mathsf{Ledger}_{\mathsf{left}} := \mathsf{Ledger}_b$ and $\mathsf{Ledger}_{\mathsf{right}} := \mathsf{Ledger}_{\bar{b}}$.

3. $\mathcal{A}$ sends its queries to $\mathcal{C}$. Each query Q must be in the form of a pair of subqueries (Q_0, Q_1). $\mathcal{C}$ forwards one of them to $\mathcal{O}_0$ and the other one to $\mathcal{O}_1$. Allowed query types by $\mathcal{A}$ and the forwarded oracle $\mathcal{O}$'s consequent actions are as follows:

   - PrivCashKeyGen: $\mathcal{O}$ generates a user with private/public key pair s, p by running the algorithm PrivCashKeyGen. It gives p to $\mathcal{A}$ and keeps s private.
   - $(\mathsf{PrivCashMint}, r, s, \mathsf{p})$: r is the referenced BaseCash, s is the related private key to the referenced BaseCash, and p is the public key of a user under $\mathcal{C}$'s control generated by PrivCashKeyGen. If r is unspent, $\mathcal{O}$ generates a $\mathsf{TX}_{\mathsf{PrivCashMint}}$ transaction referencing r.
   - $(\mathsf{PrivCashSpend}, r, p)$: r is a reference to the related $\mathsf{TX}_{\mathsf{PrivCashMint}}$ and p is the public key of the recipient. If no previous transaction has referenced r, $\mathcal{O}$ generates a $\mathsf{TX}_{\mathsf{PrivCashSpend}}$ transaction referencing r spent to the recipient p.
   - $(\mathsf{Insert}, \mathsf{TX}/\mathsf{BaseCash})$ where TX is any type of transaction among $\mathsf{TX}_{\mathsf{PrivCashMint}}$ and $\mathsf{TX}_{\mathsf{PrivCashSpend}}$. If the query input includes BaseCash or the verification algorithm verifies the transaction, $\mathcal{O}$ accepts.

   Insert type of queries are for directly writing to the ledgers. Therefore, in this case, the subqueries Q_0 and Q_1 are forwarded to $\mathcal{O}_b$ and $\mathcal{O}_{\bar{b}}$ by $\mathcal{C}$, respectively. Other queries are for generation of the corresponding transactions. Therefore, in this case, $\mathcal{C}$ forwards Q_0 to $\mathcal{O}_0$ and Q_1 to $\mathcal{O}_1$. We require the following *public consistency* restrictions of [Sasson et al., 2014]) for each query $Q = (Q_0, Q_1)$ to deter trivial de-anonymization.

   - Q_0 and Q_1 must be of the same type.

- If both oracles generate transactions as a result, each oracle writes the generated transaction to the corresponding ledger. Otherwise, both transactions are dropped.
- If both oracles accept as a result of Insert, each oracle writes the accepted transaction or BaseCash to the corresponding ledger. Otherwise, both transactions are dropped.
- If the type is PrivCashMint, the reference r and the public key p must be the same in Q_0 and Q_1.
- If the type is PrivCashSpend, the the recipient p and the public information info must be the same in Q_0 and Q_1.

4. Eventually, $\mathcal{A}$ outputs a bit b'. $\mathcal{A}$ wins (i.e., the game output is defined as 1), if $b = b'$. Otherwise, the $\mathcal{A}$ loses (i.e., the game output is defined as 0).

Definition 6 (Ledger Indistinguishability)**.** *A* PrivCash *scheme satisfies the ledger indistinguishability property, if for all PPT adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{n}(\cdot)$ *such that*

$$\Pr[\text{L-IND} = 1] < \frac{1}{2} + \mathsf{n}(\lambda)$$

For the sake of simplicity, our e-donation proof utilize the definition given here.

3.3 E-Donation Framework

In this section, we define the properties and security notions of an e-donation scheme, along with the games related to them. Our framework requires a Ledger scheme (as described in Section 3.2) that can be instantiated by a trusted third party or a distributed system, and is publicly available for read access. No one can directly write on the Ledger except for the honest Ledger writer. The Ledger is characterized as permanent, i.e., whatever is written on it remains unalterable afterwards. The e-donation framework is built on top of a BaseCash scheme that makes donations worthy.

3.3.1 Operations

We define the e-donation operations, GlobalSetup, IdPJoin, AuthJoin, UserJoin, RecipJoin, AttrGen, PrivateDonate, and PrivateClaim, as follows.

$\mathsf{GlobalSetup}(1^\lambda) \rightarrow (dparams)$: This operation is run by the Ledger writer at the start of the protocol. It takes as input a unary security parameter 1^λ, and outputs global setup parameters $dparams$. The Ledger writer may also update $dparams$ using this operation. We note that if the Ledger writer is a distributed system (as in blockchain), this operation might run as a multi-party protocol where all involved parties have the same input 1^λ, and obtain $dparams$ at the end.

$\mathsf{IdPJoin}(dparams) \rightarrow (is, ip)$: This operation is run by an identity provider for generation of its private/public key pair (is, ip).

$\mathsf{AuthJoin}(dparams) \rightarrow (as, ap)$: This operation is run by an attribute authority for generation of its private/public key pair (as, ap).

$\mathsf{UserJoin}(dparams) \rightarrow (\mathsf{s}, \mathsf{p})$: This operation is run by a user (a donor or a recipient) for generation of its private/public key pair s, p.

$\mathsf{RecipJoin}\{\text{User}(uid, ip), \text{Identity Provider}(uid, is, ip)\} \rightarrow (\mathsf{S}), (\perp)$: This operation is a two-party protocol between a user (a potential recipient) and an identity provider for generating the user's secret key S that will be used receiving donations. The identity provider has a secret key is, and both parties know the user identity uid and the identity provider public key ip. The algorithm outputs to the user the secret key S that is tied to uid.

$\mathsf{AttrGen}\{\text{User}(uid, \mathsf{S}, ap, \omega), \text{Attribute Authority}(uid, as, ap, \omega)\} \rightarrow (\sigma_\omega), (\perp)$: This operation is a two-party protocol between a user and an attribute authority for issuing the attribute token to the user. The user has a secret S obtained from RecipJoin, the authority has a secret key as, and both parties know the user identity uid, the authority public key ap, and the attribute ω that the user wishes to get approved upon. The algorithm outputs to the user the attribute token σ_ω that proves the conformity of the user with secret key S to the attribute ω.

$\mathsf{PrivateDonate}(r, s, \mathsf{s}, \mathsf{p}, \beta, dparams, \mathsf{Ledger_C}) \rightarrow (\mathsf{TX_{Donate,1}}, \ldots, \mathsf{TX_{Donate,k}})$: This operation is run by a donor for casting the donation to an attribute policy. It

takes as input some BaseCash r with a total of at least a unit amount donation[5] u with the corresponding secret key s, the donor's private/public key pair (s, p), an attribute policy β about the recipient, and the system parameters $dparams$. The algorithm outputs a tuple $(\mathsf{TX}_{\mathsf{Donate},1}, \ldots, \mathsf{TX}_{\mathsf{Donate},\mathsf{k}})$ of k transactions. At least one of these transactions publishes β. We call the transaction tuple $(\mathsf{TX}_{\mathsf{Donate},1}, \ldots, \mathsf{TX}_{\mathsf{Donate},\mathsf{k}})$ belonging to the same PrivateDonate operation as a transaction $\mathsf{TX}_{\mathsf{PrivateDonate}}$. We note that the amount u is deducted from the donor's balance, and is added to the claimable amounts with the policy β as a result.

$\mathsf{PrivateClaim}(r, \mathsf{S}, \Sigma_\beta, IAP, \mathsf{s}, \mathsf{p}, p, dparams, \mathsf{Ledger}_\mathsf{C}) \rightarrow (\mathsf{TX}_{\mathsf{Claim},1}, \ldots, \mathsf{TX}_{\mathsf{Claim},\mathsf{h}})$: This operation is run by a recipient for claiming an unclaimed donation. It takes as input a reference previous $\mathsf{TX}_{\mathsf{Donate},\mathsf{k}}$ transaction r on the current $\mathsf{Ledger}_\mathsf{C}$, the recipient's secret key S for receiving donations, an attribute token set Σ_β for proving conformity to β of the referenced $\mathsf{TX}_{\mathsf{Donate},\mathsf{k}}$ issued by attribute authorities, the set IAP of identity provider and attribute authority public keys, the recipient's key pair (s, p), the BaseCash public key p of the payee[6], and the system parameters $dparams$. The algorithm outputs a tuple $(\mathsf{TX}_{\mathsf{Claim},1}, \ldots, \mathsf{TX}_{\mathsf{Claim},\mathsf{h}})$ of h transactions. At least one of these transactions publishes p. We call the transaction tuple $(\mathsf{TX}_{\mathsf{Claim},1}, \ldots, \mathsf{TX}_{\mathsf{Claim},\mathsf{h}})$ belonging to the same PrivateClaim operation as a transaction $\mathsf{TX}_{\mathsf{PrivateClaim}}$. We note that the amount u is deducted from the claimable amounts with β, and is added to the payee's balance as a result.

Each transaction also has a corresponding verification algorithm, and is written to the Ledger after the Ledger writer verifies it. The public outputs of

[5]We set the unit donation amount for achieving fair distribution of donations (see Section 3.3.2) via VABS [Biçer and Küpçü, 2019]. Yet, for further improvements it can be removed. We allow the reference to have total amount more than u for possible donation fees charged by the Ledger writer.

[6]The payee is not the recipient of the e-donation, but one to whom the recipient pays the e-donation. PrivateClaim operation not only ensures the privacy of the recipient but also hides the final payee of a donation, who may be the same as or different from the recipient.

IdPJoin and AuthJoin can be written to Ledger for authenticity. We enforce each PrivateDonate and each PrivateClaim to have the same unit amount u determined by the Ledger writer (as a public system parameter, for fair distribution of donations). This issue will become clearer when we present our construction. We note that some e-donation constructions may not support revocation. Therefore, we do not formalize the related operations, yet in our e-donation construction we achieve revocation via use of VABS implicitly. Also, we note that both k and h would ideally, be set as 1 for better efficiency. Yet, currently this causes issues in privacy, which will be clearer in the privacy definition in Section 3.3.2.

3.3.2 Security Definitions

In this section, we provide the security requirements in an e-donation scheme $\Pi = (\mathsf{Ledger}, \mathsf{BaseCash}, \mathsf{GlobalSetup}, \mathsf{IdPJoin}, \mathsf{AuthJoin}, \mathsf{UserJoin}, \mathsf{RecipJoin}, \mathsf{AttrGen}, \mathsf{PrivateDonate}, \mathsf{PrivateClaim})$. We provide our unforgeability and balance game building on top of the conventional existential unforgeability games, the Balance of [Miers et al., 2013], and the TR-NM game of [Sasson et al., 2014] (see Section 3.2). We propose a fair distribution of donations definition, which we prepared building upon the soundness of n-times unlinkable scheme of [Camenisch et al., 2006], applying that to e-donation. Our privacy notions for donation and recipient privacy follows similar ideas to the transaction privacy in the decentralized anonymous e-cash scheme [Sasson et al., 2014].

Unforgeability and Balance. The unforgeability and balance property of an e-donation scheme ensures that each donation has a valid donor who owns some BaseCash with enough amount, and that the recipient of a donation conforms to the attribute policy chosen by the donor. Our following definition implies that any forging or imbalance attempt within an e-donation scheme will be detected with one minus negligible probability. In fact, we cover four different possibilities of dishonest attempts (i.e., posting $\mathsf{TX}_{\mathsf{PrivateDonate}}$ using another party's BaseCash, posting imbalanced $\mathsf{TX}_{\mathsf{PrivateDonate}}$ (or modification of one generated by an honest

user as a forged donation to a different attribute policy[7]), posting $\mathsf{TX}_{\mathsf{PrivateClaim}}$ that references $\mathsf{TX}_{\mathsf{Donate},2}$ for a user that does not satisfy the related attribute policy, and posting imbalanced $\mathsf{TX}_{\mathsf{PrivateClaim}}$) (or modification of one generated by an honest user as a forged claim to a different payee[7]) by an active adversary. Here, we combine these four attacking possibilities in a single definition, since their definitions would be overlapping if we had defined them separately. Formally, for an e-donation protocol Π, a PPT adversary $\mathcal{A}$, a challenger $\mathcal{C}$ who also controls the Ledger writer, and a security parameter λ, consider the following unforgeability experiment $\mathsf{eDonForge}^{u,n,\delta}_{\mathcal{A},\Pi}(\lambda)$:

1. $\mathcal{C}$ gives to $\mathcal{A}$ the security parameter 1^λ, the system parameters $dparams$ obtained by running $\mathsf{GlobalSetup}(1^\lambda)$, and read access to all Ledger content. $\mathcal{A}$ is also given the unit donation amount $u \in poly(\lambda)$, the total donation amount $D = n \cdot u$ that can be received by a recipient in a given time period where $n \in poly(\lambda)$, and the duration δ of a time period[8]. Time period counter t is initialized as $t := 1$, and is started.

2. At any step,

   (a) $\mathcal{A}$ is allowed to generate identity providers and attribute authorities and issue secret keys and attributes from those authorities. $\mathcal{A}$ shares authority public keys with $\mathcal{C}$ so that it would be able to verify the transactions. Similarly, $\mathcal{A}$ can ask $\mathcal{C}$ to generate authorities by IdPJoin and AuthJoin operations, respectively. In this case, $\mathcal{C}$ shares authority public keys with $\mathcal{A}$, and provides RecipJoin and AttrGen oracle access to $\mathcal{A}$.

   (b) $\mathcal{A}$ can generate users by running UserJoin and run RecipJoin with any identity provider to upgrade any of those users to a recipient. Also, $\mathcal{A}$ can generate polynomially-many different transaction tuples

[7]This is important in case of Blockchain use for Ledger instantiation for security against malicious network participants.

[8]We define δ as a quantized value, i.e., a multiple of computation steps of $\mathcal{A}$.

of $\mathsf{TX}_{\mathsf{PrivateDonate}}$ or $\mathsf{TX}_{\mathsf{PrivateClaim}}$ or BaseCash employing users under its control, and give the tuples to $\mathcal{C}$ for writing to the Ledger. Similarly, $\mathcal{A}$ can ask $\mathcal{C}$ to generate users by UserJoin and to upgrade any user to a recipient by RecipJoin with any identity provider. In this case their public keys are given to $\mathcal{A}$. In this case their public keys are given to $\mathcal{A}$. $\mathcal{A}$ can also ask $\mathcal{C}$ to generate transactions for users under $\mathcal{C}$'s control and write them to the Ledger. In any case, only the transactions verified by the verification algorithm are written to the Ledger.

3. $\mathcal{C}$ generates BaseCash tuples $(v_1, p_1), \ldots, (v_\ell, p_\ell)$ where $\ell \in poly(\lambda)$ with $\mathcal{A}$'s request, writes them to the Ledger, and keeps the related secret keys $s_1, \ldots, s_\ell$ private.

4. $\mathcal{A}$ eventually returns an attribute ω (that was not queried to AttrGen oracles) to $\mathcal{C}$. We note that ω can only be verified by the (honest) authority public keys generated by the challenger. If $\mathcal{A}$ queries an attribute authority oracle for ω, the attribute is issued but is immediately revoked. If the scheme does not support revocation, definition simply says that such an attribute assignment request will be denied.

5. $\mathcal{A}$ eventually returns (at least) one of the following: (1) a transaction tuple $\mathsf{TX}_{\mathsf{PrivateDonate}}$ whose references include a subset of the $\{(v_1, p_1), \ldots, (v_\ell, p_\ell)\}$ (as if a malicious user is spending some honest user's coins), (2) a transaction tuple $\mathsf{TX}_{\mathsf{PrivateDonate}}$ without the deduction of the amount u from the donor's balance, or without the addition of u to the balance of unclaimed donations with the same policy, or with the alteration of the balance of another party or of unclaimed donations with another policy, or that can replace an existing transaction tuple $\mathsf{TX}'_{\mathsf{PrivateDonate}}$ with a different policy on the Ledger generated by a user under $\mathcal{C}$'s control (and the verification algorithm verifies all the transactions on the Ledger upto and including $\mathsf{TX}_{\mathsf{Donate,k}}$) (3) a transaction tuple $\mathsf{TX}_{\mathsf{PrivateClaim}}$ that references a

$\mathsf{TX}_{\mathsf{Donate},2}$ with a policy $\beta \wedge \omega$ for any β (as if claiming a donation without satisfying the attribute), (4) a transaction tuple $\mathsf{TX}_{\mathsf{PrivateClaim}}$ that has a final payee that receives more amount than u without the deduction of the amount u from the donations with the same policy β that are unclaimed, or without the addition of u to the payee's balance, or with the alteration of the balance of any other party or of unclaimed donations with another policy, or that can replace an existing transaction tuple $\mathsf{TX}'_{\mathsf{PrivateClaim}}$ on the Ledger with a different payee generated by a user under $\mathcal{C}$'s control (and the verification algorithm verifies all the transactions on the Ledger upto and including $\mathsf{TX}_{\mathsf{Claim},\mathsf{k}}$). $\mathcal{A}$ sends the generated tuples to $\mathcal{C}$. The output of the game is defined as 1 (i.e., $\mathcal{A}$ wins), if any such transaction tuple is written to the Ledger. Otherwise, the output of the game is defined as 0 (i.e., $\mathcal{A}$ loses).

Definition 7 (Unforgeability and Balance)**.** *An e-donation scheme* Π *is* unforgeable and balanced*, if* $\forall u, n, \delta \in poly(\lambda)$*, for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{n}(\cdot)$ *such that*

$$\Pr[\mathsf{eDonForge}^{u,n,\delta}_{\mathcal{A},\Pi}(\lambda) = 1] < \mathsf{n}(\lambda)$$

Fair Distribution of Donations. By fairness in donation distribution, we refer to the objective that none of the recipients would be allowed to claim too many donations even if she satisfies their policies. To achieve this, we limit the total donation amount that can be received by a recipient within a time interval. Our definition ensures that any adversarial attempt within an e-donation scheme for obtaining more than the allowed amount within a time period will be detected with one minus negligible probability. For an e-donation protocol Π, a PPT adversary $\mathcal{A}$, a challenger $\mathcal{C}$ who also controls the Ledger writer, and a security parameter λ, consider the following fair distribution of donation experiment $\mathsf{FairDist}^{u,n,\delta}_{\mathcal{A},\Pi}(\lambda)$:

1. $\mathcal{C}$ gives to $\mathcal{A}$ the security parameter 1^λ, the system parameters *dparams* obtained by running $\mathsf{GlobalSetup}(1^\lambda)$, and read access to all Ledger content. $\mathcal{A}$ is

also given the unit donation amount $u \in poly(\lambda)$, the total donation amount $D = n \cdot u$ that can be received by a recipient in a given time period where $n \in poly(\lambda)$, and the duration δ of a time period. Time period counter t is initialized as $t := 1$, and is started.

2. At any step,

   (a) $\mathcal{A}$ is allowed to generate identity providers and attribute authorities and issue secret keys and attributes from those authorities. $\mathcal{A}$ shares authority public keys with $\mathcal{C}$ so that it would be able to verify the transactions. Similarly, $\mathcal{A}$ can ask $\mathcal{C}$ to generate authorities by $\mathsf{IdPJoin}$ and $\mathsf{AuthJoin}$ operations, respectively. In this case, $\mathcal{C}$ shares authority public keys with $\mathcal{A}$ and provides $\mathsf{RecipJoin}$ and $\mathsf{AttrGen}$ oracle access to $\mathcal{A}$.

   (b) $\mathcal{A}$ can generate users by running $\mathsf{UserJoin}$ and run $\mathsf{RecipJoin}$ with any identity provider to upgrade any of those users to a recipient. Also, $\mathcal{A}$ can generate polynomially-many different transaction tuples of $\mathsf{TX}_{\mathsf{PrivateDonate}}$ or $\mathsf{TX}_{\mathsf{PrivateClaim}}$ or $\mathsf{BaseCash}$ employing users under its control, and give the tuples to $\mathcal{C}$ for writing to the Ledger. Similarly, $\mathcal{A}$ can ask $\mathcal{C}$ to generate users by $\mathsf{UserJoin}$ and to upgrade any user to a recipient by $\mathsf{RecipJoin}$ with any identity provider. In this case their public keys are given to $\mathcal{A}$. In this case their public keys are given to $\mathcal{A}$. $\mathcal{A}$ can also ask $\mathcal{C}$ to generate transactions for users under $\mathcal{C}$'s control and write them to the Ledger. In any case, only the transactions verified by the verification algorithm are written to the Ledger.

3. $\mathcal{A}$ eventually returns to $\mathcal{C}$ a transaction tuple for $\mathsf{TX}_{\mathsf{PrivateClaim},\mathcal{A}}$ whose recipient with secret key S has claimed at least D amount in the current time period. The output of the game is defined as 1 (i.e., $\mathcal{A}$ wins), if the transaction tuple is written to the Ledger. Otherwise, the output of the game is defined as 0 (i.e., $\mathcal{A}$ loses).

Definition 8 (Fair Distribution of Donations)**.** *An e-donation scheme* Π *provides* fair

The challenger $\mathcal{C}$	The adversary $\mathcal{A}$
Picks $b \leftarrow_r \{0,1\}$. Initializes two ledgers L_0 and L_1, and two oracles $\mathcal{O}_0$ and $\mathcal{O}_1$ that writes to them, respectively. Defines ledgers $L_{left} = L_b$ and $L_{right} = L_{\bar{b}}$.	
$\xrightarrow{pparams, \text{ read access to } L_{left} \text{ and } L_{right}, \text{ query access}}$	
	$\mathcal{A}$ may query polynomially-many times as pairs of subqueries (Q_0, Q_1). Types of Q_0 and Q_1 should be the same in each query.
If query type is Insert, forwards Q_0 to $\mathcal{O}_0$, and Q_1 to $\mathcal{O}_1$. Otherwise, forwards Q_0 to $\mathcal{O}_b$, and Q_1 to $\mathcal{O}_{\bar{b}}$.	
	At the end, outputs a bit b'. Wins, if $b = b'$. Otherwise, loses.

Figure 3.3: A high level description of the $\mathsf{Privacy}^{u,n,\delta}_{\mathcal{A},\Pi}(\lambda)$ game.

distribution of donations, *if* $\forall u, n, \delta \in poly(\lambda)$, *for every PPT adversary* $\mathcal{A}$, *there exists a negligible function* $\mathsf{n}(\cdot)$ *such that*

$$\Pr[\mathsf{FairDist}^{u,n,\delta}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \mathsf{n}(\lambda)$$

Donation and Recipient Privacy. Our privacy notion includes hiding the identities of the donor and recipient of a donation and its final payee. We note that payee privacy is also very important for ensuring the privacy of a recipient, as the payee is likely to be recipient himself or someone with a relationship to him.

To cover the above-mentioned privacy issues, we propose a strong privacy definition similar to the ledger indistinguishability of [Sasson et al., 2014]. Formally, for an e-donation protocol Π, a PPT adversary $\mathcal{A}$, a challenger $\mathcal{C}$ who also controls the Ledger writer, and a security parameter λ, consider the following donation and recipient privacyexperiment $\mathsf{Privacy}^{u,n,\delta}_{\mathcal{A},\Pi}(\lambda)$:

1. $\mathcal{C}$ gives to $\mathcal{A}$ the security parameter 1^λ, the system parameters $dparams$ obtained by running $\mathsf{GlobalSetup}(1^\lambda)$. $\mathcal{A}$ is also given the unit donation amount $u \in poly(\lambda)$, the total donation amount $D = n \cdot u$ that can be received by a recipient in a given time period where $n \in poly(\lambda)$, and the duration δ of a time period. Time period counter t is initialized as $t := 1$, and is started.

2. At any step,

(a) $\mathcal{A}$ is allowed to generate identity providers and attribute authorities and issue secret keys and attributes from those authorities. $\mathcal{A}$ shares authority public keys with $\mathcal{C}$ so that it would be able to verify the transactions. Similarly, $\mathcal{A}$ can ask $\mathcal{C}$ to generate authorities by $\mathsf{IdPJoin}$ and $\mathsf{AuthJoin}$ operations, respectively. In this case, $\mathcal{C}$ shares authority public keys with $\mathcal{A}$ and provides $\mathsf{RecipJoin}$ and $\mathsf{AttrGen}$ oracle access to $\mathcal{A}$.

(b) $\mathcal{A}$ can generate users by running $\mathsf{UserJoin}$ and run $\mathsf{RecipJoin}$ with any identity provider to upgrade any user to a recipient. Similarly, $\mathcal{A}$ can ask $\mathcal{C}$ to generate users by $\mathsf{UserJoin}$ and to upgrade any user to a recipient by $\mathsf{RecipJoin}$ with any identity provider. In this case their public keys are given to $\mathcal{A}$.

3. $\mathcal{C}$ picks a bit $b \twoheadleftarrow \{0,1\}$. It also initializes two ledger writer oracles $\mathcal{O}_0$ and $\mathcal{O}_1$ which writes to ledgers Ledger_0 and Ledger_1, respectively.

4. $\mathcal{A}$ is read access to the ledgers $\mathsf{Ledger}_{\mathsf{left}}$ and $\mathsf{Ledger}_{\mathsf{right}}$, where $\mathsf{Ledger}_{\mathsf{left}} := \mathsf{Ledger}_b$ and $\mathsf{Ledger}_{\mathsf{right}} := \mathsf{Ledger}_{\bar{b}}$.

5. $\mathcal{A}$ sends its queries to $\mathcal{C}$. Each query Q must be in the form of a pair of subqueries (Q_0, Q_1). $\mathcal{C}$ forwards one of them to $\mathcal{O}_0$ and the other one to $\mathcal{O}_1$. Allowed types of subqueries are $\mathsf{BaseCash}$, $(\mathsf{Donate}, \mathsf{i})$ for $\mathsf{i} \in (1, \ldots, \mathsf{k})$, $(\mathsf{Claim}, \mathsf{i})$ for $\mathsf{i} \in (1, \ldots, \mathsf{h})$, and Insert. Insert queries are for the $\mathsf{BaseCash}$ tuples and transactions generated by the users under $\mathcal{A}$'s control, while the other ones are for those under $\mathcal{C}$'s control. The format of and the oracle's consequent action for each type should be as follows.

   - $(\mathsf{BaseCash}, v, p)$: v is the amount and p is the public key of the owner. If p is among the public keys of the users under $\mathcal{C}$'s control the oracle generates a $\mathsf{BaseCash} = (v, p)$.
   - $(\mathsf{PrivateDonate}, r, \mathsf{p}, \beta)$: r is the referenced $\mathsf{BaseCash}$, p is the public key of a user, and β is an attribute policy. If no previous transaction has

referenced r, p is among the users under $\mathcal{C}$'s control generated by $\mathsf{UserJoin}$, and β is a validly structured attribute policy, the oracle assigns a unique donate identifier qid_D to the query, gives qid_D to $\mathcal{A}$, and records the query and its qid_D.

- $(\mathsf{Donate}, \mathsf{i}, qid_D)$ for $\mathsf{i} \in (1, \ldots, \mathsf{k})$: qid_D is the referenced $\mathsf{PrivateDonate}$ query. If no previous transaction $\mathsf{TX}_{\mathsf{Donate,i}}$ referencing qid_D is written on the oracle's Ledger, the oracle generates a $\mathsf{TX}_{\mathsf{Donate,i}}$ transaction by executing the related transaction generation algorithm using the inputs of the related $\mathsf{PrivateDonate}$ query.
- $(\mathsf{PrivateClaim}, r, \mathsf{p}, p)$: r is the referenced $\mathsf{TX}_{\mathsf{Donate,k}}$, p is the public key of a user, and p is the public key of the payee. If no previous transaction has referenced r, p is among the users under $\mathcal{C}$'s control generated by $\mathsf{UserJoin}$, and p is a validly structured public key, the oracle assigns a unique claim identifier qid_C to the query, gives qid_C to $\mathcal{A}$, and records the query and its qid_C.
- $(\mathsf{Claim}, \mathsf{i}, qid_C)$ for $\mathsf{i} \in (1, \ldots, \mathsf{h})$: qid_C is the referenced $\mathsf{PrivateClaim}$ query. If no previous transaction $\mathsf{TX}_{\mathsf{Claim,i}}$ referencing qid_C is written on the oracle's Ledger, the oracle generates a $\mathsf{TX}_{\mathsf{Claim,i}}$ transaction by executing the related transaction generation algorithm using the inputs of the related $\mathsf{PrivateClaim}$ query.
- $(\mathsf{Insert}, \mathsf{TX}/\mathsf{BaseCash})$ where TX is any type of transaction among $\mathsf{TX}_{\mathsf{Donate,i}}$ for $\mathsf{i} \in (1, \ldots, \mathsf{k})$ and $\mathsf{TX}_{\mathsf{Claim,i}}$ for $\mathsf{i} \in (1, \ldots, \mathsf{h})$. If the query input includes $\mathsf{BaseCash}$ or the verification algorithm verifies the transaction, the oracle accepts.

Insert type of queries are for directly writing to the ledgers. Therefore, in this case, the subqueries Q_0 and Q_1 are forwarded to $\mathcal{O}_b$ and $\mathcal{O}_{\bar{b}}$ by $\mathcal{C}$, respectively. Other queries are for generation of the corresponding transactions. Therefore, in this case, $\mathcal{C}$ forwards Q_0 to $\mathcal{O}_0$ and Q_1 to $\mathcal{O}_1$. We require the following restrictions (defined similarly to *public consistency* of [Sasson et al.,

2014]) for each query $Q = (Q_0, Q_1)$ to deter trivial de-anonymization.

- Q_0 and Q_1 must be of the same type.
- If both oracles generate transactions as a result, each oracle writes the generated transaction to the corresponding ledger. Otherwise, both transactions are dropped.
- If both oracles accept as a result of Insert, each oracle writes the accepted $\mathsf{BaseCash}$ to the corresponding ledger. Otherwise, both transactions are dropped.
- If the type is $\mathsf{Donate}, \mathsf{i}$ for $\mathsf{i} \in (1, \ldots, \mathsf{k})$, there exists two cases. If the attribute policies are being published with the resulting transactions, the policies in both referenced $\mathsf{PrivateDonate}$ queries by $qid_{D,0}$ (belonging to Q_0) and $qid_{D,1}$ (belonging to Q_1) must be the same. Otherwise, the referenced $\mathsf{Basecash}$ r and the related public key p in both referenced $\mathsf{PrivateDonate}$ queries by $qid_{D,0}$ and $qid_{D,1}$ must be the same.
- If the type is $\mathsf{Claim}, \mathsf{i}$ for $\mathsf{i} \in (1, \ldots, \mathsf{h})$, there exists two cases. If the payee public keys are being published with the resulting transactions, the public keys in both referenced $\mathsf{PrivateClaim}$ queries by $qid_{C,0}$ (belonging to Q_0) and $qid_{C,1}$ (belonging to Q_1) must be the same. Otherwise, the referenced $\mathsf{Donate}, \mathsf{k}$ r in both referenced $\mathsf{PrivateDonate}$ queries by $qid_{C,0}$ and $qid_{C,1}$ must be the same.

6. Eventually, $\mathcal{A}$ outputs a bit b'. If $b = b'$, the output of the game is defined as 1 (i.e., $\mathcal{A}$ wins). Otherwise, the output of the game is defined as 0 (i.e., $\mathcal{A}$ loses).

Definition 9 (Donation and Recipient Privacy)**.** *An e-donation scheme* Π *provides* donation and recipient privacy*, if* $\forall u, n, \delta \in poly(\lambda)$*, for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{n}(\cdot)$ *such that*

$$\Pr[\mathsf{Privacy}^{u,n,\delta}_{\mathcal{A},\Pi}(\lambda) = 1] < \frac{1}{2} + \mathsf{n}(\lambda)$$

Figure 3.3 provides a high level description of this game.

Remark 2. *Consider schemes with* $\mathsf{k} = 1$ *or* $\mathsf{h} = 1$. *Regarding* PrivateDonate, *the resulting transactions need to include the referenced* BaseCash *and the attribute policy* β. *Having them together in one transaction directly shows whose* BaseCash *is donated to a particular policy. Regarding* PrivateClaim, *the resulting transactions need to include the referenced* $\mathsf{TX}_{\mathsf{Donate},\mathsf{k}}$ *and the public key* p *of the payee. Having them together in one transaction would directly show which claimed donation (along with the published attribute policy) is spent to a particular payee. Therefore, in our construction, we achieve security with* $\mathsf{k} = 2$ *and* $\mathsf{h} = 2$, *by splitting the* PrivateDonate *and* PrivateClaim *transactions into two steps, disassociating the referenced* BaseCash *from the attribute policy, and disassociating the referenced donation from the payee, respectively.*

3.4 Our e-Donation Scheme

In this section, we propose the first e-donation scheme $\Pi = (\mathsf{Ledger}, \mathsf{GlobalSetup}, \mathsf{BaseCash}, \mathsf{IdPJoin}, \mathsf{AuthJoin}, \mathsf{UserJoin}, \mathsf{RecipJoin}, \mathsf{AttrGen}, \mathsf{PrivateDonate}, \mathsf{PrivateClaim})$ that achieves the security requirements of the generic e-donation framework, namely unforgeability and balance, fair distribution of donations, and donation and recipient privacy. Our scheme also provides multi-authority attribute issuing for recipients. We do not propose or enforce a specific choice of the Ledger instantiation, i.e., it can be a decentralized blockchain that utilizes any consensus methodology (such as proof-of-work, proof-of-stake, or Byzantine agreement, or a combination of multiple ledgers[9]). Our PrivateDonate and PrivateClaim instantiation utilizes as building blocks a VABS scheme defined as in [Biçer and Küpçü, 2019] and a PrivCash scheme such as Zerocash [Sasson et al., 2014]. We highlight the necessity of the VABS scheme, because it is the only

[9]BaseCash and PrivCash schemes, in practice, might have separate ledgers (e.g., Bitcoin and Zerocash). Here, we treat the whole system as written on a single ledger for simplicity. In case Zerocash is used for implementation, e-donation operations might be written on the ledger of Zerocash with a modification on its Ledger writer (e.g., miners' code for verifying VABS).

known ABS scheme that enables both anonymous attribute policy proving and limiting the number of signatures that a signer can generate in a time period (see Section 3.5 for further discussion). In what follows, we describe our e-donation scheme proposal in detail.

In GlobalSetup of our e-donation scheme, the setup operations of the BaseCash, PrivCash, and VABS schemes are executed by the Ledger writer. It also generates a BaseCash private/public key pair $(\dot{s}, \dot{p})$ for indicating e-donation operations and completness. The Ledger writer then sets $dparams := (vparams, pparams, \dot{s}, \dot{p})$. In our proposed scheme IdPJoin and AuthJoin operations are obtained by authorities via execution of VABS.IdPJoin and VABS.AuthJoin operations, respectively. Likewise, UserJoin is obtained by users via execution of PrivCashKeyGen. Further, for RecipJoin and AttrGen, the user and the authority simply run the VABS.UserJoin and VABS.AttrIssue operations, respectively. It should be noted that the VABS scheme of [Biçer and Küpçü, 2019] also requires each user to run VABS.UserJoin with an identity provider, before obtaining her attribute signatures, when they first join the system.

Algorithms 4 and 5 present our PrivateDonate and PrivateClaim operations run by a donor and a recipient respectively. A PrivateDonate transaction employs PrivCashMint and PrivCashSpend transactions to maintain unlinkability, while adding the attribute policy. A PrivateClaim transaction utilizes again PrivCashMint and PrivCashSpend transactions to maintain unlinkability, and a VABS.Sign operation to prove adherence to the attribute policy. We note that the resulting $\mathsf{TX}_{\mathsf{Donate},1}$, $\mathsf{TX}_{\mathsf{Donate},2}$, $\mathsf{TX}_{\mathsf{Claim},1}$, and $\mathsf{TX}_{\mathsf{Claim},2}$ transactions in Algorithms 4 and 5 correspond to PrivCashMint, Donate, Claim, and PrivCashSpend transactions given in Overview of Our Techniques in Section 3.1.

For verification of $\mathsf{TX}_{\mathsf{Donate},1}$, $\mathsf{TX}_{\mathsf{Donate},2}$, $\mathsf{TX}_{\mathsf{Claim},1}$, and $\mathsf{TX}_{\mathsf{Claim},2}$ transactions, the Ledger writer runs Algorithm 6. SDB is a database on a dedicated portion of the Ledger. The Ledger writer keeps there the VABSs obtained from each Claim transaction.We highlight that some portion (e.g., SDB can also be used) of the Ledger is dedicated to storing the identity provider and attribute authority public

Algorithm 4 PrivateDonate executed by donors.

input: some BaseCash r on the current $\mathsf{Ledger}_\mathsf{C}$ with a total amount of at least u, the related secret key s to the BaseCash, the donor's private/public key pair (s, p), an attribute policy β, and the global setup parameters $dparams = (vparams, pparams, \dot{s}, \dot{p})$

output: a tuple $\mathsf{TX}_{\mathsf{PrivateDonate}} = (\mathsf{TX}_{\mathsf{Donate},1}, \mathsf{TX}_{\mathsf{Donate},2})$

$(c, \mathsf{TX}_{\mathsf{Donate},1}) \leftarrow \mathsf{PrivCashMint}(r, s, \mathsf{s}, \mathsf{p}, \mathsf{Ledger}_\mathsf{C}, pparams)$

Send $\mathsf{TX}_{\mathsf{Donate},1}$ to the Ledger

$\mathsf{TX}_{\mathsf{Donate},2} \leftarrow \mathsf{PrivCashSpend}(c, \mathsf{s}, \mathsf{p}, \beta, \dot{p}, \mathsf{Ledger}_\mathsf{C}, pparams)$

Send $\mathsf{TX}_{\mathsf{Donate},2}$ to the Ledger

keys to ensure their authenticity via a public key infrastructure as in [Axon and Goldsmith, 2017, Yakubov et al., 2018, Kubilay et al., 2018]. This would be useful as the VABS scheme [Biçer and Küpçü, 2019] also allows an authority to join the system at anytime via a valid certificate that demonstrates its qualification. Furthermore, the Ledger writer can dismiss an authority from issuing some (or all) attributes by excluding its public key from verification.

3.5 Related Work

Private Transactions. For PrivCash instantiation, many Bitcoin-like solutions depending only on pseudonyms for user privacy and not preventing linkability between transactions are not usable. Zerocash [Sasson et al., 2014] can be utilized to instantiate PrivCash. For utilization of Zerocoin [Miers et al., 2013] as a PrivCash scheme, it needs to be shown to be or further improved for satisfying the balance and transaction malleability (see Section 3.2), and ledger indistinguishability of (see Section 3.2). Further, mixing based schemes (e.g., CryptoNote [v. Saberhagen, 2013], Mixcoin [Bonneau et al., 2014], CoinShuffle [Ruffing et al., 2014], CoinParty [Ziegeldorf et al., 2015]) also provide transaction privacy. Nevertheless, an analysis [Möser et al., 2018] on Monero [Monero, 2014] (i.e., a CryptoNote based cryptocurrency) shows that these type of cryptocurrencies are vulnerable to "chain-reaction analysis". To completely avoid these type of attacks, the anonymized

Algorithm 5 PrivateClaim executed by recipients.

input: a previous unclaimed transaction $\mathsf{TX}_{\mathsf{PrivCashMint}}$ referenced with r on the current $\mathsf{Ledger}_{\mathsf{C}}$, the recipient's secret key S for receiving donations, a set of authority signatures Σ_β for proving conformity to the attribute policy β, the recipient's key pair (s, p), the BaseCash public key p of the payee, the set IAP of identity provider and attribute authority public keys, the global setup parameters $dparams = (vparams, pparams, \dot{s}, \dot{p})$, and a PrivateClaim counter J

output: a tuple $\mathsf{TX}_{\mathsf{PrivateClaim}} = (\mathsf{TX}_{\mathsf{Claim,1}}, \mathsf{TX}_{\mathsf{Claim,2}})$

Obtain the attribute policy β from donation r

Obtain the time period t from $\mathsf{Ledger}_{\mathsf{C}}$

$(c, \mathsf{TX}_{\mathsf{Claim,1'}}) \leftarrow \mathsf{PrivCashMint}(r, \dot{s}, \mathsf{s}, \mathsf{p}, \mathsf{Ledger}_{\mathsf{C}}, pparams)$

$\sigma \leftarrow \mathsf{VABS.Sign}(\mathsf{TX}_{\mathsf{Claim,1'}}, \mathsf{S}, \beta, \Sigma_\beta, IAP, t, J, vparams)$

Send $\mathsf{TX}_{\mathsf{Claim,1}} := (\mathsf{TX}_{\mathsf{Claim,1'}}, \sigma)$ to the Ledger

$\mathsf{TX}_{\mathsf{Claim,2}} \leftarrow \mathsf{PrivCashSpend}(c, \mathsf{s}, \mathsf{p}, *, p, \mathsf{Ledger}_{\mathsf{C}}, pparams)$

Send $\mathsf{TX}_{\mathsf{Claim,2}}$ to the Ledger

transactions in these protocols should use all of the previously blinded coins during mixing, which decreases their efficiency due to zero-knowledge proofs. Also, without this, these schemes do not satisfy the L-IND definition (in Section 3.2) that we expect PrivCash to satisfy. Private smart contract schemes (e.g., Hawk [Kosba et al., 2016], Arbitrum [Kalodner et al., 2018], Zether [Bünz et al., 2019], ZEXE [Bowe et al., 2020], and Enigma [Eni, 2019]) enable smart contract computations on private data. There exists no restriction for their utilization as PrivCash basis.

Limited Times Anonymous Attribute Policy Authentication. Our e-donation scheme essentially requires a non-interactive anonymous, multi-authority, and revocable attribute policy signature protocol, where the number of signatures in a period can be limited. We briefly mention the candidate solutions in the literature. A line of work that can be useful is anonymous credential schemes [Camenisch and Lysyanskaya, 2003, Camenisch et al., 2006, Camenisch and Groß, 2012, Au et al., 2013, Garman et al., 2014, Lueks et al., 2015, Derler et al., 2015, Camenisch et al., 2016] that can be utilized for proving conformation to some attribute policy while keeping the anonymity. However, they cannot be trivially modified for directly providing all requirements of the e-donation at once

Algorithm 6 Verification by the Ledger writer.

input: the current $\mathsf{Ledger_C}$ and the global setup parameters $dparams = (vparams, pparams, \dot{s}, \dot{p})$

output: updated $\mathsf{Ledger_C}$

On receiving a $\mathsf{TX_{Donate,1}}$ *transaction:*

if $\mathsf{PrivCashMint.Vrfy}(\mathsf{TX_{Donate,1}}) = 1$ **then**

$\mathsf{Ledger_C} := \mathsf{Ledger_C}||\mathsf{TX_{Donate,1}}$

On receiving a $\mathsf{TX_{Donate,2}}$ *transaction:*

if $\mathsf{PrivCashSpend.Vrfy}(\mathsf{TX_{Donate,2}}) = 1$ **then**

$\mathsf{Ledger_C} := \mathsf{Ledger_C}||\mathsf{TX_{Donate,2}}$

On receiving a $\mathsf{TX_{Claim,1}}$ *transaction:*

Parse $(\mathsf{TX_{Claim,1'}}, \sigma) := \mathsf{TX_{Claim,1}}$

if $\mathsf{PrivCashMint.Vrfy}(\mathsf{TX_{Claim,1'}}) = 1$ **and** $\mathsf{VABS.Verify}(\mathsf{TX_{Claim,1'}}, \sigma, \beta, IAP, t, SDB, vparams) = 1$ **then**

$\mathsf{Ledger_C} := \mathsf{Ledger_C}||\mathsf{TX_{Claim,1}}$

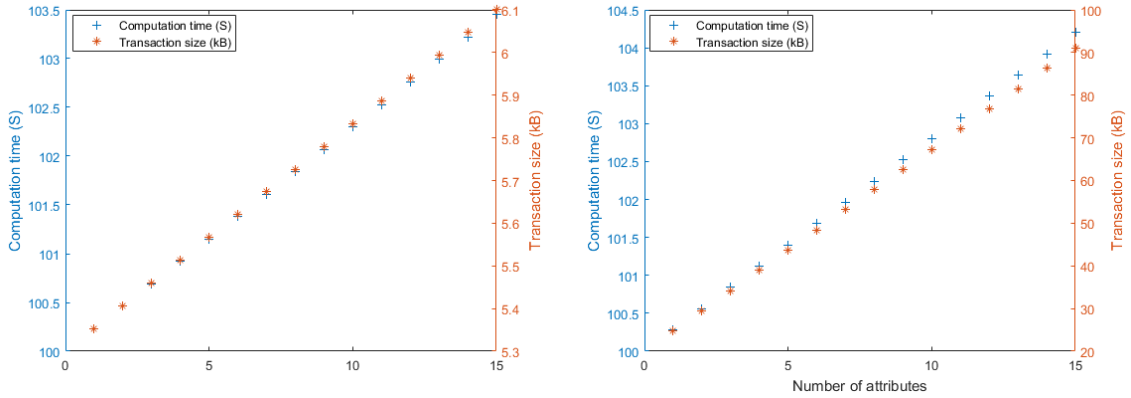
$SDB := SDB||\sigma$

On receiving a $\mathsf{TX_{Claim,2}}$ *transaction:*

if $\mathsf{PrivCashSpend.Vrfy}(\mathsf{TX_{Claim,2}}) = 1$ **then**

$\mathsf{Ledger_C} := \mathsf{Ledger_C}||\mathsf{TX_{Claim,2}}$

(e.g., fair distribution of donations among recipients). Group signature schemes [Camenisch and Groth, 2005, Bellare et al., 2005, Yang et al., 2011, Slamanig et al., 2014, Hwang et al., 2015, Nisansala et al., 2017, Emura et al., 2017] may seem as a reasonable place to start. Some group signature schemes allow revocability [Nisansala et al., 2017, Emura et al., 2017]. On the other hand, again, the use of keys in those schemes cannot be easily bounded by a fixed value n. Attribute based signature (ABS) schemes [Maji et al., 2011, Cao et al., 2012, Okamoto and Takashima, 2013, El Kaafarani et al., 2014b, Hampiholi et al., 2015, Urquidi et al., 2016, Drăgan et al., 2018, Biçer and Küpçü, 2019] are non-interactive signature solutions for anonymous attribute policy proving. They provide binary attribute policy authentication (including Boolean relations). In particular, the existing multi-authority and decentralized ABS schemes of [Cao et al., 2012, Okamoto and Takashima, 2013, El Kaafarani et al., 2014b, Drăgan et al., 2018, Biçer and Küpçü,



(a) Computation time and transaction size of PrivateDonate

(b) Computation time and transaction size of PrivateClaim

Figure 3.4: Computation time and total transaction size estimates for (a) PrivateDonate and (b) PrivateClaim transactions based on the number of referenced BaseCash tuples and the number of attributes in the referenced policy, respectively.

2019] might be modified to achieve periodic n-times usability and revocability.

Gitcoin[10]**.** Gitcoin is a system to fund open-source software projects via cryptocurrencies (e.g., Ethereum [Buterin, 2013]). It allows a developer to join the system by showing only her skills, while all parties remain anonymous. Although the flow of Gitcoin is similar to e-donation, apart from targeted audience (developers vs. general), their operational differences are as follows: (1) The former requires a centrally managed issue advertisement and accountability system while in the latter, the whole system achieve these in a decentralized way. (2) For skills, the former does not require proofs from authorities, while the latter require these proofs for attributes. (3) In the former, there exists no fairness in distribution of salaries, while the latter ensures fair distribution of donations. Although the matters (2) and (3) with Gitcoin can be resolved via updates, the matter (1) with it (i.e., system being centralized) would not be trivially improved.

[10]https://medium.com/gitcoin

3.6 Efficiency

We conclude with the efficiency analysis of our solution based on the state-of-the-art PrivCash scheme Zerocash Sapling [Hopwood et al., 2020] and the VABS scheme [Biçer and Küpçü, 2019].

Test Setup. We have run our tests using the given implementation of [Company, 2018] for "Output" and "Spend" transactions described in [Hopwood et al., 2020]. We note that the former directly corresponds to PrivCashMint, while the latter corresponds to PrivCashSpend. For our Claim operation, we have tested a prototype code for Sign and Verify algorithms of [Biçer and Küpçü, 2019] via the given implemented primitives in the Cashlib cryptographic library [Cashlib, 2010, Meiklejohn et al., 2010]. We have run our test on a machine with Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz 2.90 - dualcore and 8GB RAM. The RSA and discrete logarithm moduli are set as 2048 bits, and SHA-256 is used as the random oracle. Our tests are repeated 10 times and results are averaged. The results are combined to obtain an overall cost output for our PrivateDonate and PrivateClaim operations.

Cost of PrivateDonate. Figure 3.4a provides cost estimates for PrivateDonate operations with varying number of BaseCash tuples referenced (on which the cost linearly depends). We highlight that PrivateDonate operation is efficient enough in large majority of practical cases, since according to statistics of Nonprofits Source[11], the average monthly online donation of United States of America citizens is $52 in 2017, corresponding to 2 Zerocashes referenced as of March 2020, taking around 1.7 min and 5.4 kB. We note that the overhead is mainly due to costly operations in Zerocash.

Cost of PrivateClaim. Figure 3.4b provides cost estimates for PrivateClaim operations with varying number of attributes to be proven (on which the cost linearly depends). Indeed, while the attribute-based signature cost scales linearly with the number of attributes, we do not see this behavior in computation cost

[11] `https://nonprofitssource.com/online-giving-statistics/`

clearly, since VABS takes a few hundred milliseconds while Zerocash takes more than a hundred seconds. We highlight that the PrivateClaim operation is efficient enough in large majority of practical cases, since the donor behaviour tends to be simplistic with usually less than 5 attributes [Breeze, 2013], taking around 1.7 min and 45 kB. The overhead in size compared to PrivateDonate is due to the included VABS.

3.7 Conclusion and Future Work

In this work, we proposed a novel e-donation framework, where donors donate to recipients by only selecting them via their attributes without meeting in person. Our framework requires unforgeability and balance, fair distribution of donations, and donation and recipient privacy. Then, we constructed the first e-donation scheme that provably satisfies the requirements of the framework. We now propose some future directions for improvement of the e-donation framework and protocol.

Fairness. Our notion, fair distribution of donations, is based on equality of receivable donations by each user. Our work leaves improved definitions of fairness (e.g., adjustments based on location) as future work. We acknowledge that complete achievement of fairness requires replacing all ICOs with an e-donation system. This is unlikely to occur all at once in the near future and likely to require a gradual transition period.

Attribute Authorities. In this work, as a design choice, we do not distribute the task of issuing an attribute (e.g., threshold issuing) among multiple authorities due to the trade-off of the added benefit vs. requiring a user to have multiple interactions for each attribute (resulting in lower efficiency), and to reveal his identity to multiple authorities (increasing the risk of identity disclosure). We leave as future work design of e-donation frameworks and schemes that do not rely on single authorities, yet still achieve fine efficiency and privacy features.

Efficiency. The efficiency of our e-donation scheme is mainly affected by the underlying PrivCash and VABS schemes. Besides efficiency gains from relax-

ations of the framework requirements, improvements on both computation time and bandwidth use in both primitives will directly improve the efficiency of our scheme. Moreover, it may be possible to design a more efficient e-donation protocol that is independent of these building blocks.

3.8 Security Proof of Our Scheme

We now show that our e-donation scheme satisfies the security requirements of our e-donation framework, and prove Theorem 1 by providing partial proofs for each of unforgeability and balance, fair distribution of donations, and donation and recipient privacy definitions given in the generic e-donation framework in Section 3.3 via formal reductions to the security assumptions on the underlying primitives.

3.8.1 Proof of Unforgeability and Balance

Lemma 1. *If the underlying* PrivCash *satisfies the balance and non-malleability property (see Section 3.2), the digital signature scheme utilized for signing* PrivCashMint *is existentially unforgeable under an adaptive chosen-message attack [Katz and Lindell, 2007], and the underlying* VABS *protocol satisfies signature unforgeability defined in [Biçer and Küpçü, 2019] (see Section 3.2); then our e-donation scheme is unforgeable and balanced.*

Proof. Applying the definition of unforgeability and balance to our scheme, the adversary can win by generating one of the following transactions:

- a $\mathsf{TX}_{\mathsf{Donate},1}$ whose references include at least one of the transactions $(v_1, p_1), \ldots, (v_\ell, p_\ell)$ of honest users $\big($corresponding to the winning condition (1)$\big)$,

- a $\mathsf{TX}_{\mathsf{Donate},2}$ that spends a PrivCash other than an unspent one of the donor of the related $\mathsf{TX}_{\mathsf{Donate},1}$, or that can replace an existing transaction $\mathsf{TX}'_{\mathsf{Donate},2}$ with a different policy on the Ledger generated by a user under $\mathcal{C}$'s control (and the verification algorithm verifies all the transactions on the Ledger

upto and including $\mathsf{TX}_{\mathsf{Donate},2}$) (corresponding to the winning condition (2), since the amount of the minted PrivCash by $\mathsf{TX}_{\mathsf{Donate},1}$ cannot exceed the total amount of its referenced BaseCash due to the scanning check by the Ledger writer and the output only adds to the donations with the related β),

- a $\mathsf{TX}_{\mathsf{Claim},1}$ that references a $\mathsf{TX}_{\mathsf{Donate},2}$ with policy $\beta \wedge \omega$ (corresponding to the winning condition (3)),

- a $\mathsf{TX}_{\mathsf{Claim},2}$ that spends a PrivCash other than an unspent one of the recipient of the related $\mathsf{TX}_{\mathsf{Claim},1}$, or that can replace an existing transaction tuple $\mathsf{TX}'_{\mathsf{Claim},2}$ on the Ledger generated by a user under $\mathcal{C}'$s control such that $\mathsf{TX}'_{\mathsf{Claim},2} \neq \mathsf{TX}_{\mathsf{PrivateDonate}}$ (and the verification algorithm verifies all the transactions on the Ledger upto and including $\mathsf{TX}_{\mathsf{Donate},\mathsf{k}}$) (corresponding to the winning condition (4), since the amount of the minted PrivCash by $\mathsf{TX}_{\mathsf{Claim},1}$ cannot exceed the total amount of its referenced $\mathsf{TX}_{\mathsf{Donat2},2}$ due to the scanning check by the Ledger writer and the output only adds to the balance of the payee p).

If a PPT adversary $\hat{\mathcal{A}}$ wins the game eDonForge with non-negligible probability, we can utilize it to construct a PPT algorithm $\hat{\mathcal{B}}$ who wins the signature existential unforgeability game (Sig-forge in [Katz and Lindell, 2007]) or the BalanceTNM game or the VABSForge game with non-negligible advantage. In eDonForge, $\hat{\mathcal{A}}$ plays the role of $\mathcal{A}$, and $\hat{\mathcal{B}}$ plays the role of $\mathcal{C}$. In Sig-forge, BalanceTNM, and VABSForge games, $\hat{\mathcal{B}}$ plays the role of $\mathcal{A}$. We denote the honest challengers of them as $\hat{\mathcal{C}}_f$, $\hat{\mathcal{C}}_b$, and $\hat{\mathcal{C}}_a$, respectively.

1. In Sig-forge, using the security parameter 1^λ, $\hat{\mathcal{C}}_f$ generates a key pair (p, s). It then gives 1^λ, p, and access to signing oracle with s to $\hat{\mathcal{B}}$.

2. In BalanceTNM, $\hat{\mathcal{B}}$ receives 1^λ (assumed to be the same as in Step 1 above), the unit donation amount u, $pparams$, and query access to $\mathcal{O}$, from $\hat{\mathcal{C}}_b$.

3. In VABSForge, $\hat{\mathcal{B}}$ receives 1^λ (assumed to be the same as in Step 1 above), n, δ, $vparams$ (Step 1 of VABSForge).

4. In eDonForge, $\hat{\mathcal{B}}$ generates $(\dot{s}, \dot{p})$ gives to $\hat{\mathcal{A}}$ the values 1^λ, $dparams := (vparams, pparams, \dot{s}, \dot{p})$, and read access to all Ledger content. $\hat{\mathcal{B}}$ gives u, $D := n \cdot u$, and δ to $\hat{\mathcal{A}}$ (Step 1 of eDonForge).

5. In eDonForge, $\hat{\mathcal{B}}$ follows Step 2 of the game with $\hat{\mathcal{A}}$ in the exact same way as an honest $\mathcal{C}$ would do, except for the following: Whenever $\hat{\mathcal{A}}$ requests from $\hat{\mathcal{B}}$ an identity provider or attribute authority oracle generation, in VABSForge, $\hat{\mathcal{B}}$ requests an authority from $\hat{\mathcal{C}}_a$. $\hat{\mathcal{B}}$ obtains the public key of the authority from $\hat{\mathcal{C}}_a$, and gives it to $\hat{\mathcal{A}}$ as response. Further, if $\hat{\mathcal{A}}$ request him to generate users, $\hat{\mathcal{A}}$ queries $\hat{\mathcal{C}}_b$ with PrivCashKeyGen, and forwards the output to $\hat{\mathcal{A}}$. Whenever $\hat{\mathcal{A}}$ queries AttrGen or RecipJoin oracle, $\hat{\mathcal{B}}$ queries the related authority in VABSForge (Step 2 of VABSForge). $\hat{\mathcal{B}}$ obtains the output, and returns it back to $\hat{\mathcal{A}}$. We note that $\hat{\mathcal{B}}$ keeps a table that matches each user public key generated by PrivCashKeyGen query in BalanceTNM and the secret key generated by $\hat{\mathcal{B}}$ for RecipJoin. For generation of BaseCash by the users under $\hat{\mathcal{B}}$'s control, $\hat{\mathcal{B}}$ generates them itself, and queries $\mathcal{O}$ with Insert of BaseCash. For generation of $\mathsf{TX}_{\mathsf{Donate},1}$, $\mathsf{TX}_{\mathsf{Donate},2}$, and $\mathsf{TX}_{\mathsf{Claim},2}$ by the users under $\hat{\mathcal{B}}$'s control, $\hat{\mathcal{B}}$ straightforwardly queries $\mathcal{O}$ with types PrivCashMint, PrivCashSpend (with $\dot{p}$), and PrivCashSpend, respectively. It then writes the transactions generated on the Ledger of BalanceTNM to the Ledger of eDonForge. Regarding generation of $\mathsf{TX}_{\mathsf{Claim},1}$, first $\hat{\mathcal{B}}$ queries $\mathcal{O}$ with PrivCashMint (with $\ldots s$), and obtains the resulting transaction as $\mathsf{TX}_{\mathsf{PrivCashMint}}$. $\hat{\mathcal{B}}$ then generates VABS σ' on $\mathsf{TX}_{\mathsf{PrivCashMint}}$, and writes the resulting transaction $(\mathsf{TX}_{\mathsf{PrivCashMint}}, \sigma')$ to the Ledger of eDonForge. The transactions BaseCash, $\mathsf{TX}_{\mathsf{Donate},1}$, $\mathsf{TX}_{\mathsf{Donate},2}$, and $\mathsf{TX}_{\mathsf{Claim},2}$ generated by $\hat{\mathcal{A}}$ are inserted (i.e., via Insert queries) to the Ledger of BalanceTNM as BaseCash PrivCashMint, PrivCashSpend, and PrivCashSpend, respectively. Regarding Insert with $\mathsf{TX}_{\mathsf{Claim},1}$, first $\hat{\mathcal{B}}$ drops VABS, and then queries $\mathcal{O}$ for Insert of

the resulting transaction (Step 1 of BalanceTNM).

6. In eDonForge, $\hat{\mathcal{B}}$ generates some BaseCash tuples $(v_1, p), (v_2, p_2), \ldots, (v_\ell, p_\ell)$ with $\hat{\mathcal{A}}$'s request, and writes them to the Ledger, while keeping the secret keys private (Step 3 of eDonForge).

7. In eDonForge, $\hat{\mathcal{A}}$ returns ω to $\hat{\mathcal{B}}$ (Step 4 of eDonForge).

8. In VABSForge, $\hat{\mathcal{B}}$ gives ω to $\hat{\mathcal{C}}_a$ (Step 3 of VABSForge).

9. Step 4 of VABSForge is conducted by $\hat{\mathcal{B}}$ by forwarding messages between $\hat{\mathcal{C}}_a$ and $\hat{\mathcal{A}}$.

10. In eDonForge, $\hat{\mathcal{A}}$ eventually outputs a transaction $\mathsf{TX}_{\mathsf{Donate},1}$ whose references include a subset of $\{(v_1, p), (v_2, p_2) \ldots, (v_\ell, p_\ell)\}$, or a transaction $\mathsf{TX}_{\mathsf{Donate},2}$, or a transaction $\mathsf{TX}_{\mathsf{Claim},1}$ that references a $\mathsf{TX}_{\mathsf{Donate},2}$ with policy $\beta \wedge \omega$, or a transaction $\mathsf{TX}_{\mathsf{Claim},2}$. $\hat{\mathcal{A}}$ sends this transaction to $\hat{\mathcal{B}}$ (Step 5 of eDonForge).

    (a) If this transaction is a $\mathsf{TX}_{\mathsf{Donate},1}$, assuming that it particularly references (v_1, p), in Sig-forge, $\hat{\mathcal{B}}$ parses $\mathsf{TX}_{\mathsf{Donate},1}$ for the digital signature σ and the signed transcript $(\mathsf{TX}_{\mathsf{Donate},1'})$, possible as $\mathsf{TX}_{\mathsf{Donate},1}$ is obtained via a plain PrivCashMint operation.

    (b) If this transaction is a $\mathsf{TX}_{\mathsf{Donate},2}$ or $\mathsf{TX}_{\mathsf{Claim},2}$, it is output as the transaction $\mathsf{TX}'_{\mathsf{PrivCashSpend}}$ in BalanceTNM (Step 2 of BalanceTNM).

    (c) If this transaction is a $\mathsf{TX}_{\mathsf{Claim},1}$, in VABSForge, $\hat{\mathcal{B}}$ parses $(\mathsf{TX}_{\mathsf{Claim},1'}, \sigma') := \mathsf{TX}_{\mathsf{Claim},1}$ and outputs $(\mathsf{TX}_{\mathsf{Claim},1'}, \beta \wedge \omega, \sigma')$ (Step 5 of VABSForge).

Regarding the case (a), the output of the game Sig-forge is exactly the same as that of eDonForge. Namely, $\hat{\mathcal{B}}$ wins Sig-forge with non-negligible advantage, if $\hat{\mathcal{A}}$ wins eDonForge with non-negligible advantage.

Regarding the case (b), if $\hat{\mathcal{A}}$ can win eDonForge with non-negligible probability, then it succeeds in casting an imbalanced $\mathsf{TX}_{\mathsf{Donate},2}$ or $\mathsf{TX}_{\mathsf{Claim},2}$ whose amount exceeds the balance of BaseCash tuples referenced, or in forging an existing one of its type. The output of the game BalanceTNM is exactly the same as that of eDonForge. Namely, $\hat{\mathcal{B}}$ wins BalanceTNM with non-negligible advantage, if $\hat{\mathcal{A}}$ wins eDonForge with non-negligible advantage.

Regarding the case (c), due to direct referencing and check by the Ledger writer, $\hat{\mathcal{A}}$ cannot generate an invalid Claim whose amount exceeds u. The only way for $\hat{\mathcal{A}}$ to win eDonForge with non-negligible probability is claiming a $\mathsf{TX}_{\mathsf{Donate},2}$ without having the policy requirement. The output of the game VABSForge is exactly the same as that of eDonForge. Namely, $\hat{\mathcal{B}}$ wins VABSForge with non-negligible advantage, if $\hat{\mathcal{A}}$ wins eDonForge with non-negligible advantage.

All in all, if $\hat{\mathcal{A}}$ wins eDonForge with non-negligible advantage, then $\hat{\mathcal{B}}$ wins Sigforge, or BalanceTNM, or VABSForge with non-negligible advantage, concluding the proof.

□

3.8.2 *Proof of Fair Distribution of Donations*

Lemma 2. *If the underlying* VABS *protocol satisfies signature soundness defined in [Biçer and Küpçü, 2019] (see Chapter 2), then our e-donation scheme provides fair distribution of donations.*

Proof. We now reduce the fair distribution of donations in our e-donation scheme to the signature soundness of the underlying VABS.If a PPT adversary $\hat{\mathcal{A}}$ wins the game FairDist with non-negligible advantage, then we can utilize $\hat{\mathcal{A}}$ to construct a PPT algorithm $\hat{\mathcal{B}}$ who wins the game VABSSound with non-negligible advantage. In the FairDist game, $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ play the roles of $\mathcal{A}$ and $\mathcal{C}$, respectively. $\hat{\mathcal{B}}$ also plays the role of $\mathcal{A}$ against the honest challenger $\hat{\mathcal{C}}$ in VABSSound.

1. In VABSSound, $\hat{\mathcal{B}}$ receives 1^λ, n, δ, and *vparams* from $\hat{\mathcal{C}}$ (Step 1 of VABSSound).

2. In FairDist, $\hat{\mathcal{B}}$ itself picks an arbitrary $u \in poly(\lambda)$ and generates $pparams$ and $(\dot{s}, \dot{p})$ using 1^λ. $\hat{\mathcal{B}}$ gives to $\hat{\mathcal{A}}$ the values 1^λ, $D := n \cdot u$, δ, $dparams := (vparams, pparams, \dot{s}, \dot{p})$, and read access to all Ledger content (Step 1 of FairDist).

3. In FairDist, $\hat{\mathcal{B}}$ follows Step 2 of the game with $\hat{\mathcal{A}}$ in the exact same way as an honest $\mathcal{C}$ would do, except for the following: Whenever $\hat{\mathcal{A}}$ requests from $\hat{\mathcal{B}}$ an attribute authority oracle generation, in VABSSound, $\hat{\mathcal{B}}$ requests an attribute authority from $\hat{\mathcal{C}}$. $\hat{\mathcal{B}}$ obtains the public key of the authority from $\hat{\mathcal{C}}$, and gives it to $\hat{\mathcal{A}}$ as response. Similarly, whenever $\hat{\mathcal{A}}$ queries AttrGen oracle, $\hat{\mathcal{B}}$ queries the related authority in VABSSound (Step 2 of VABSSound). $\hat{\mathcal{B}}$ obtains the output, and returns it back to $\hat{\mathcal{A}}$.

4. In FairDist, if $\hat{\mathcal{A}}$ wins (i.e., $\hat{\mathcal{A}}$ comes up with a transaction tuple for $\mathsf{PrivateClaim}_{\hat{\mathcal{A}}}$ whose recipient with secret key s has claimed at least D amount in the current time period, and the transaction tuple is verified for being written to the Ledger), $\hat{\mathcal{B}}$ parses each $\mathsf{TX}_{\mathsf{Claim},1} = (r, c, \sigma)$ transaction that is written to the Ledger as (r, c) and VABS σ (Step 3 of FairDist).

5. In VABSSound, $\hat{\mathcal{B}}$ gives the message (r, c) and VABS σ pairs to $\hat{\mathcal{C}}$ for verification (Step 3 of VABSSound).

If $\hat{\mathcal{A}}$ can win FairDist with non-negligible probability, then it succeeds in minting (via $\mathsf{TX}_{\mathsf{Claim},1}$ transactions) more than $n \cdot u$ amount PrivCash for a specific user (i.e., signing more than n messages with the same s that gets verified). The output of the game VABSSound is exactly the same as that of FairDist, since Verify algorithm run on the message and ABS pairs of those $\mathsf{TX}_{\mathsf{Claim},1}$s outputs 1. Namely, $\hat{\mathcal{B}}$ wins VABSSound with non-negligible advantage if $\hat{\mathcal{A}}$ wins FairDist with non-negligible advantage.

□

3.8.3 Proof of Donation and Recipient Privacy

Lemma 3. *If the underlying* PrivCash *satisfies the ledger indistinguishability of [Sasson et al., 2014], and the underlying* VABS *protocol satisfies anonymity defined in Chapter 2; then our e-donation scheme provides donation and recipient identity privacy.*

Proof. We utilize the following deductions from the Lemma 3. Its contrapositive:

If our e-donation scheme does not provide donation and recipient privacy; then the underlying PrivCash does not satisfy the ledger indistinguishability of [Sasson et al., 2014], or the underlying VABS protocol does not satisfy anonymity defined in [Biçer and Küpçü, 2019] (see Chapter 2).

From the above form of Lemma 3, we can obtain:

If our e-donation scheme does not provide donation and recipient privacy, and the underlying VABS protocol satisfies anonymity defined in [Biçer and Küpçü, 2019] (see Chapter 2); then the underlying PrivCash does not satisfy the ledger indistinguishability of [Sasson et al., 2014].

Assuming that the underlying VABS satisfies anonymity of [Biçer and Küpçü, 2019], we now reduce the donation and recipient privacy of our e-donation scheme to the ledger indistinguishability of the underlying PrivCash. If a PPT adversary $\hat{\mathcal{A}}$ wins the donation and recipient privacy game Privacy with non-negligible advantage, then we can utilize $\hat{\mathcal{A}}$ to construct a PPT algorithm $\hat{\mathcal{B}}$ that wins ledger indistinguishability game L-IND with non-negligible advantage. In the game Privacy, $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ play the roles of $\mathcal{A}$ and $\mathcal{C}$, respectively. In the L-IND game, $\hat{\mathcal{B}}$ plays the role of $\mathcal{A}$ against an honest challenger $\hat{\mathcal{C}}$.

1. In L-IND, Steps 1, 2, and 3 of the game are followed between $\hat{\mathcal{B}}$ and $\hat{\mathcal{C}}$.

2. In Privacy, $\hat{\mathcal{B}}$ gives to $\hat{\mathcal{A}}$ 1^λ that is obtained in L-IND. $\hat{\mathcal{B}}$ runs the setup operation of VABS to generate $vparams$ and generates $(\dot{s}, \dot{p})$. Then, $\hat{\mathcal{B}}$ sets $dparams = (vparams, pparams, \dot{s}, \dot{p})$ and gives $dparams$ to $\hat{\mathcal{A}}$. $\mathcal{A}$ is also given the unit donation amount u obtained in L-IND, the total donation amount $D = n \cdot u$ that can be received by a recipient in a given time period where

$n \in poly(\lambda)$, and the duration δ of a time period. Time period counter t is initialized as $t := 1$, and is started (Step 1 of Privacy).

3. In Privacy, $\hat{\mathcal{B}}$ generates all the authorities and runs their oracles itself. Regarding UserJoin for the users under $\hat{\mathcal{B}}$'s control $\hat{\mathcal{B}}$ queries both Q_0 and Q_1 as PrivCashKeyGen in L-IND. Regarding RecipJoin for the users under $\hat{\mathcal{B}}$'s control, $\hat{\mathcal{B}}$ generates the recipient secret key S itself. $\hat{\mathcal{B}}$ keeps a table that matches each user public key generated by PrivCashKeyGen query in L-IND and the secret key generated by $\hat{\mathcal{B}}$ for RecipJoin (Step 2 of Privacy).

4. In Privacy, $\hat{\mathcal{B}}$ sets $\mathsf{Ledger}_{\mathsf{left}}$ and $\mathsf{Ledger}_{\mathsf{right}}$ essentially as $\mathsf{Ledger}_{\mathsf{left}}$ and $\mathsf{Ledger}_{\mathsf{right}}$ of L-IND (Step 3 of Privacy).

5. $\hat{\mathcal{B}}$ forward $\mathsf{Donate},1$, $\mathsf{Donate},2$, and $\mathsf{Claim},2$ queries in Privacy as PrivCashMint, PrivCashSpend (with $\dot{p}$), and PrivCashSpend queries to $\hat{\mathcal{C}}$ in L-IND without changing the order of (Q_0, Q_1), respectively. $\hat{\mathcal{B}}$ then writes to $\mathsf{Ledger}_{\mathsf{left}}$ and $\mathsf{Ledger}_{\mathsf{right}}$ in Privacy the resulting transaction on $\mathsf{Ledger}_{\mathsf{left}}$ and $\mathsf{Ledger}_{\mathsf{right}}$ in L-IND, respectively. Regarding BaseCash queries for the users under $\hat{\mathcal{B}}$'s control, $\hat{\mathcal{B}}$ generates them itself, and queries in L-IND with Insert of BaseCash without changing the order of (Q_0, Q_1). Regarding $\mathsf{Claim},1$ queries, first $\hat{\mathcal{B}}$ queries them in L-IND as PrivCashMint with $\dot{s}$ without changing the order of (Q_0, Q_1). $\hat{\mathcal{B}}$ obtains the resulting transactions $\mathsf{TX}_{\mathsf{PrivCashMint,left}}$ and $\mathsf{TX}_{\mathsf{PrivCashMint,right}}$ in L-IND. $\hat{\mathcal{B}}$ then generates VABSs σ'_{left} and σ'_{right} on $\mathsf{TX}_{\mathsf{PrivCashMint,left}}$ and $\mathsf{TX}_{\mathsf{PrivCashMint,right}}$, and writes the resulting transactions $\mathsf{TX}_{\mathsf{Claim,left}} := (\mathsf{TX}_{\mathsf{PrivCashMint,left}}, \sigma'_{\mathsf{left}})$ and $\mathsf{TX}_{\mathsf{Claim,left}} := (\mathsf{TX}_{\mathsf{PrivCashMint,right}}, \sigma'_{\mathsf{right}})$ to the $\mathsf{Ledger}_{\mathsf{left}}$ and $\mathsf{Ledger}_{\mathsf{right}}$ of Privacy. The transactions BaseCash, $\mathsf{TX}_{\mathsf{Donate},1}$, $\mathsf{TX}_{\mathsf{Donate},2}$, and $\mathsf{TX}_{\mathsf{Claim},2}$ inserted (i.e., via Insert queries) by $\hat{\mathcal{A}}$ to the Ledger of Privacy are inserted to the Ledger of L-IND as BaseCash PrivCashMint, PrivCashSpend, and PrivCashSpend, respectively. Regarding Insert with $\mathsf{TX}_{\mathsf{Claim},1}$s, first $\hat{\mathcal{B}}$ drops VABSs, and then queries for Insert of the resulting transactions in L-IND without changing the order of

(Q_0, Q_1) (Step 4 of Privacy).

6. $\hat{\mathcal{B}}$ eventually outputs in Privacy the bit that $\hat{\mathcal{A}}$ eventually outputs in Privacy.

In the above scenario, $\hat{\mathcal{B}}$ does not directly pick a bit, but implicitly sets it as the bit that is picked by $\hat{\mathcal{C}}$. We highlight that $\mathsf{Ledger}_{\mathsf{left}}$ and $\mathsf{Ledger}_{\mathsf{right}}$ of Privacy and those of L-IND given here are only different in terms of VABSs of the transactions $\mathsf{TX}_{\mathsf{Claim},2}$. As the underlying VABS scheme is assumed to satisfy anonymity of [Biçer and Küpçü, 2019], if $\hat{\mathcal{A}}$ wins the game Privacy, then $\hat{\mathcal{B}}$ wins the game L-IND. If $\hat{\mathcal{A}}$ wins the game Privacy with $1/2$ plus non-negligible probability, then $\hat{\mathcal{B}}$ wins the game L-IND with $1/2$ plus non-negligible probability. We note that in case the underlying Privacy scheme is assumed to satisfy L-IND, it is trivial to utilize an adversary obtaining non-negligible advantage in Privacy in construction of one obtaining that in VABSAnonym. □

We note that our security definitions and construction do not model timing attacks (e.g., the ones in [Tramèr et al., 2020] for de-anonymization), as in Zerocoin [Miers et al., 2013] and Zerocash [Sasson et al., 2014]. Although a full solution to such side channel attacks is out of scope of this chapter, one needs to ensure to have resolved them before deployment of e-donation.

	donor privacy	**recipient privacy**	**fair distribution**	**decentralization**	**always-on**	**security**	**scalability**	**accountability**	**revocability**
ICO	✓*	✓*	✓*		✓*	✓*	✓*	✓*	✓*
NCC				✓	✓	✓	✓	✓	
ACC	✓	✓		✓	✓	✓	✓	✓	
PSC	✓	✓		✓	✓	✓	✓	✓	
Our scheme	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 3.1: Comparison of traditional international charity organizations (ICO), the non-anonymous cryptocurrency (NCC) schemes (e.g., Bitcoin [Nakamoto, 2008] and Ethereum [Buterin, 2013]), the anonymous cryptocurrency (ACC) schemes (e.g., Monero [Monero, 2014], Zerocoin [Miers et al., 2013], and Zerocash [Sasson et al., 2014]), the private smart contract (PSC) schemes (e.g., Hawk [Kosba et al., 2016], Arbitrum [Kalodner et al., 2018], Zether [Bünz et al., 2019], ZEXE [Bowe et al., 2020], and Enigma [Eni, 2019]) and our e-donation scheme (see Section 3.4 in terms of adequacy of e-donation requirements. ✓denotes satisfaction of a requirement. By ✓*, we refer the facts that the traditional ICOs can only provide donor and recipient privacy in a limited way, as they do learn the identities of the parties (but they may choose to hide this information when trusted); and that fair distribution of donations in these schemes is limited, as recipients can obtain donations from various ICOs; and that their security, being always-on, scalability, accountability, and revocability depend on the honesty of a single organization for all these tasks to be carried out.

Chapter 4

BLOCKSIM-NET: A NETWORK-BASED BLOCKCHAIN SIMULATOR

4.1 Introduction

Proposed by Nakamoto[1], blockchain has applications in cryptocurrencies (e.g., Bitcoin, Ethereum[2], Litecoin[3]), certificate transparency[4] [Etemad and Küpçü, 2015], governmental services [David et al., 2019], etc. This is due to its transparency and immutability features [David et al., 2019, Singh et al., 2019, Mashamba-Thompson and Crayton, 2020]. Blockchains are maintained by peer-to-peer (P2P) networks, composed of nodes called "miners". Miners are incentivized by a competition through a process of mining, i.e., a trial-and-error process conducted via invested computational resources. Nakamoto's security assumption for Bitcoin has been that as long as a majority of the computational resources belongs to honest miners, the system will award each miner with their fair share. In other words, the reward that each miner receives will be proportional to their invested resources or "hashing power". [Garay et al., 2015] has shown that the blockchain shows immutability of the history if honest miners are majority. However, in 2013, Eyal and Sirer [Eyal and Sirer, 2018] showed that an adversarial miner can obtain more than their fair share even without having a

[1]S. Nakamoto (2008). Bitcoin: A peer-to-peer electronic cash system. Website `https://bitcoin.org/bitcoin.pdf` [accessed 15 April 2021].

[2]V. Buterin (2013). Ethereum White Paper: A next-generation smart contract and decentralized application platform. Website `https://ethereum.org/en/whitepaper/` [accessed 15 April 2021].

[3]Website `https://litecoin.org/` [accessed 15 April 2021].

[4]Website `http://www.certificate-transparency.org/` [accessed 15 April 2015].

majority of the total hashing power. Since then, there has been extensive research to optimize this attack [Sapirshtein et al., 2017, Nayak et al., 2016], and to defend against it [Heilman, 2014, Zhang and Preneel, 2017][5]. The proposal of these improvements clarified the need for a dependable environment to conduct the tests. However, real blockchain systems are too large and complex for these purposes.

To clarify the effects of selfish mining attacks and defences, some general simulation environments have been developed in recent times [Stoykov et al., 2017, Aoki et al., 2019, Alharby and van Moorsel, 2020]. Although it is possible to show these effects theoretically as well, it is generally reasonable to use tools for support. Also, sometimes we see that mathematical analysis is complicated to perform or error-prone. Among these environments, we focus on BlockSim [Alharby and van Moorsel, 2020], which is a versatile blockchain simulator. While BlockSim might suffice for certain security tests, since it simulates everything on a single CPU, it fails to be a network-based simulator. A network-based platform would provide a more realistic simulation of the real blockchain environments since network delays will be automatically integrated under real network conditions. This can also help simulation of other attacks and issues associated with the network itself, e.g., network randomness due to propagation delays, and oracle and bold mining attacks [5]. Factoring these ideas, we propose BlockSim-Net, a simple, easy-to-use, highly-efficient, network-based blockchain simulator for use in blockchain security research on both attacks and defences. Additionally, our simulator can also be used for beta-testing and optimization benchmarking.

4.1.1 Our contributions

We build on top of an existing work, BlockSim [Alharby and van Moorsel, 2020], which simulates a blockchain on a local system. The contribution of this work is two-fold:

[5] O. Biçer and A. Küpçü (2020). FORTIS: Selfish Mining Mitigation by (FOR)geable (TI)me(S)tamps. Website `https://eprint.iacr.org/2020/1290.pdf` [accessed 15 April 2021].

1. **From Local to Network:** Our main contribution is converting the BlockSim's [Alharby and van Moorsel, 2020] local simulation to a network simulation. As opposed to a local simulation, a network simulation is more distributed, decentralized, and comparable to a real blockchain, since multiple miners from different locations can be involved in the simulation of a blockchain network together. Such a simulator can also be used to effectively simulate network-based attacks.

2. **Efficiency:** Since simulation on the network comes with associated heavy communication costs, our work also includes multiple optimizations. The main optimizations are mentioned below:

   (a) *Switching to another miner's blockchain when a block is received:* Since [Alharby and van Moorsel, 2020] simulates on a single system, miner nodes can easily access the local blockchains of other miner nodes. Over a network, however, to switch to another miner's blockchain, a node would need to receive the entire blockchain of that miner. This would mean high communication costs. Thus, instead of that, we propose the use of a local dictionary that stores all the blocks that the miner receives during the simulation. Most of the time, using this local dictionary, miners can recreate the chain that was supposed to be received with minimal communication. However, in the case where a required block is lost in communication, recreating the blockchain may not be possible.

   (b) *Consensus:* In [Alharby and van Moorsel, 2020], since the entire simulation happens on a single system, the main thread accesses the longest local blockchains of all miners at the end of the simulation and decides on the longest chain. However, over a network of miners, each miner would have to send their longest local blockchain to the admin server for consensus (refer Section 4.3.7). However, the propagation of the entire blockchain would be very costly in terms of communication

	Paral.	Propagation of Blocks	Simulation	Scalability
BlockSim	✗	Simulation	Single CPU	Depends on the power of the CPU
SimBlock	✗	Simulation	Single CPU	Depends on the power of the CPU
VIBES	✓	Actual propagation	On a real blockchain	Bulky; hard to scale
Blockchain Demo	✗	Simulated	Web-based visualization	N/A
Talaria	✗	Simulated	Single CPU	Depends on the power of the CPU
BlockSim-Net	✓	Over multiple servers	Over a network	Depends on the power of the servers

Table 4.1: A qualitative comparison between state-of-the-art simulators. "Paral." is an abbreviation for "Parallelizable."

complexity. Instead, we propose that miners only send the last block of their longest local blockchain to the admin server for consensus. We elaborate on why the last block is enough for consensus in Section 4.3.7.

4.1.2 Related Work

Simulators. In the recent past, there have been several proposed blockchain simulators. SimBlock [Aoki et al., 2019], proposed in 2019, is a blockchain simulator in Java that is mainly designed to study the impact of node behaviour on the network. Their experiments have proven that SimBlock provides a good picture of a real blockchain. Similarly, BlockSim [Alharby and van Moorsel, 2020], proposed in 2020, is another blockchain simulator in Python that captures the network, incentive, and consensus layers of a blockchain. It is a modular, intuitive, and easy-to-use framework that allows the user to simulate a blockchain network of a multitude of nodes. Talaria[6], proposed in 2021, is another simulator that extends the capability of BlockSim by including permissioned-blockchains. However, it is important to note that SimBlock, BlockSim, and Talaria simulate the whole network on a single CPU. This means that the complications associated with the network and communication between nodes cannot be realistically captured. While [Alharby and van Moorsel, 2020, Aoki et al., 2019] and Talaria are highly scalable,

[6] J. Xing, D. Fischer, N. Labh, R. Piersma, B. C. Lee, Y. A. Xia, T. Sahai, and V. Tarokh (2021). Talaria: A Framework for Simulation of Permissioned Blockchains for Logistics and Beyond. Website `https://arxiv.org/abs/2103.02260` [accessed 15 April 2021].

BlockSim-Net is better for security analyses because it is a closer approximation of an actual blockchain network.

Some more recent blockchain simulators include VIBES [Stoykov et al., 2017] and Blockchain Demo[7]. VIBES is a simulator that allows users to specify configurations to create and simulate custom blockchains. It works on real networks, and this makes the simulation bulky, complex, and not scalable. Blockchain Demo is a web-based simulator that allows a user to modify transactions and node configurations on the interface, and thereby understand the inner workings of blockchains. SimBlock and VIBES can be used to understand Blockchain networks visually. Table 4.1 provides a comparative summary of existing simulators.

Selfish Mining. In the original Bitcoin paper, Nakamoto works on the assumption that as long as the majority of the hashing power belongs to the honest miners, every miner gets a reward that is proportional to their work. However, Eyal and Sirer [Eyal and Sirer, 2018] have introduced the concept of selfish mining, which allows a miner to earn more than her fair share, even when the honest majority assumption is true. The primary idea behind selfish mining is that a seemingly-honest miner can deviate from the honest protocol by populating a private chain until it becomes longer than the existing longest chain in the network. By extending on a private chain, a miner can potentially eliminate blocks in the network and get rewarded more than one's fair share. Several selfish mining defenses [Sapirshtein et al., 2017, Gervais et al., 2016, Nayak et al., 2016, Wang et al., 2019a] have also been proposed by various works against attacks proposed by [Eyal and Sirer, 2018, Heilman, 2014, Zhang and Preneel, 2017, Bissias and Levine, 2020]. In the original Bitcoin implementation, in case multiple chains with the same number of blocks occur, miners choose the one that they receive first. However, [Eyal and Sirer, 2018] proposes choosing one of the chains uniformly at random. This defense prevents attackers with less than 25% of the total hashing power of the network from obtaining high rewards. However, the later attack proposal by [Sapirshtein et al., 2017] on Optimal Selfish Mining reduces

[7]Website `https://andersbrownworth.com/blockchain/`, [accessed 30th April 2021].

this value to 23.2%. [Zhang and Preneel, 2017] proposes the publish or perish scheme and this takes the value back to 25%. Given this trend of proposals of selfish mining and network-based attacks and defenses, blockchain security research needs to constantly work to improve with better defenses. In this scenario, there arises a need for simulators that can simulate a real network as realistically as possible in order for defenses to be tried and tested.

Lastly, a preliminary version of this paper was presented at the BAŞARIM 2020 conference, but it was not published in any proceedings.

4.2 Preliminaries

Proof-of-Work Blockchains. A blockchain is a chain of blocks that contain information about various transactions happening in the network. Each block in the chain consists of the hash of the previous block (i.e., the parent block) in the chain, the depth of the current block, a nonce, and a difficulty target. In some models (e.g., GHOST protocol [Sompolinsky and Zohar, 2015]), a block may also refer to an uncle chain, which is a separate chain that comprises of orphaned blocks built on top of the parent block. Each miner in the network constantly attempts to mine a block. To mine a block, a miner should be able to create a block such that the hash of the block is lower than the difficulty target. The miner to achieve this first gets a reward. Creating a block such that the hash of the block is less than the difficulty target requires computational power and, consequently, a certain amount of time. The network is designed such that the miners adjust the difficulty target from time to time in order to ensure that the inter-block time is fixed even if the computational resources of the miners vary. For instance, in Bitcoin, the inter-block time is fixed to approximately 10 minutes. Therefore, every miner that is able to mine a block can be said to have done some "work", which is easy to verify. Since the network constantly updates the difficulty target based on the computational power of the miners, every miner that is able to mine a block is said to have demonstrated "proof-of-work".

BlockSim. BlockSim and BlockSim-Net deal with mainly three layers of

blockchain: the *network layer*, the *consensus layer*, and the *incentive layer*. The network layer mainly copes with the propagation of blocks and transactions in the blockchain system. The consensus layer ensures the protocol for honest nodes to come into consensus in the main chain. The incentive layer deals with miners' incentives for playing the honest mining strategy by using transaction fees and block rewards. The main differences of this work from BlockSim are on the simulation of the network and the consensus layers. Algorithm 7 describes BlockSim's simulation of a blockchain network.

In Algorithm 7, `current_time` refers to the current time in the simulation at any given point. It starts with 0 at the beginning and goes up to `sim_time`, which marks the end of the simulation. At the beginning of the simulation, the `current_time` is set to 0 and the hash power of each of the `num_nodes` nodes are randomly initialized. Afterwards, a transaction pool and a genesis block are created centrally for the simulation. Further, this genesis block is appended to each node's local blockchain. Later, each node starts mining on top of the genesis block. In this simulation, mining is essentially the creation of block "events". Based on a node's hashing power, BlockSim calculates a time `t` (in seconds) that would be sufficient to show proof of work. Therefore, when a block is said to be mined, an "event" is created with its timestamp set to `current_time + t`.

During the simulation, the block event that is scheduled for the earliest point in time is fetched from the queue by calling `get_next_event()`. The current time is then set to the timestamp associated with the event. If the block event is of a "create" type, then a block object is created and appended to the miner's local blockchain if and only if the block is built on top of the last block in the miner's local blockchain. This block is then sent to the other miners in the network. However, if this block event is of a "receive" type, all the miners (except the miner of this block event) verify whether certain conditions are met. In the case that the block satisfies these conditions, the miners append it to their local blockchain and start mining on top of it. This is done repeatedly until the simulation end time is reached.

Algorithm 7 Simulation in BlockSim.

```
procedure F_blocksim(num_nodes, sim_time)
    current_time = 0
    set_hash_power(num_nodes)
    create a transaction pool
    create genesis block and append to each node's local blockchain
    for node in all_nodes do
        mine(node, current_time)
    end for
    while (!empty(Queue) and current_time < sim_time) do
        event = get_next_event()
        current_time = event.time
        run_event(event)
        remove_from_queue(event)
    end while
    lb = get_longest_chain(all_nodes)
    set_global_blockchain(lb)
    print_consensus_stats()
end procedure
```

Once the simulation end time is reached, the longest blockchain is chosen centrally by comparing the local blockchains of all miners. Once the longest blockchain `lb` is found, the local blockchain of all miners is set to `lb`.

4.3 Simulation

BlockSim-Net is intended to be a simulator of a blockchain on a real network with different servers acting as miners. For implementation, we have split the structure of the simulator into two types of servers: 1) the *Admin Server*; and 2) the *Miner*. All miners on the network can deploy the miner code at their ends and run a simulation collectively. The admin server runs on a single node during the simulation.

4.3.1 The Admin Server (AS)

For our purposes, we have used an admin server (AS) that acts as a common point of contact to each of the miners. Every node connects to this server at the beginning of a simulation. AS, having received connections from the miners, collects information about them: miner ID, hash rate, IP, the port on which each of

the miners host their server socket during the simulation. We call this information *miner-info*. Being the common point of contact for each of the miners, AS also shares the information of other miners in the network. Each node can communicate with each other for the purposes of the simulation. Although, in reality, such an admin server is not present in a blockchain, here we need it for the sake of efficiency, and utilize it only at the start and end of the simulation.

We emphasize that AS only initiates the miners for the simulation and has no role in the creation and exchange of blocks in the simulation. AS assigns an ID to each of the miner nodes in order to make identification of the miner nodes easier. Once each of the nodes has been provided with miner-info, they are ready to run the simulation independently without any help from AS. The miner-info includes miner ID, the total hash rate of all the nodes in the network, and the IP and the port on which each of the miners will host their server socket during the simulation.

Apart from communicating information about other miners in the network with each node, AS is also responsible for the creation of the genesis block and creating a pool of transactions that are to be used by the miners for the duration of the simulation. As all miners have access to the genesis block and a transaction pool (with common transactions) in an actual network, the AS provides the genesis block and a pool of transactions to all the miners at the beginning of the simulation as a common starting point.

The next time that the AS is needed in the simulation is at the time of consensus, i.e., once the simulation is complete. While in a real blockchain, consensus happens as a result of communication between the miners themselves, here, it is important to remember that BlockSim-Net performs a timed simulation; a simulation has a start-time and an end-time. Due to this property of a simulator, there arises a need for an entity that can make up for a potential abrupt halt in the network. Thus, when the simulation end-time is reached, we use AS to ensure that the local blockchains of all miners are complete and uniform.

4.3.2 The Miner

The role of a miner is to create blocks on the network and propagate those to the other miners on the network. A miner also listens to the other miners on the network for blocks. Upon receiving blocks from other miners, a miner updates/maintains their blocks, based on some predefined rules. These rules differ from one another based on the type of blockchain or application.

Initially, each one of the miners connects to the admin server, AS. They learn miner-info from AS. As mentioned in the previous section, miner-info includes miner ID, the total hash rate of all the nodes in the network, and the IP and the port on which each of the miners hosts their server socket during the simulation. Once this basic information is acquired, the simulation timer starts running.

As soon as the simulation begins, AS creates a genesis block and a transaction pool and sends them to the miner nodes. Each miner node, upon receiving the genesis block and a pool of transactions, can start creating a new block at depth 1 (considering the genesis block is at depth 0). Each miner constantly listens for blocks from other miners on the network.

4.3.3 The Overview of the Simulation

The main steps of the simulation is shown in Figure 4.3.3. They can be summarized as follows:

- ① Miner connects to the admin server.
- ② Miner gets basic information from the admin server.
- ③ Miner starts the simulation.
- ④ Miner gets the Genesis Block and Transaction Pool from the admin server at the start of the simulation.
- ⑤ Miner starts the process of creating blocks and listening to blocks created by other miners on the network. With every new block (either created or

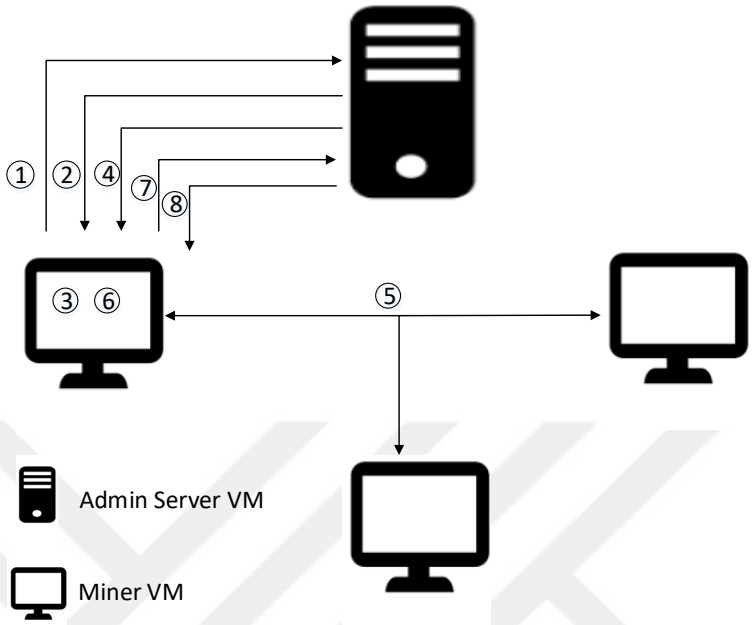


Figure 4.1: BlockSim-Net Simulation Flow. Descriptions of steps given inside circles are provided in Section 4.3.3.

received), Miner may update its local blockchain.

- ⑥ At the end of the simulation, Miner replaces the empty blocks (if any) in its longest chain with the blocks received.
- ⑦ Miner sends the last block of their longest chain to the admin server for consensus.
- ⑧ Based on the results of consensus, Miner gets the longest chain from the admin server (if its chain is not the longest).

In the rest of this section, We will describe them in detail.

4.3.4 *Initialization*

One of the main responsibilities of AS is to provide basic information to the miners in the network. This information includes the miner ID to uniquely identify

a miner, IP addresses and ports of all the miners on the network so that they can communicate with each other, and the total hash rate of the network, which is computed by the admin server by adding the individual hash rates of each miner. This basic information is referred to as miner-info in our paper. Additionally, at the start of the simulation, AS also provides a pool of transactions to be used for a simulation period and the genesis block to the miners at the start of each simulation. On the miner end, each of the miners randomly initializes themselves with a hash rate (a random sample from 0-30). After receiving their IDs from the admin server, the miners also initialize themselves with that value.

At the start of the simulation, each miner gets a genesis block and a pool of transactions from AS. After this, they start mining on top of the genesis block. The created blocks are stored in a 'create queue'. The miners also listen for new blocks from other miners on the network. The received blocks are stored in a 'receive queue'.

Algorithm 8 BlockSim-Net Simulation Algorithm For A Miner.

```
procedure F(txn_pool, genesis_block, node_info, sim_time)
    while current_time < sim_time do
        if (! empty(create_queue)) then
            block = getnextcreateblock()
            if current_time >= block.scheduled_time then
                update_local_blockchain(block,"create")
            end if
        end if
        if (! empty(receive_queue)) then
            block = getnextreceiveblock()
            F_update_local_blockchain(block,"recv")
        end if
    end while
    build_full_chain()
    consensus()
end procedure
```

In order to simulate proof of work in our blockchain network, we use the concept of "events" similar to [Alharby and van Moorsel, 2020]. In our implementation, each miner uses their (randomly generated) hash rate, h, in conjunction

with the value of the total hash rate obtained from AS to calculate the time (in seconds) a miner would take to create a block. Thus, the creation of a block, in our context, means the creation of a block object that is scheduled to be sent out into the network only when its blocktime (as calculated) is reached (or exceeded). As mentioned above, once created, these block objects are added to the 'create queue'.

The miner iteratively checks for blocks in the 'create queue' and the 'receive queue'. When a block is created, the miner appends it to its own blockchain if certain conditions are met (explained in the next section). On the other hand, if new blocks have been received, the miner checks if the blocks impose any further action to be taken on its blockchain. At the end of the simulation, the miner tries to build a full chain to account for any delays in the propagation of certain blocks. This has been explained in detail in Section 4.3.6. Finally, the miners send the last blocks in their local blockchains to the AS for consensus (explained in Section 4.3.7). This entire protocol has been illustrated in Algorithm 8.

4.3.5 Updating local blockchain

Any update to the local blockchain of a miner happens based on some predefined rules. A miner node may have to update its local blockchain in two cases: (a) when it creates a block; (b) when it receives a block from another miner node in the network. Algorithm 9 describes the protocol to update miners' local blockchain. We now analyse the two cases:

1. When it creates a block: As discussed above, in this paper, 'creation of a block' actually corresponds to the scheduling of an event. When a block object is created, it is appended to the miner's 'create queue'. Any action on the created block is only taken once the block object's blocktime has been reached or exceeded. As we have also discussed above, a miner needs to constantly check the blocks in its 'create queue' to see if the blocktime of the block (next_create_block) with the earliest timestamp in the queue has been reached (or exceeded). Once the blocktime of

Algorithm 9 Local Blockchain Update Procedure of BlockSim-Net.

```
procedure F_update_local_blockchain(block, flag)
    if flag = "create" then
        if ((myCH.last_block().depth < block.depth) and (myCH.last_block().id = block.previous)) then
            myCH.append(block)
            send_block(block)
            myCH.create_block_event()
        end if
    else
        if (block.previous = myCH.last_block().id) then // case 1: received block is built on my last block
            myCH.append(block)
            send_block(block)
            myCH.create_block_event()
        else// case 2: the received block is not built on my last block
            if (block.depth > myCh.depth) then
                myCH = construct_chain(block)
                myCH.create_block_event()
            else
                myUncleCH.append(block)
            end if
        end if
        receive_queue.remove(block)
    end if
end procedure
```

next_create_block has been reached (or exceeded), the miner looks at the last block (last_block) in its local blockchain. If the depth of last_block is less than that of next_create_block, the miner appends next_create_block to its local blockchain. After this process, next_create_block is removed from the queue. The miner can now start mining on top of next_create_block, which is now the last block of the updated blockchain. If a block with the same depth as next_create_block is received, it will be added to the miner's uncle chain.

2. When it receives a block: When a miner receives a block from another miner in the network, the block is first added to the 'receive queue'. During the simulation, the miner fetches the block (next_receive_block) with the earliest timestamp from the 'receive queue'. If the depth of this block is greater than that of the last block in the miner's local blockchain, the miner needs

to update its blockchain. If not, next_receive_block is added to the uncle chain.

If there is a need to update one's local blockchain, the following rules have to be followed:

- If a received block is built on top of the last block of the miner's longest local blockchain, the miner appends it to its longest chain. If the miner was already trying to mine a block at the same depth as the received block, it will discard that block. The miner, having updated its local blockchain, can now start mining on top of the received block.
- However, if the received block is not built on top of the last block of the miner's longest chain, the miner looks at its depth. If the depth of the received block is less than or equal to that of the last block of the miner's longest local blockchain, the miner adds the block to its uncle chain. However, if the depth of the received block is greater than that of the last block of the miner's longest local blockchain, the miner needs to switch to the chain of the miner from whom the block was received.

In Algorithm 9, the procedure `update_local_blockchain` is described. This procedure receives a block and a flag associated with it as inputs. Here, `myCh` refers to a miner's local blockchain, `block.depth` is the depth associated with a block, `myCH.last_block()` returns the last block in the miner's local blockchain, `block.previous` refers to the block ID of the block that this block was mined on top of, `myUncleCH` refers to the uncle chain which contains the orphaned blocks of the miner. Lastly, `construct_chain` is a method that will allow the miner to efficiently switch to another miner's blockchain, without having to receive that miner's entire blockchain. The details of this method have been described in the next section.

4.3.6 Switching to another miner's blockchain

A simulation entails multiple exchanges of blocks. However, not all of the blocks involved (created or received) make it to the longest chain of a miner immediately. However, because of consensus, many of these blocks make it to the miner's longest chain at some point with a considerable probability. Thus, if some blocks are discarded because they do not make it to the miner's longest blockchain at this point but they make it to the miner's longest blockchain at a point in the future, it does not make sense to discard them.

In our implementation, the miners locally store all the blocks that they create and receive from other miners on the network in a dictionary with the block IDs as keys and block objects as values. Thus, when a miner has to switch to another miner's chain, they trace back from the latest received block up to the genesis block. If the miner does not have some blocks in their local dictionary due to potential delays, we put temporary empty blocks up on their behalf. We then try to replace the empty blocks with the correct blocks, if available in the dictionary, once the simulation ends. This last step corresponds to the `build_full_chain` method given in Algorithm 8.

4.3.7 Consensus

At the end of the simulation, each miner sends the last block of their longest local blockchain to the admin server. The admin server determines the miner with the longest chain from the depth of these blocks. In case more than one miner has the longest blockchain, it considers the miner whose last block was created the earliest.

Once the admin server determines the miner with the longest chain, the admin server communicates with this miner to obtain its longest local chain. After that, the admin server sends it to all the other miners. If the miner with the longest chain has empty block objects that could not be replaced (due to the unavailability of those blocks) as part of its chain, that particular simulation is discarded.

This protocol has been described in Algorithm 10. According to this algorithm, the admin server first determines the miner whose blockchain is the longest, using the method $\mathcal{F}_{determine_accepted_miner}$. It then requests this miner to send its chain and stores it in `global_chain`. Finally, `global_chain` is sent to all miners with the help of the method `send_chain()`. The admin server also prints the statistics of the consensus at the end.

Algorithm 10 Consensus Algorithm of BlockSim-Net Executed by Admin Server.

```
procedure F_determine_accepted_miner(last_blocks)
    maxLength = -1 // length of the longest blockchain
    minerId = -1 // variable to store the ID of the miner with the longest chain
    for block in last_blocks do
        if (block.depth ≥ maxLength and block.timestamp < current_time) then
            maxLength = block.depth
            timestamp = block.timestamp
            minerId = block.minerId
        end if
    end for
    return minerId
end procedure

procedure F_consensus(last_blocks)
    minerId = F_determine_accepted_miner(last_blocks)
    global_chain = get_chain(minerId)
    send_chain(global_chain)
    print_consensus_stats()
end procedure
```

4.4 Results

Our simulator codebase is available at: `https://github.com/prashanthi-r/blocksim-net`. Our implementation is in Python 3.6, and it uses the following packages: `NumPy` (version 1.20.3), `SciPy` (version 1.6.3), `math`, `random`, `pickle`, `threading`, and `socket`. We present the results of our simulation for both Bitcoin and Litecoin using the recent market shares.

For Bitcoin, we have deducted the mining statistics of the top 6 pools (F2Pool, Poolin, BTC.com, AntPool, Huobi.pool, and ViaBTC) for the time interval be-

Mining Pools and Hash Percentage	Real mining share(%)	% of total blocks mined	
		BlockSim	BlockSim-Net
F2Pool	16.7	15.26	17.58
Poolin	13.7	15.06	12.43
BTC.com	11.5	12.95	11.41
AntPool	11.1	11.54	11.67
Huobi.pool	8.7	7.63	8.74
Binance Pool	8.7	7.44	7.57
Others	29.6	30.06	30.52

Table 4.2: Block percentage for major Bitcoin miners in a BlockSim-Net simulation. Expected block interval = 5 seconds; Simulation time = 10,000 seconds; Expected number of blocks for the simulation = 2000; Total number of blocks mined in BlockSim: 1559; Total number of blocks mined in BlockSim-Net: 1979.

tween 10 May 2020 and 10 May 2021 from btc.com[8]. We feed this information as the hashing powers for 7 miner nodes (6 pools + Others) for a 10,000-second simulation. The results have been presented in Table 4.2. Similarly, for Litecoin, we have deducted the mining statistics of 5 of its pools (LTC.top, AntPool, F2Pool, Litecoinpool.org, and ViaBTC) from btc.com[9]. We feed this information as the hashing powers for 6 miner nodes (5 pools + Others) for a 10,800-second simulation. The results have been presented in Table 4.3. We have run both BlockSim and BlockSim-Net simulations, with the latter on different ports of an Intel Core i5 8th Generation Processor with 4 cores and 8GB RAM.

In BlockSim-Net simulations, we observe that when the expected number of mined blocks in the simulation is higher, the deviation from the expected fair share (i.e., the hash power of each miner) is lower. Both Tables 4.2 and 4.3 demonstrate that every miner gets a fair share (total block percentage) that aligns with their hashing power. Note that Table 4.2 induces less deviation from the expected rewards compared to Table 4.3 because of a higher block interval. The expected

[8]Website `https://btc.com/stats/pools`, [accessed 10 May 2021].

[9]Website `https://miningpools.com/litecoin/`, [accessed 10 May 2021].

block interval and simulation times affect the result as they affect the expected number of blocks found in a simulation. As a result, for future experiments, we recommend a small expected block interval (typically 5 seconds) for fast simulation. We also recommend a simulation with at least 2,000 expected blocks for low deviation.

Comparison with BlockSim. To compare with BlockSim simulations, our results give similarly close rewards to hashing powers of miners. To quantify closeness, we use a standard deviation type of metric *reward deviation* or $\mathcal{R}_{dev}$ that we define as follows:

$$\mathcal{R}_{dev} = \sum_{i=1}^{N} (\alpha_i - \mathcal{R}_i^{sim})^2$$

where N is the total number of miners in the network, α_i is the hashing power of the i-th miner, and $\mathcal{R}_i^{sim}$ is the reward share that the i-th miner receives in a simulation. The lower this value, the more consistent will the reward be to the hashing power. By this metric, we calculate that for the Bitcoin simulation in Table 4.2, BlockSim-Net results in $\mathcal{R}_{dev} = 4.85$, and BlockSim results in $\mathcal{R}_{dev} = 9.16$, and for the Litecoin simulation in Table 4.3, BlockSim-Net results in $\mathcal{R}_{dev} = 13.29$, while BlockSim results in $\mathcal{R}_{dev} = 7.59$. Since $\mathcal{R}_{dev}$ for BlockSim and BlockSim-Net are of the same order in the two simulations, we infer that one can expect BlockSim and BlockSim-Net to give similar rewards for miners with given hashing powers.

Additionally, we define a metric *block ratio* or λ, which is the ratio of the number of blocks mined in a simulation to the expected number of blocks for the specified simulation time and block interval. This metric is a measure of the closeness of the mining process in a simulation to a *real* blockchain. Note that the higher the block ratio, the closer a simulation is to the real mining process. For the Bitcoin simulation, BlockSim produces $\lambda = 77.95$, while BlockSim-Net produces $\lambda = 98.95$. Further, for the Litecoin simulation, BlockSim produces $\lambda = 92.06$, while BlockSim-Net produces $\lambda = 95.74$. One can observe that BlockSim-Net captures the essence of real blockchain mining better than BlockSim in both simulations. We encapsulate the comparison between BlockSim and BlockSim-Net for both simulations in Table 4.4 based on the defined metrics: reward deviation

Mining Pools and Hash Percentage	Real mining share(%)	% of total blocks mined	
		BlockSim	BlockSim-Net
LTC.top	22.0	22.32	24.60
AntPool	20.0	20.07	21.24
F2Pool	19.0	21.07	16.44
Litecoinpool.org	14.0	13.84	13.20
ViaBTC	12.0	10.47	12.24
Others	13.0	12.09	12.12

Table 4.3: Block percentage for major Litecoin miners in a BlockSim-Net simulation. Expected block interval = 12.42 seconds; Simulation time = 10800 seconds; Expected number of blocks for the simulation = 869; Total number of blocks mined in BlockSim: 801; Total number of blocks mined in BlockSim-Net: 833.

Metrics	$\mathcal{R}_{dev}$		λ	
	BlockSim	BlockSim-Net	BlockSim	BlockSim-Net
Simulation in Table 4.2	9.16	4.85	77.95	98.95
Simulation in Table 4.3	7.59	13.29	92.06	95.74

Table 4.4: Comparison of the performance of BlockSim and BlockSim-Net based on reward deviation ($\mathcal{R}_{dev}$) and block ratio (λ) in the simulations in Tables 4.2 and 4.3.

and block ratio.

4.5 Conclusion

In this paper, we propose BlockSim-Net, a simulation framework for blockchain technologies on a real network with multiple communication optimizations. Our codebase in Python, as an extension of the implementation of BlockSim, has been divided into two aspects: the admin server and the miner. Future work can focus on improving the scalability of the system. Furthermore, while BlockSim-Net provides a codebase for proof-of-work blockchain networks, our work could be extended to other blockchain systems and simulate possible attacks and defence

mechanisms on these systems.

Chapter 5

FORTIS: SELFISH MINING MITIGATION BY (FOR)GEABLE (TI)ME(S)TAMPS

5.1 Introduction

First proposed by [Nakamoto, 2008] for Bitcoin, proof-of-work (PoW) blockchain plays a crucial role as the underlying technology behind modern cryptocurrencies (e.g., Bitcoin, Ethereum, and Litecoin) and many distributed and cloud-based applications (e.g., certificate transparency as in [Google, 2021], smart contracts as in [Chainlink, 2021], e-government as in [David et al., 2019], e-voting as in [Khan et al., 2021], online donations systems as in [Biçer and Küpçü, 2020a], smart appliances as in [Singh et al., 2019], and healthcare as in [Shi et al., 2020]). Blockchain technology is quite promising to cope with some prominent challenges that cloud computing is recently facing, e.g., data security, data management, compliance, reliability [Murthy et al., 2020]. PoW blockchain depends on an incentive based mechanism called *mining* rather than a central authority for its proper operation. Mining is an unremitting competition for finding and propagating the hash value of the next block that gets appended to the blockchain. *Miners* are investors on computational resources that are specialized for mining procedures. The investment may be in the form of buying or hiring the resources, or the miners can outsource the mining work to cloud services, in which case the process is called *cloud mining* (e.g., via Genesis Mining[1] or NiceHash[2]). When a miner *mines* a block, she receives a wealthy block reward, which builds up the incentive for her investment.

[1]https://www.genesis-mining.com

[2]https://www.nicehash.com

Selfish mining attack. Nakamoto claimed that as long as the majority (>50%) of hashing power in the system belongs to the miners that follow the correct mining algorithm, the reward of each miner would be proportional to her expenditures, securing the proper operation of the blockchain. Yet, [Eyal and Sirer, 2018] proposed the *selfish mining (SM) attack* that yields a miner more than her fair share by deviating from honest mining (i.e., following the publicly accepted correct algorithm), even if the honest majority assumption holds. The main idea of the attack is keeping the mined blocks private for further extending them individually, and releasing them at a later time for elimination of others blocks from the main chain. The attack works because the honest miners always choose the chain with more blocks and the difficulty adjusts over time. In fact, none of these causes seem avoidable, as the former is a countermeasure against network partitions and propagation delays, and the latter is an initial design choice to ensure the expected inter-block time of the main chain would be constant. The selfish mining attack is extensively studied in the recent blockchain literature, including the research for optimizing it as in [Sapirshtein et al., 2016, Gervais et al., 2016, Nayak et al., 2016, Wang et al., 2019a], combining it with other attacks (e.g., with eclipse attack as in [Nayak et al., 2016, Heilman et al., 2015] and block withholding attack as in [Dong et al., 2019]), and defending against it as in [Eyal and Sirer, 2018, Heilman, 2014, Zhang and Preneel, 2017, Bissias and Levine, 2020]. Although so far no known selfish mining attack occurred on Bitcoin, its practice would be harmful not only by reducing the fair shares of miners, but also by resulting in inconsistent views of blockchain and allowing double-spend exploits, overall reducing trust of honest users on the system and negatively affecting its perception and wide-spread use. We highlight that selfish mining remains as a major focus in blockchain research [Hou et al., 2021].

Timestamp based SM defense. The initial defense mechanism proposed by [Eyal and Sirer, 2018] was that honest miners should choose one of the chains at random when there are multiple chains of the same *length* (i.e., the number of blocks), while in the deployed Bitcoin implementation of April 2021, miners

choose the one that they received first. Regardless of the attacker's location or bandwidth, this defense prevented him from obtaining high rewards for computational powers below 25% hashing power of the whole system.

A later proposal by [Heilman, 2014] utilizes timestamps issued by an authority to improve this resistance up to the computational power of 33.3%. Although by the state-of-the-art selfish mining attack by [Sapirshtein et al., 2016], its security lowers to 30.1%, the proposal of [Heilman, 2014] still remains as the most resistant work against selfish mining. The defense idea is that honest miners would pick the chain with fresher timestamps in case of a tie in chain lengths. As this solution was contradicting to the decentralized philosophy of blockchain, Heilman also considered removal of the timestamp authority. However, we show that this solution without a timestamp authority (i.e., with forgeable timestamps) is susceptible to two attacks, one by setting the timestamp to the future and another by mining on the block previous to the last mined one [3].

Our approach. We would like to leverage the high security of [Heilman, 2014] against the state-of-the-art selfish mining attacks. However, requiring an authority for timestamps prevents this solution from a general use. Therefore, we would like to remove it, and let the miners put the current time into their mined blocks themselves. Yet, we detect two natural issues arising with this scheme, i.e., it becomes possible for a defective miner to set the timestamp of the block being mined to a future point (*Oracle mining*), and it gets easier to mine an alternate chain by starting from an old block (*Bold mining*). We show that if the attacker chooses the attack parameters fine-tuned, unfortunately, choosing the fresher chain works in favor of the defectively mined chain in both cases. Although these attacks are simple and natural, their analyses and optimization for the attacker led us to develop a new mathematical model, which can be standardized for using in selfish mining attack analyses beyond our work. To solve these issues, we

[3]Without a timestamp authority, [Heilman, 2014] also considered a possible attack by setting the timestamp to the future, and named it as "slothful mining". However, this attack has neither been formally defined nor analyzed so far.

then propose `Fortis`, a mining algorithm based on forgeable timestamps that is optimized to be *not* vulnerable to these attacks or the state-of-the-art optimal selfish mining attack [Sapirshtein et al., 2016] against a computational power ratio up to 27.0%. Thereby, we achieve the highest security against a single rational miner capable of performing all the selfish attacks known to date. More concretely, we list the contributions of this work as follows:

1. We define two natural mining attacks as mentioned above, namely Bold mining and Oracle mining, exploiting timestamp based solution of [Heilman, 2014].

2. We propose generic formulas that can be utilized for revenue calculations in selfish mining type of attacks. Our formulas makes complicated analysis of attacks easier and more dependable by providing a systematical methodology. We use this to show that there exists optimized choices of parameters in both Bold and Oracle mining, that yield a miner more than his fair share, even if his resources are very low.

3. We propose our selfish mining mitigation algorithm `Fortis` that defends against previous selfish mining attacks and our proposed ones, as long as the attacker's computational power ratio is less than 27.0% of the whole system, providing the best-known solution to date. Our solution is decentralized and the only assumption that we make over Bitcoin is availability of a global synchronous clock (which is a common assumption in other blockchain proposals [Kiayias et al., 2017, Badertscher et al., 2018b]).

4. We provide the simulation results for Oracle and Bold mining attacks against the solution of [Heilman, 2014] and against our `Fortis` algorithm. Our simulation is based on implementation of these attacks and defenses on top of the core blockchain simulation Blocksim by [Alharby and van Moorsel, 2020]. It confirms our theoretical results by showing that these

attacks are indeed effective against the solution of [Heilman, 2014] and ineffective against ours. We also simulate small (but realistic) clock drifts and show that the attacks and our defense are still viable.

5.2 Related Work

Selfish mining attacks. A major strike for blockchain security was the invention of selfish mining (SM) attack by [Eyal and Sirer, 2018]. Briefly, the attacker keeps his mined blocks as a private chain for further mining on them, and later releases his private chain, when the public chain approaches it in terms of length. This way, the attacker can eliminate the honest blocks from the main chain. The proposal shows that even if the honest majority assumption holds, a miner can obtain more revenue than honest mining via this mining strategy, since the difficulty adjusts automatically through time. Eventually what matters is the ratio of blocks being found by the attacker (the relative revenue) in the main chain, although block rewards and his resource investment remain the same.

We call any type of attack involving a miner that keeps his mined blocks private to obtain high relative revenue as an SM-type attack. Some recent studies including [Sapirshtein et al., 2016, Gervais et al., 2016, Nayak et al., 2016, Wang et al., 2019a] of selfish mining optimizes it further by allowing the attacker to mine on the private chain even when the public chain is longer. Other focuses of attention related to the SM attack are combining it with different attacks (e.g., with Eclipse attack by [Heilman et al., 2015, Nayak et al., 2016] and block withholding attack by [Bag et al., 2017, Elliott, 2019, Dong et al., 2019]), conducting the attack on different currencies (e.g., on Ethereum [Ritz and Zugenmaier, 2018]), and exploring the game theoretical implications of the attack by [Kiayias et al., 2016b, Liu et al., 2018, Marmolejo-Cossío et al., 2019, Leelavimolsilp et al., 2019, Koutsoupias et al., 2019]. We note that the state-of-the-art optimized selfish mining attack is "optimal selfish mining" (OSM) algorithm of [Sapirshtein et al., 2016]. We do not go over the details of the algorithm or its relative revenue calculations here, and refer the reader to the original paper.

Selfish mining is harmful to the blockchain system, not only since it is stealing from the fair shares of honest miners, but also since it results in inconsistent views of blockchain among the users. A recent study by [Davidson and Diamond, 2020] showed that other currencies than Bitcoin are far more susceptible to selfish mining, and to the best of our knowledge, differentiating this attack from a network partition is not fully possible yet. So far, selfish mining seems to be an unavoidable burden for the blockchain community for being robust against network partitions [Zhang and Preneel, 2017].

Existing selfish mining defenses. Here we briefly review the existing selfish mining defenses (honest mining algorithms) in the literature.

Uniform tie-breaking. The initial SM defense proposed by [Eyal and Sirer, 2018] was that an honest miner picks a branch uniformly at random among the longest chains in case of ties. We call this defense as Uniform Tie-breaking (UT). Against the SM attack, this defense achieves security up to hashing power 25.0%. However, the later optimized attack proposal of OSM of [Sapirshtein et al., 2016] reduced this to 23.2%.

Freshness preferred. Heilman proposes use of timestamps for tie-braking to reduce the relative revenue of a selfish miner [Heilman, 2014]. In case of a tie, the miners choose the branch that is fresher (i.e., the branch whose timestamps (TS) are closer to the current time τ), hence the scheme is called "Freshness Preferred" (FP). As the primary proposal, the author recommended support of a TS authority that generates unforgeable timestamps that will be embedded in the blocks to eliminate any forgery risks. If the timestamps are generated in an infinitesimal fashion, the scheme improve security against SM to the hashing power 33.3% and against OSM to the hashing power 30.1% (computed by using the implementation of [Optimal selfish mining strategies implementation, 2017]). However, this scheme suffers from centralization and as we show removing this authority turns out to be tricky.

Publish or Perish (PP) by [Zhang and Preneel, 2017]. provides incentive compatibility against OSM up to a hashing power 25.0%. As we do not utilize this

algorithm and its related defense ideas in our work, we skip the protocol details and refer the reader to [Zhang and Preneel, 2017]. For comparison, Fortis provides security to all known attacks (including ones analyzed in this work) up to a hashing power 27.0%. Also, [Zhang and Preneel, 2017] defines a notion "fail-safe parameter" as the minimum length difference required for enforced adoption of the longest branch, and suggests 3 as the optimum fail-safe parameter for PP. The paper shows that in this case, there exists an attack that results in a expected 18.5 blocks to pass for a consensus in case of a tie. We stress that this attack is not possible in our case and even in case of a tie, the consensus among the honest miners occurs immediately. Although in a setting with complete absence of a global clock publish and perish remains as the most secure solution known to date, our experiments in Section 5.8 shows that even with a low synchrony Fortis defends against attackers with higher hashing powers than 25.0%.

To clarify the significance of our 2.0% improvement against PP, in Bitcoin, it corresponds to roughly 300 Million US Dollars additional investment in hardware for an attacker to benefit from the attack, as of October 2021.[4]

GHOST of [Sompolinsky and Zohar, 2015], Bobtail of [Bissias and Levine, 2020], and the other works [Bahack, 2013, Shultz, 2015, Solat and Potop-Butucaru, 2017], unfortunately, fail to satisfy backward incompatibility [Zhang and Preneel, 2017] (i.e., old blocks cannot be verified with them). Also, there exist some works including [Saad et al., 2019, Chicarino et al., 2020] for detecting the behavior of the selfish miner and eliminate his blocks from the main chain. However, this detection has not been reliably achieved yet, as there seems to be no clear way for differentiating it from network partitioning.

[4]The cost is calculated from the fact that in October 2021, the total hash rate is 120 million tera hash per second (TH/s) and a mining hardware with 100 TH/s can be found for the price of 13,000 dollars.

5.3 Preliminaries

Notation for mining algorithms. In a miner's view, there exists two separate chains, the chain `PubCh` known by all miners and the chain `MyCh` chosen by the miner for mining on. `PubCh` gets updated automatically and might have forks of equal length, in which case a miner needs to choose which block to mine on.

- `Append`(`Chain`,b) denotes that "append the block b to the head of the `Chain` and other blocks between b and the head of the `Chain`".

- `Publish`(b) denotes that "publish the block b".

- `Publish`(`MyCh`,z/head) denotes that "publish all the blocks until and including either z-th block of `MyCh` indexed from start of the fork from `PubCh` (the first forked block is indexed as 1) or the head of `PubCh`".

- `Trun`(`Chain`,z) returns the chain obtained by truncating `Chain` by z blocks backwards from the head. `Last` denotes the index of the last block found by the miner in `PubCh` backwards from the head.

- $a \leftarrow b$ denotes that a is set as whatever b is at that moment.

- A $\twoheadrightarrow b$ denotes that "A finds the next block b". A denotes "The attacker's pool" and O denotes "Any pool other than attacker's one".

- `Mine`(`Chain`,`Fork`,z,TS) denotes that "mine on starting from z-th block backward indexed from the head (the index of the head is 1) of the `Fork` of the `Chain`, and set the timestamp of the currently mined block as TS". We highlight that an attacker always mines on the fork that includes more blocks found by him, therefore we omit the `Fork` input in attack algorithms. Also, if the fork resolving policy does not depend on timestamps, the input TS can be omitted.

Abbreviation/Notation	Meaning
TS	Timestamp
UT	Uniform tie-braking
FP	Freshness preferred
Authority FP	Freshness preferred scheme with unforgeable timestamps
Decentralized FP	Freshness preferred scheme with forgeable timestamps
PP	Publish or perish scheme of [Zhang and Preneel, 2017]
SM	Selfish mining attack [Eyal and Sirer, 2018]
SM-type attacks	Attacks to receive high revenue by private mining
OSM	Optimized selfish mining attack [Sapirshtein et al., 2016]
OM	Oracle mining (our attack)
BM	Bold mining (our attack)
α	Hashing power
γ	Network power
τ	The current timestamp (depending on the scheme based on the global clock or timestamp authority)
ϵ	Increment of the timestamp
t_m	The difference of the timestamp from current time for maximizing the revenue of Oracle miner
t_C	The expected time interval between block found by any miner (meaningful in Oracle mining)
S_{pub}	The state of the SM-type attacker when mining on a publicly known block
S_{priv}	The state of the SM-type attacker when mining on a block private to himself
R	Revenue of a miner

Table 5.1: Table of abbreviations and notations used throughout Chapter 5

Table of abbreviations. We provide Table 5.1 for the frequently used abbreviations and notations throughout the chapter.

Uniform tie-breaking algorithm. The honest miner's algorithm in uniform tie-breaking (UT) [Eyal and Sirer, 2018] is given in Algorithm 12.

Algorithm 11 Uniform Tie-breaking Mining Algorithm of [Eyal and Sirer, 2018].

```
while true do
    if PubCh has 1 branch then
        Mine(PubCh,PubCh,1)
    else if PubCh has n branches then
        Mine(PubCh,i-th branch,1) with probability 1/n
    end if
    Propagate PubCh
end while
```

Freshness preferred algorithm. The honest miner's algorithm of [Heilman, 2014] is given in Algorithm 12. In case of a tie, the miners choose the branch that is fresher (i.e., the branch whose timestamps (TS) are closer to the current time τ), hence the scheme is called "Freshness Preferred" (FP). As the primary proposal, the author recommended support of a TS authority that generates unforgeable

timestamps that will be embedded in the blocks to eliminate any forgery risks. However, considering that this would be an obstacle for decentralization, the author also argued that the authority may not be necessary in the presence of a global synchronous clock, as the timestamps cannot be changed once the block is mined. In this case each miner can just embed the current time into the block being mined. We call the former and the latter schemes as Freshness Preferred with unforgeable timestamps (Authority FP) and Freshness Preferred with forgeable timestamps (Decentralized FP), respectively.

Algorithm 12 Freshness Preferred Mining Algorithm of [Heilman, 2014].

```
while true do
    if PubCh has 1 branch then
        Mine(PubCh,PubCh,1,τ)
    else if PubCh has n branches then
        Mine(PubCh,freshest branch,1,τ)
    end if
    Propagate PubCh
end while
```

Standard model of revenue analysis.[5] The model that we utilize throughout the chapter mainly includes two players, Adam (the attacker) and Helen (the collection of the rest of the miners, assumed to be honest). Adam has a relative hashing power $\alpha < 0.5$ (i.e., the ratio of the number of hash operations by him over that all hash operations in a given time interval) and a relative network power γ (i.e., the expected ratio of the honest miners that will work on the selfish miner's chain in case of a tie). We note that in Bitcoin, γ depends on the bandwidth and location of the attacker's competitor blocks origin, as miners choose the branch that they receive first in case of a tie. We omit propagation delays

[5]This model is also the one provided by the original SM paper by [Eyal and Sirer, 2018]. Although analyses in different settings (e.g., without fixed block rewards [Carlsten et al., 2016, Tsabary and Eyal, 2018], in asynchronous setting as in [Gervais et al., 2016, Pass et al., 2017], in the presence of multiple attackers as in [Liu et al., 2018, Marmolejo-Cossío et al., 2019, Leelavimolsilp et al., 2019]) also exist, the previous attacks and defenses are mostly analyzed in this model.

and network partitions, therefore honest miners know whatever is public immediately. Honest miners may mine on different blocks due to forks. We assume a fully anonymized network, where detecting a malicious miner personally is impossible. We assume that computation times other than hashes and costs due to mining are 0. The finder of any block in the main chain is rewarded 1, unless it is stated otherwise. We omit the transaction fees from the reward for simplicity. Difficulty adjustment is quick enough that the revenue of attacker is equivalent to his relative revenue (i.e., the ratio of the blocks found by him over those found by all miners). We say that Adam benefits from an attack if his relative revenue from the attack is more than his benefit from honest mining. Otherwise, we say that he does not benefit from an attack.

It would be interesting to see the analyses of our attack and defense proposals in presence of multiple attackers. It has been shown that selfish miners perform better, when there are other selfish miners in the system [Liu et al., 2018, Marmolejo-Cossío et al., 2019, Leelavimolsilp et al., 2019]. Yet, these analyses become very complicated, even when there exist two attackers. Thus, in this work we limit ourselves to one attacker. Also, the analysis of our defence in more realistic setting "imperfect network" as in [Yang et al., 2020, Wang et al., 2019b] is left as an interesting future work. We note that other mentioned defenses (except for Uniform Tie-breaking) are analyzed only in presence of one attacker with to a large extent our mentioned assumptions as well. The only additional assumption that we make over these works is presence of a global clock, but later in Section 5.8, we provide simulation results to show that even with a low synchronicity we provide high security.

5.4 Oracle and Bold Mining Attacks

In this section we present two SM-type attacks, the Oracle and Bold Mining attacks, on Decentralized FP. These attacks are direct exploits of enforcing the honest miners pick the freshest *looking* chain in case of a tie. We highlight that these attacks are not covered in the optimal selfish mining of [Sapirshtein et al., 2016],

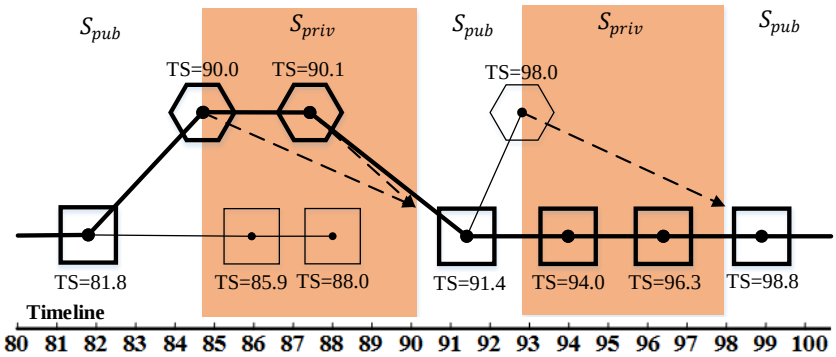


Figure 5.1: An example flow of Oracle mining attack. The squares are honest miners' blocks, while the hexagons are those of the attacker. Straight lines shows parent-child relation between two blocks. The thick lined blocks are the ones that remains in the main chain and gets rewarded, while the thin lined ones are orphan blocks. The dashed arrows point to the time when the attacker publishes the block. S_{pub} and S_{priv} states are shown by the areas colored as white and coral, respectively.

as that work do not consider the use of self-generated timestamps for selfish mining mitigation.

We represent all SM-type attack algorithms in this chapter as two distinct states of the attacker: one where he is mining on a block on the public chain (we call this S_{pub}) and one where he is mining on his private chain (we call this S_{priv}). At a given time, the attacker will be at either S_{pub} or S_{priv} state. This separation is sound based on the generic idea of selfish mining attack strategies that the attacker wants to eliminate some of the already-found blocks of the honest majority from the main chain. Often, the attacker continues with the honest mining in S_{pub}, but sometimes when the attacking conditions occur, he deviates from the honest mining strategy in order to go to S_{priv}. The reason for separation of these two states is simplicity of revenue calculation, and will be clearer in Section 5.5).

Oracle mining algorithm. We now describe a rational algorithm based on setting the timestamp of the currently mined block to the future. At S_{pub} the

attacker sets the timestamp $TS = \tau + t_m$ of the block he is mining on, where τ is the current time (based on the global clock) and $t_m > 0$. We will later show how to set t_m for maximizing the revenue depending on α. If he succeeds, he will have a block that cannot be published immediately, but can be kept private for mining on until about the time $\tau + t_m$ (i.e., by switching to S_{priv}). He will increment the timestamps of the next blocks he finds at S_{priv} with the unit increment ϵ. If the honest miners outperform him by mining two blocks more than him at S_{priv}, the attacker's found blocks will be lost. Otherwise, he will kick all the blocks of honest miners found within S_{priv} off the main chain. Algorithm 13 provides full description of the OM attack, where ϵ and t_m denote the incremental unit of timestamps and the value t that maximizes the attacker's revenue, respectively. Also, an example flow of OM is shown in Figure 5.1, including example S_{pub} and S_{priv} states.

Bold mining algorithm. The attacker's strategy is essentially to mine a sibling to the k-th past block from the current head at S_{pub} (if he does not own any block between the current head and the k-th past block, both inclusive), and then to keep it private (going to S_{priv}), and next to try to catch up with honest miners by mining on his found block. If the honest chain surpasses his chain (with $k+1$ blocks), he accepts defeat and goes back to S_{pub}. If his chain succeeds by having an equal number of blocks to the public chain, he publishes his chain and goes back to S_{pub} as the winner. We especially require the honest chain to be $k+1$ blocks ahead for the attacker's failure, since if the honest chain is k blocks ahead, the attacker is better off by following his private chain, since his private chain has more blocks belonging to him than the other one. Upon going to S_{pub} with success, to be flexible with the number k, we allow the attacker to choose a value K, such that K is the trail bound of the honest miners' blocks for k. Algorithm 14 provides the Bold mining (BM) algorithm. Also, an example flow of the BM attack is shown in Figure 5.2, including example S_{pub} and S_{priv} states.

As a side note, the bold mining attack is applicable to even Authority FP. This is because unlike oracle mining, the attacker mines the alternative chain to the

Algorithm 13 Oracle Mining Attack Algorithm.

```
procedure S_pub
    Set Δ ← 0, flag ← 0
    while flag = 0 do
        Find t_m for α
        MyCh ← PubCh, Mine(MyCh,τ + t_m)
        if A ↠ b then
            Append(MyCh,b)
            Set t ← τ + t_m, Δ ← 1, flag ← 1
        end if
    end while
    Go to procedure S_priv
end procedure

procedure S_priv
    while τ < t + (Δ − 1) · ϵ do
        Mine(MyCh,1,t + Δ · ϵ)
        if A ↠ b and Δ > 0 then
            Append(MyCh,b), Set Δ ← Δ + 1
        end if
    end while
    Publish(MyCh,head), Go to procedure S_pub
end procedure
```

Algorithm 14 Bold Mining Attack Algorithm.

```
procedure S_pub
    Set flag ← 0
    while flag = 0 do
        Set Δ ← min(Last, K) − 1
        MyCh ← Trun(PubCh, Δ), Mine(MyCh, 1, τ)
        if A ↠ b and Δ = 0 then
            Publish(MyCh,head)
        else if A ↠ b and Δ > 0 then
            Append(MyCh,b)
            Set k ← Δ, Δ ← Δ − 1, flag ← 1
        else if O ↠ b and Δ < K then
            Set Δ ← Δ + 1
        end if
    end while
    Go to procedure S_priv
end procedure

procedure S_priv
    while 0 < Δ < k + 1 do
        Mine(MyCh,1,τ)
        if A ↠ b and Δ > 1 then
            Append(MyCh,b), Set Δ ← Δ − 1
        else if A ↠ b then
            Set Δ ← Δ + 1
        end if
    end while
    Publish(MyCh,head), Go to procedure S_pub
end procedure
```

public one by using the current timestamp. Thus any results of this attack are achievable even in the presence of a timestamp authority. Although our concern is the decentralized setting, this attack and its analysis and optimization given in Section 5.6.2 applies to the settings with timestamp authorities.

Optimum parameters for the attacks. The values t_m and K in Oracle and Bold Mining are the parameters that the attacker should choose, respectively. As a rational party, he would naturally be interested in setting them in a way that it would maximize his share in the system. However, this requires a complete analysis of both attacks' revenues. In particular, the analysis of oracle mining is difficult, and although the attack idea exists in a basic form in [Heilman, 2014], it has never been analyzed ever since. Thus we will return setting the optimum of

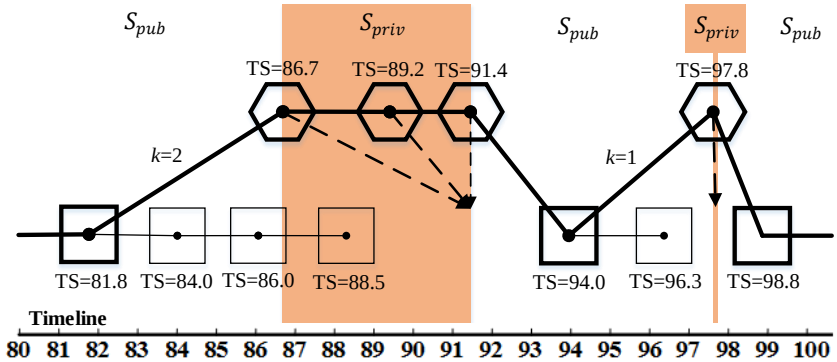


Figure 5.2: An example flow of Bold mining attack. The squares are honest miners' blocks, while the hexagons are those of the attacker. Straight lines shows parent-child relation between two blocks. The thick lined blocks are the ones that remains in the main chain and gets rewarded, while the thin lined ones are orphan blocks. The dashed arrows point to the time when the attacker publishes the block. S_{pub} and S_{priv} states are shown by the areas colored as white and coral, respectively. Note that in case $k = 1$, the attacker immediately releases the block in S_{priv}.

these parameters, after their analyses in Section 5.6.

5.5 Generic Formulas For SM Attacks

The formulation. We now derive a formulation that we will use for analyzing the attacks given in Section 5.4. The separation of the states S_{pub} and S_{priv} that we provided at the beginning of that section will help us with this aim. We start by the main formulas for the expected revenue of the attacker which can be calculated as:

$$R = \frac{\alpha - f \cdot \ell_A}{1 - f \cdot (\ell_A + \ell_H)} \tag{5.1}$$

where ℓ_A, ℓ_H, and f denote the expected block loss of the attacker in S_{priv}, the expected block loss of the honest parties in S_{priv}, and the expected frequency of returning to S_{priv}, respectively. Since being at states S_{priv} and S_{pub} keep alternat-

ing, f can be calculated as

$$f = \frac{1}{ES_{priv} + ES_{pub}} \tag{5.2}$$

where ES_x is the expected number of all blocks mined by both parties secretly or publicly while being at state S_x. The intuition for the equation is that if the attacker only executes honest mining all the time during an expected period $ES_{priv} + ES_{pub}$, the expected number of blocks that he finds would be $\alpha(ES_{priv} + ES_{pub})$ (where α is the attacker's hashing power) and those that are found by all miners would be $ES_{priv} + ES_{pub}$. We obtain Eq. 5.1 by subtracting the corresponding expected losses from both, then dividing them by the period, and then calculating the ratio of the former to the latter.

Whether the attacking strategy is expected to benefit the attacker depends on the ratio of $\frac{\ell_A}{\ell_H}$. More concretely, iff the attacking strategy benefits the attacker, then we have

$$\frac{\ell_A}{\ell_A + \ell_H} < \alpha \quad \text{or equivalently} \quad \frac{\ell_A}{\ell_H} < \frac{\alpha}{1-\alpha} \tag{5.3}$$

We name the ratio $\frac{\ell_A}{\ell_H}$ as profitability ratio B. We emphasize that sole knowledge of ℓ_A and ℓ_H is not enough to determine the attacker's revenue, but rather a tool for his decision-making in choosing between applying an SM-type mining strategy and honest mining. We note that setting $R > \alpha$ in Eq. 5.1 leads to Eq. 5.3.

Example application. To show that our formulas indeed work, we apply them to the basic selfish mining algorithm of [Eyal and Sirer, 2018]. Algorithm 15 provides the same SM algorithm given by [Eyal and Sirer, 2018], but it shows S_{priv} and S_{pub} states as distinct procedures.

In S_{pub}, the adversary tries to mine a block. Whenever he mines it, instead of publishing, he keeps it secret and goes to S_{priv}. His block loss can only occur if an honest miner finds a block first in S_{priv}, and then $\gamma(1-\alpha)$ fraction of honest miners that work on Helen's block finds the next block. In this case, Adam loses 1 block. As the probability that this case occurs is $(1-\alpha)^2(1-\gamma)$, we obtain

$$\ell_{A,SM} = (1-\alpha)^2(1-\gamma)$$

Algorithm 15 Selfish Mining Attack Algorithm of [Eyal and Sirer, 2018].

```
procedure S_pub
    Set Δ ← 0, flag ← 0
    while flag = 0 do
        if |MyCh| = |PubCh| and MyCh ≠ PubCh then
            Mine(MyCh,1)
            if A ↠ b then
                Append(MyCh,b), Publish(MyCh,head)
            end if
        else
            MyCh ← PubCh, Mine(MyCh,1)
            if A ↠ b then
                Append(MyCh,b), Set Δ ← 1, flag ← 1
            end if
        end if
    end while
    Go to procedure S_priv
end procedure

procedure S_priv
    Set flag ← 0
    while flag = 0 do
        Mine(MyCh,1)
        if A ↠ b and Δ > 0 then
            Append(MyCh,b), Set Δ ← Δ + 1
        else if O ↠ b and Δ ≤ 2 then
            Publish(MyCh,head), flag ← 1
        else if O ↠ b and Δ > 2 then
            Publish(MyCh,1), Set Δ ← Δ − 1
        end if
    end while
    Go to procedure S_pub
end procedure
```

Helen may lose blocks in S_{priv} mainly in two different ways: (1) Helen finds the first block, Adam publishes the head of his private chain, Adam's block wins the block race with probability $(1-\alpha)\big(\gamma(1-\alpha)+\alpha\big)$, Helen loses 1 block. (2) Adam finds the first block with probability α, eventually Helen catches up with Adam, Adam publishes his chain and Helen loses all the blocks she has found within S_{priv}. For the second outcome, we consider this as a "monkey at the cliff" problem [Alm, 2002], starting when Adam's chain is 2 blocks ahead of Helen's chain, going away from the cliff with probability α and towards it with probability $1-\alpha$. Therefore, the expected number of steps for this walk to the end is $\frac{1}{(1-\alpha)-\alpha} = \frac{1}{1-2\alpha}$ [Alm, 2002]. Since the expected steps towards the cliff should be 1 more than those away from it for the state to end, we calculate Helen's expected block loss in this case as $\frac{1}{2}\big(\frac{1}{1-2\alpha}+1\big) = \frac{1-\alpha}{1-2\alpha}$. At the end, we obtain

$$\ell_{H,SM} = (1-\alpha)\big(\gamma(1-\alpha)+\alpha\big) + \alpha \cdot \frac{1-\alpha}{1-2\alpha}$$

Regarding f_{SM}, we need to obtain ES_{priv} and ES_{pub}. Upon going to state S_{priv}, the probability that Helen finds the next block is $(1-\alpha)$, ending the state with 1 block. Upon going to state S_{priv}, the probability that Adam finds the next block is α, ending the state in expected $1+\frac{1}{1-2\alpha}$ steps. We calculate $ES_{priv} = (1-\alpha) + \alpha\left(1+\frac{1}{1-2\alpha}\right) = 1+\frac{\alpha}{1-2\alpha}$. At the beginning of S_{pub}, there may be a block race where either Adam's branch or Helen's one (the forks differ by only 1 block) will be the winner if Helen found the first block in the last S_{priv}. Thus, this case occurs with probability $1-\alpha$. After this, S_{pub} finishes when the attacker mines the next block requiring the expected number of $\frac{1}{\alpha}$ blocks since it is a geometric random variable. Hence, we calculate $ES_{pub} = 1-\alpha+\frac{1}{\alpha}$. From Eq. 5.2, we obtain that

$$f_{SM} = \frac{1}{2+\frac{\alpha}{1-2\alpha}-\alpha+\frac{1}{\alpha}} = \frac{(1-2\alpha)(\alpha)}{1-4\alpha^2+2\alpha^3}$$

If we plug $\ell_{A,SM}$, $\ell_{H,SM}$, and f_{SM} into Eq. 5.1, we obtain an equation equivalent to the revenue equation of [Eyal and Sirer, 2018] as

$$R_{SM} = \frac{\alpha(1-\alpha)^2(4\alpha+\gamma(1-2\alpha))-\alpha^3}{1-\alpha(1+(2-\alpha)\alpha)}, \tag{5.4}$$

which is also deducible from Figure 5.3, that shows the revenues of the attacker with respect to his hashing power for network powers $\gamma = 0$, $\gamma = 0.5$, and $\gamma = 1$ for both equations.

5.6 Analyses of Oracle and Bold Mining

We now show that there exists parameters for t_m in Oracle Mining and K in Bold Mining, such that the attacker is better off compared to honest mining regardless of his hashing power. The simulation results in Section 5.8 confirms our results obtained here.

5.6.1 Oracle Mining Attack Optimization

Analysis. For simplicity we let $\epsilon = 0$. We model the mining in a given time range as a Poisson random variable as in [Sapirshtein et al., 2016]. The success

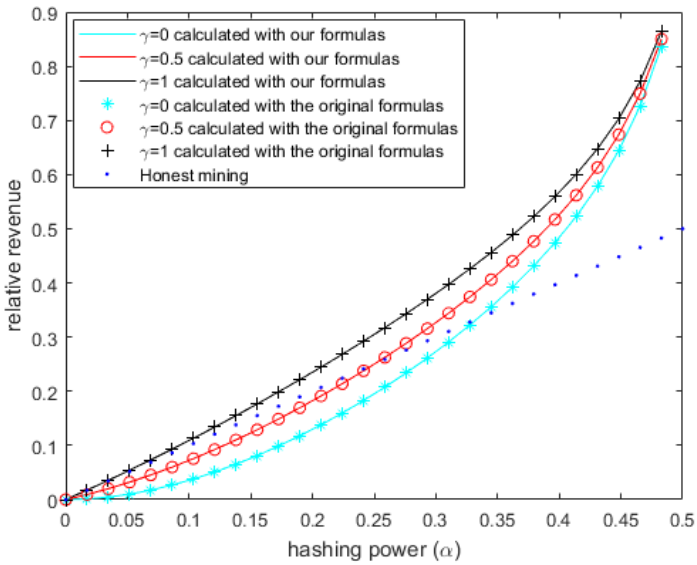


Figure 5.3: Revenue from selfish mining [Eyal and Sirer, 2018] with respect to hashing power α for network powers $\gamma = 0$, $\gamma = 0.5$, and $\gamma = 1$ calculated using our Eq. 5.1 and the original equation of [Eyal and Sirer, 2018] given at Eq. 5.4.

probability is given as $P[X = x] = \frac{e^{-\mu}\mu^x}{x!}$ where $X = \{0, 1, 2, 3 \ldots\}$ is a set of possible number of successes, e is the Euler's number, and μ is a success rate. Let T_c denote the current expected time interval between block found by any miner. This differs from the public expected inter-block time by including the blocks that do not end up in the main chain. The attacker can determine this value from his hashing power, or can estimate it from statistics as he knows his and honest miners' found blocks. We let $\mu_A = \frac{\alpha t}{T_c}$ and $\mu_H = \frac{(1-\alpha)t}{T_c}$ denote success rates of Adam and Helen, respectively.

We can state

$$\ell_{A,OM} = \sum_{h=2}^{\infty} \sum_{a=0}^{h-2} \big(P[A = a] \cdot P[H = h] \cdot (a + 1)\big)$$

$$\ell_{H,OM} = \sum_{a=0}^{\infty} \sum_{h=1}^{a+1} \big(P[A = a] \cdot P[H = h] \cdot h\big)$$

Here $P[A = a]$ and $P[H = h]$ denote probabilities that Adam finds a blocks and Helen finds h blocks within S_{priv}, respectively. Since these are Poisson random variables, $P[A = a] = \frac{e^{-\mu_A}\mu_A^a}{a!}$ and $P[H = h] = \frac{e^{-\mu_H}\mu_H^h}{h!}$. As the time spent in S_{priv}

is t_m, we calculate $ES_{priv} = \frac{t_m}{T_c}$. Also, $ES_{pub} = \frac{1}{\alpha}$, as S_{pub} finishes when Adam finds a block. Hence, from Eq. 5.2, we obtain

$$f_{OM} = \frac{1}{\frac{t_m}{T_c} + \frac{1}{\alpha}} = \frac{\alpha T_c}{\alpha t_m + T_c}$$

Plugging $\ell_{A,OM}$, $\ell_{H,OM}$, and f_{OM} into Eq. 5.1, it is straightforward to obtain Adam's revenue. By setting t_m properly, the attacker can maximize his revenue from this attack and thus beat the honest mining strategy for any α

How to set the optimum t_m. Intuitively, setting t as a large value does not make much sense for an attacker with $\alpha < 0.5$, since it gives the honest majority a larger time interval (and hence a higher chance) to beat him. From the analysis, we deduce that increasing the value of t increases both $\ell_{A,OM}$ and $\ell_{H,OM}$, and decreases f_{OM}. Figure 5.4 shows the revenue of the attacker with respect to α for optimum t_m/T_c. The attacker can keep setting the t_m by preparing a look up table for t_m/T_c and α from this figure or computing them by himself. We note that both α and T_c may vary over time, but the attacker can estimate them from the difficulty and the statistics of the publicly and privately mined blocks.

Impact of the attack. The most important implication of this attack is that for any α value, there exists a value $t_m > 0$ that provides more revenue than honest mining does. Even if the direct revenue gain from this attack does not seem as great as the previously known SM-type attacks, it is more likely to occur in practice in an Decentralized FP application, since it may be attractive for any rational miner, while the latter usually requires large α or γ values. Its effects are similar to selfish mining: It exhausts honest miners, which can reduce their numbers, leading to a more vulnerable blockchain system to other SM-type attacks. It also results in inconsistency as a ledger of transactions.

5.6.2 *Bold Mining Attack Optimization*

Analysis of profitability Ratio for all bounds K. We know from the previous analyses (including the one by Nakamoto [Nakamoto, 2008]) of PoW mining that it is not a good idea to mine on from long past blocks; therefore, there must be

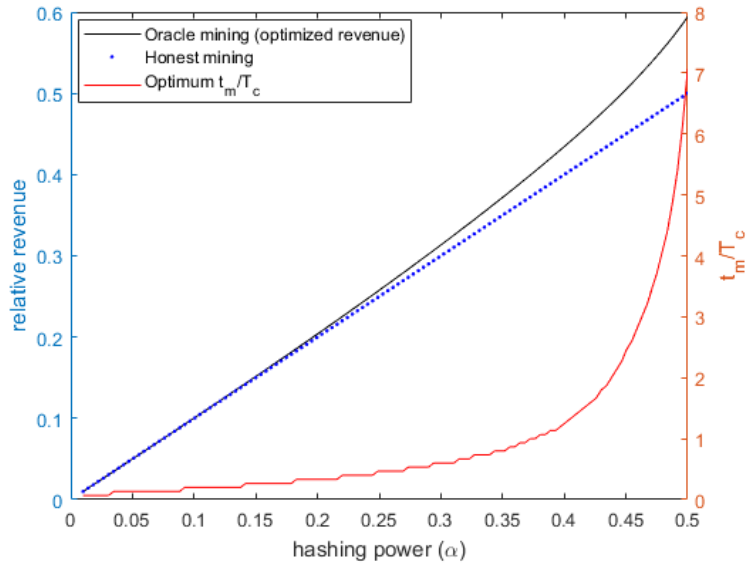


Figure 5.4: Revenue from Oracle mining on Decentralized FP w.r.t. the hashing power α when the timestamps set optimally. Also the optimum t_m/T_c values for all α of interest are provided.

a good bound K for the plausible values of k. In what follows, we show that for $\alpha < 0.432$ the only plausible value is $K = 1$ (and in the range $\alpha \in (0.432, 0.5)$, the only other plausible bound is $K = 2$), then calculate the revenue equation based on this choice.

For simplicity, we define $\ell_{A,BM,k}$ and $\ell_{H,BM,k}$ as expected losses of Adam and Helen, respectively, in case Adam goes to S_{priv} with k. The corresponding profitability ratio $B_{BM,k} = \frac{\ell_{A,k}}{\ell_{H,k}}$ still needs to be less than $\frac{\alpha}{1-\alpha}$. We state that

$$\ell_{A,BM,k} = P[\text{Helen surpasses } k+1 \text{ blocks}](1 + E_{HA})$$

$$\ell_{H,BM,k} = P[\text{Adam catches up}](1 + E_{AH})$$

where $E_{H\,\text{wins},A}$ and $E_{A\,\text{wins},H}$ denote the expected number of blocks found by Adam in S_{priv} given that Helen wins, and those by Helen given that Adam wins, respectively. The flow of S_{priv} can be modeled as the "Gambler's ruin" problem [Alm, 2002].Let E_H and E_A denote the expected number of blocks found by all parties in S_{priv} given that Helen wins (Adam loses) and those given that Adam

wins, respectively. $E_{A\text{ wins},H} = \frac{E_A-(k-1)}{2}$ and $E_{H\text{ wins},A} = \frac{E_H-2}{2}$. We utilize Corollary 2.12 of [Lorek and Markowski, 2018], and deduce that

$$E_A = \frac{r+1}{r-1} \cdot \left(N \cdot \frac{r^N+1}{r^N-1} - i \cdot \frac{r^i+1}{r^i-1}\right) \text{ if } k > 1,\ 0 \text{ for } k > 1$$

$$E_H = \frac{r+1}{r-1} \cdot \left(N \cdot \frac{r^N+1}{r^N-1} - (N-i) \cdot \frac{r^{N-i}+1}{r^{N-i}-1}\right) \text{ for } k > 1$$

where $r = \frac{\alpha}{1-\alpha}$ is the ratio of Adam's and Helen's probabilities of finding the next block, $i = 2$ is the starting point of the gambler's ruin, and $N = k+1$ is its ending point. From [Alm, 2002], we also deduce that

$$P[\text{Adam catches up}] = \frac{\left(\frac{1-\alpha}{\alpha}\right)^2 - 1}{\left(\frac{1-\alpha}{\alpha}\right)^{k+1} - 1} \text{ for } k > 1$$

$$P[\text{Helen surpasses } k+1 \text{ blocks}] = 1 - \frac{\left(\frac{1-\alpha}{\alpha}\right)^2 - 1}{\left(\frac{1-\alpha}{\alpha}\right)^{k+1} - 1} \text{ for } k > 1$$

By plugging the above formula, it is straightforward to calculate the profitability ratio $B_{BM,k} = \frac{\ell_{A,BM,k}}{\ell_{H,BM,k}}$. Using Eq. 5.3 we calculate both $\forall \alpha \in (0, 0.432)\ \forall k > 1$ $B_k \geq \frac{\alpha}{1-\alpha}$ and $\forall \alpha \in (0, 0.432)\ B_1 < \frac{\alpha}{1-\alpha}$. Therefore, by setting $K = 1$, and the attack in Algorithm 14 simplifies to the attack in Algorithm 16 as at S_{priv}, $\Delta = 0$.

Revenue analysis for optimized bound $K = 1$. Clearly $\ell_{A,BM} = 0$, as there are no blocks lost by Adam. Similarly, $\ell_{H,BM} = 1$, since Helen will lose 1 block whenever attack condition occurs. Also $ES_{priv} = 0$, as Adam immediately releases his found block. To calculate ES_{pub}, upon going out of S_{priv}, first, Helen needs to find a block, whose average length is $\frac{1}{1-\alpha}$. Then, Adam needs to find a block, whose average length is $\frac{1}{\alpha}$. From Eq. 5.2, we obtain $f_{BM} = \frac{1}{\frac{1}{1-\alpha}+\frac{1}{\alpha}} = \alpha(1-\alpha)$. Applying Eq. 5.1, we deduce

$$R_{BM} = \frac{\alpha - \alpha(1-\alpha).0}{1-\alpha(1-\alpha)(0+1)} = \frac{\alpha}{1-\alpha(1-\alpha)}$$

Figure 5.5 shows the expected revenue outcome of this attack for various α values, clearly demonstrating that it outperforms the honest mining strategy for any $\alpha < 0.5$.

Algorithm 16 Bold Mining Attack Algorithm as $K = 1$.

```
procedure S_pub
    flag ← 0
    while flag = 0 do
        Set Δ ← min(Last, 1) − 1
        MyCh ← Trun(PubCh,Δ), Mine(MyCh,1,τ)
        if A ↠ b and Δ = 0 then
            Publish(MyCh,head)
        else if A ↠ b and Δ = 1 then
            Append(MyCh,b)
            Set Δ ← Δ − 1, flag ← 1
        else if O ↠ b and Δ = 0 then
            Set Δ ← Δ + 1
        end if
    end while
    Go to procedure S_priv
end procedure

procedure S_priv
    Publish(MyCh,head), Go to procedure S_pub
end procedure
```

We note that [Sapirshtein et al., 2016] has previously stated that if freshness preferred is applied, mining on one block behind the head is profitable for any attacker. However, this deduction seems merely intuitive, as they do not provide any analysis of the attack. Here, our analysis will help us in our defense proposal.

Impacts of the attack. The impact of this attack is similar to that of Oracle mining, but with increased severity. For any α value, the attacker's revenue surpass the one receivable from honest mining, therefore it is more likely to occur in practice in an FP application than previously known SM-type attacks. It exhausts honest miners, which can reduce their numbers, leading to a more vulnerable blockchain system to other SM-type attacks. It also results in inconsistency as a ledger of transactions.

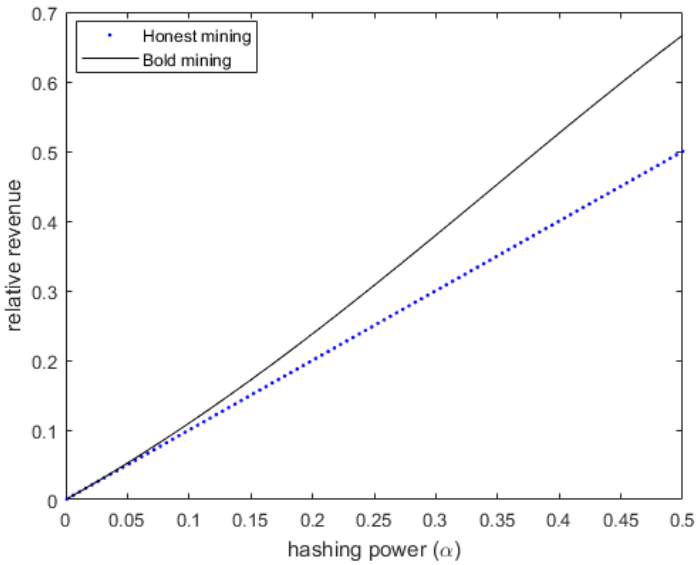


Figure 5.5: Revenue from Bold mining on Decentralized (or even Authority) FP vs. revenue from honest mining with respect to α.

5.7 Our SM Mitigation Algorithm

In the light of the analyses in Section 5.6, we now propose an algorithm `Fortis` with a tie-breaking strategy such that if applied, an attacker with $\alpha < 27.0\%$ can benefit from none of the SM-type attacks to known date including Oracle mining (OM), Bold mining (BM), and optimal selfish mining (OSM) [Sapirshtein et al., 2016]. This proposal is a relaxation of Decentralized FP, biasing some forks with respect to timestamps unlike Uniform Tie-breaking [Eyal and Sirer, 2018]. For the sake of proper operation of `Fortis`, we assume that a global synchronous clock functionality is available to every party. Yet, in Section 5.8, we also provide results that show even without high synchrony our proposal enjoys the highest security. Also, we stress that global clocks have already been used in proof-of-stake based consensus algorithms [Kiayias et al., 2017, Badertscher et al., 2018b].

Intuition. Observe that both OM and BM are applicable to FP, since in case of ties the freshest block is the one that always wins. However, neither of them is possible against uniform tie-breaking (UT) where $\gamma = 0.5$ (yet it prevents OSM only for adversaries with hashing power less than 0.232): Regarding OM, the

most likely outcome of S_{priv} is that only Helen would find one block within the time t. Thus, if $\gamma = 0.5$, Adam will risk his first found block in cases Helen also finds a block. Regarding BM, clearly, Adam is better off with honest mining, since the blocks he finds in BM will be counted only roughly half of the time, whereas Helen's block loss will be less in terms of ratio.

Let ψ denote the bias for the probability that an honest miner chooses the fresher chain in case of a tie. FP sets $\psi = 1$ for defending against SM. We can set this number to any value $0.5 \leq \psi < 1$, and still be better against OSM than UT that set it to $\psi = 0.5$. Decreasing ψ gives more chance to the blocks with lower timestamp, and we start to be susceptible to BM and OM. Therefore, we need an optimized value of ψ for a certain lowest bound of α that an attacker would benefit from any of OM, BM, and OSM. Let us reconsider the OM attack with the introduction of $\psi < 1$. Also, we need to consider that now the honest miners will be divided upon going back to S_{pub} with a tie. In this case the following occurs: Adam releases his blocks, and a block race takes place for the next block (Adam sets the timestamp of the block he is mining on as the current time). Then, the probabilities that Adam and Helen lose the mined blocks are $(1-\psi)(1-\alpha)$ and $\alpha + \psi(1-\alpha)$, respectively. Whoever finds the next block, Adam switches back to the honest chain, and continues with the attack algorithm the same way as in standard OM on FP. We state the expected block loss $\ell^{\psi}_{A,OM}$ of the attacker in S_{priv}, the expected block loss $\ell^{\psi}_{H,OM}$ of the honest parties in S_{priv}, and the expected frequency f^{ψ}_{OM} of returning S_{priv} in this attack are as follows:

$$\ell^{\psi}_{A,OM} = \sum_{h=2}^{\infty} \sum_{a=0}^{h-2} \big(P[A=a] \cdot P[H=h] \cdot (a+1)\big) + (1-\psi)(1-\alpha) \sum_{h=1}^{\infty} \big(P[A=h-1] \cdot P[H=h] \cdot h\big)$$

$$\ell^{\psi}_{H,OM} = \sum_{a=0}^{\infty} \sum_{h=1}^{a} \big(P[A=a] \cdot P[H=h] \cdot h\big) + \big(\alpha + \psi(1-\alpha)\big) \sum_{h=1}^{\infty} \big(P[A=h-1] \cdot P[H=h] \cdot h\big)$$

Algorithm 17 Our `Fortis` Honest Mining Algorithm

```
while true do
    if PubCh has 1 branch then
        Mine(PubCh,1,τ)
    else if PubCh has 2 branches then
        Mine(PubCh,fresher branch,1,τ) with probability Y
    else if PubCh has n branches s.t. n > 2 then
        Mine(PubCh,i-th branch,1,τ) with probability 1/n
    end if
    Propagate PubCh
end while
```

Moreover, as ES_{pub} of OM is elongated with 1 block due to the above-mentioned procedural change we obtain from Eq. 5.2:

$$f^{\psi}_{OM} = \frac{1}{\frac{t_m}{T_c} + 1 + \frac{1}{\alpha}} = \frac{\alpha T_c}{\alpha t_m + T_c + \alpha T_c}$$

Also, when BM is applied on `Fortis` and a tie occurs upon going back to S_{pub}, the above-mentioned block race and additional procedure takes place. We calculate the expected block loss $\ell^{\psi}_{A,BM}$ of the attacker in S_{priv}, the expected block loss $\ell^{\psi}_{H,BM}$ of the honest parties in S_{priv}, and the expected frequency f^{ψ}_{BM} of returning S_{priv} in this attack as follows:

$$\ell^{\psi}_{A,BM} = (1-\psi)(1-\alpha), \quad \ell^{\psi}_{H,BM} = \alpha + \psi(1-\alpha),$$

$$f^{\psi}_{BM} = \frac{\alpha(1-\alpha)}{1+\alpha(1-\alpha)}$$

We end up with an optimization problem regarding the revenues from the attacks OM, BM, and OSM, which can be stated as follows:

$$\begin{aligned} \textit{Maximize } \alpha \textit{ subject to}: \; & R^{\psi}_{OM}, R^{\psi}_{BM}, R^{\psi}_{OSM} < \alpha, \\ & \alpha \in (0, 0.5), \; \psi \in (0.5, 1) \end{aligned}$$

For R^{ψ}_{OM} and R^{ψ}_{BM} we apply Eq. 5.1 with the above values, and for R^{ψ}_{OSM} we utilize the implementation given in [Optimal selfish mining strategies implementation, 2017]. We obtain the solution for this problem is $\alpha \approx 0.270$ and occurs at

$\psi \approx 0.630$. We denote the number 0.630 as Y. We provide our Fortis algorithm in Algorithm 17. We highlight that in case more than 2 longest branches exist, the honest miners pick the branch uniformly at random to mine on. Note that generating more than one longest branch is less beneficial for an attacker than extending only one chain; therefore, one of the longest branches can belong to the attacker[6]. Therefore, uniform picking results in less than $1 - \text{Y}$ or Y chance for the attacker's chain, i.e., for all n we have $1/n < 1 - \text{Y}$ and $1/n < \text{Y}$, if he applies OSM or BM (OM), respectively. This is good enough to ensure the incentive compatibility bounds that our scheme provides. We also note that the index of the siblings for determining the fresher chain does not make any difference against the known attacks, but for completeness it is decided based on the timestamps of the oldest forked siblings.

Figure 5.6 shows the revenues of an attacker from OM, BM, and OSM attacks applied on a PoW blockchain where honest miners practice our Fortis algorithm. We note that the simulation results in Section 5.8 confirms our results here.

5.8 Simulation

We simulate our timestamp-based Oracle and Bold mining attacks on Freshness Preferred [Heilman, 2014] and Fortis by implementing these algorithms on top of the blockchain simulator BlockSim [Alharby and van Moorsel, 2020, Alharbi, 2020]. The Python code for our simulation can be found in the repository[7]. We note that the simulation results given in this section are probabilistic and independent of the hardware platform on which the simulation runs. Also, we highlight that we have already simulated Fortis with optimal OSM algorithm [Sapirshtein et al., 2016] while picking the optimum value for Y.

Generic setup. The setup of the simulation mimics the Bitcoin protocol pa-

[6]In case of multiple attackers, there may be multiple longest branches belonging to attackers. Yet, this has been shown as less beneficial than collusion [Liu et al., 2018, Marmolejo-Cossío et al., 2019, Leelavimolsilp et al., 2019]

[7]https://anonymous.4open.science/r/FortisSim-C2F3/

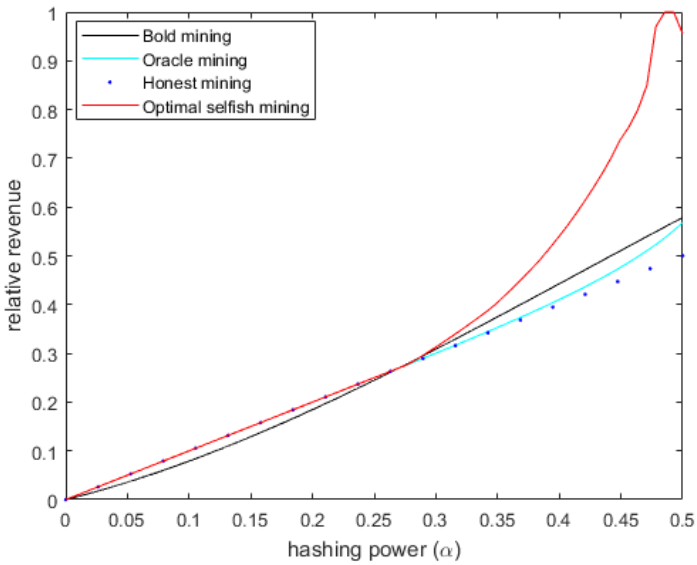
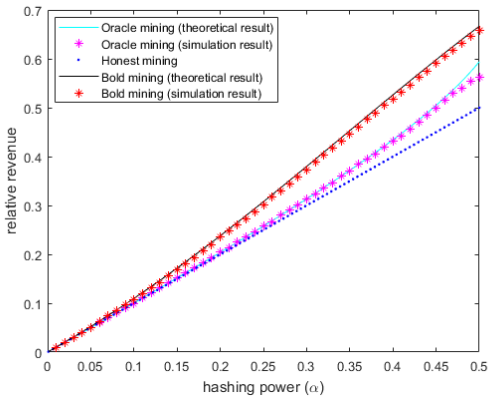


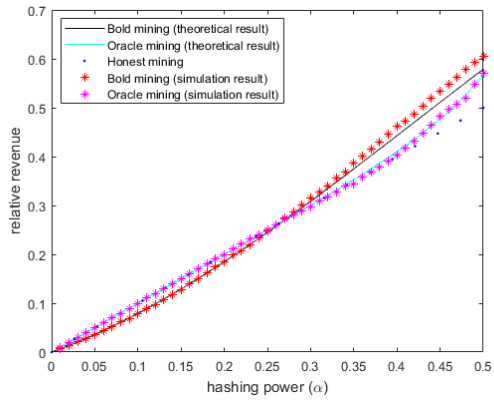
Figure 5.6: The relative revenues of the optimal selfish mining of [Sapirshtein et al., 2016], our Bold mining, and our Oracle mining on our `Fortis` algorithm. The attacker cannot benefit from these attacks, if his hashing power $\alpha < 27.0\%$. We note that for $\alpha < 27.0\%$, optimal selfish mining and Oracle mining generate the honest mining algorithm as the optimum strategy.

rameters of April 2021. Inter-block time and block size limit are set as 600 seconds and 2 MB, respectively. Regarding the block rewards, we assume a fixed block reward regime, since it is well-known that selfish mining attacks become much more minacious in the presence of transaction fees as proposed by [Carlsten et al., 2016]. We leave the study of our attacks and defenses in this setting as a future work. We enabled a single common clock for the miners, as our protocol assumes a global synchronous clock.

Propagation delay. Regarding block propagation delay, it has been considered to generally affect the outcome of selfish mining [Eyal and Sirer, 2018] due to the fact that it is closely related to the network power γ of the attacker [Göbel et al., 2016]. The attacker would like to keep the delay low between him and an honest miner, but high between two honest miners. Considering the Bitcoin tie-breaking mechanism in April 2021 (i.e., a miner follows the first block received), the attacker's aim is to be the first to convey his block. If he achieves



(a) Our Attacks on Freshness Preferred



(b) Our Attacks on `Fortis`

Figure 5.7: Simulation results of relative revenues of the Oracle and Bold miners when the honest mining algorithm is (a) Freshness Preferred and (b) our `Fortis` algorithm. Note that the theoretical revenue calculations from the previous chapters are also provided for reference.

this, the effect is obviously devastating, as it is effectively setting $\gamma = 1$. Somewhat counter-intiutive for a timestamp based scheme, block propagation delays do not have much effect in our attacks and defense. This is because both our defense mechanism `Fortis` and Freshness Preferred of [Heilman, 2014] renders the network power of the attacker less useful (i.e., γ does not directly affect his revenue). Also, those delays are observed in relatively low duration with respect to the inter-block time for an attacker to obtain a high gain from.

To demonstrate the impact of propagation delay in our simulations, we model it as an exponential random variable (in accordance with the findings of [Decker and Wattenhofer, 2013]) with a varying average computed as follows. [Croman et al., 2016] has found that a 1 MB block is received on average in 15.7 seconds. We adjust this for each block as this average time varies linearly with the block size [Gervais et al., 2016]. For simplicity, a constant 3.4 transactions per second are generated and each transaction has a size of 615 bytes. We note that these values are derived from the statistics available on the websites[8] per April 2021. We as-

[8]https://bitinfocharts.com/bitcoin/ and https://bitcoinvisuals.com/chain-tx-size

sume that an attacker (Oracle or Bold miner) and seven honest nodes exist in the system to approximate the big mining pools in the real world cryptocurrencies.

Clock asynchronicity. The clock asynchronicity is a well-known problem in distributed systems and inherently in blockchains. There exist centralized (e.g., Network Time Protocol) and decentralized solutions [Maggs et al., 2012, Kılınç-Alper, 2019]. The centralized ones give more efficient results but their security depends on authorities. To show the effect clock asynchronicity in our simulation, we assume the decentralized protocol of [Kılınç-Alper, 2019] is executed among neighbouring nodes frequently. According to this work, the expected clock difference between two neighbouring nodes is calculated as roughly 0.75 seconds in 12 hours. Modelling Bitcoin, we assume all nodes are not neighbouring. Although we do not have any exact knowledge of the distance between Bitcoin nodes, we estimate it as follows. As of January 2021, there exist about 83,000 full nodes in Bitcoin [Harper, 2021]. Since each full node, by default, connects to 8 separate full nodes [Park et al., 2019]; without any overlaps, the distance between two nodes would be $\log_8 83,000 = 5.45$. This would mean a maximum of $0.75 \times 5.45 = 4.09$ seconds clock skew. However, as there would be inevitably overlaps, we assign random clock differences between each node up to 5 seconds.[9]

Connectivity. In the ones with Freshness Preferred, the attacker's network connectivity is set to the same as the other nodes, since here we aim to show the strength of our attacks. This is contrasted to those with `Fortis`, where we aim to show the strength of our defense and let the attacker's connectivity be perfect. That is, the attacker receives a block as soon as it is found and the others receive each block mined by the attacker immediately.

Results. In our simulations, we conduct the Oracle and Bold mining attacks with hashing powers between and including 1% and 50% with 1% increments against Freshness Preferred and `Fortis` algorithms. Each simulation is repeated

[9]We highlight that deliberately deviating from network clock, as in Oracle mining, has already been disincentivized for the attacker. Further, the effects of deliberate block delaying is covered in $\mathcal{S}_{priv}$ state of the attacks.

5,000 times. Figure 5.7 provides attacker's expected relative revenues from each attack. We attribute the small deviations of the simulation results from theory to propagation delay, connectivity, and clock skew. Still, they are close, which shows that our attacks are effective on Freshness Preferred, and that our `Fortis` provides security up to attacker's hashing power being 27% of the whole system in realistic scenarios.

Chapter 6

M-STABILITY: THRESHOLD SECURITY MEETS TRANSFERABLE UTILITY

6.1 Introduction

Game theory has been applied to many areas including social sciences, economics, biology, law, and cloud security. Regarding the latter, application of game theory has been shown rather useful. This is because it usually leads to more efficient solutions for problems for which impractical protocols have been proposed [Manshaei et al., 2013].

Unlike cryptographic protocols, game theory based solutions depend more on the rationality of the involved parties [Katz, 2008]. It can be seen that in many computer and network security situations, one may assume that malicious actions will not be taken, if they harm the one who takes them. Therefore, one may reduce the threats that she will countermeasure against to those that are beneficial to the attacker. This is especially true for cloud security, as cloud companies usually care about their reputation, which directly affects their income, and hence are likely to act rationally rather than directly maliciously.

Game theory has a large application area in security, including incentivized outsourced cloud computing [Belenkiy et al., 2008, Küpçü, 2017, Avizheh et al., 2019, **?**], distributed systems and consensus protocols [Abraham et al., 2006, Abraham et al., 2013, Bag et al., 2017, Kothapalli et al., 2017], network security and routing [Eidenbenz et al., 2003, Naserian and Tepe, 2009], vehicular networks [Tian et al., 2019], distributed file sharing [Kamara and Küpçü, 2018]. Most of these works either consider simple two-party settings, or multi-party settings where coalitions are not allowed. However, especially in online settings, coali-

tions may be a more prominent issue, as it is hard to disincentivize each party by reputation loss, and sometimes the use of public mechanisms such as the blockchain directly enables coalitions (knowingly or unknowingly) [**?**]. As formalization of the studies against coalitions with up to k parties, k-resiliency definition [Abraham et al., 2006] has been influential. The authors also proposed its extension as (k, t)-robustness to cover malicious players. These definitions have been used by several works, including [Wang et al., 2012, Abraham et al., 2013, Brenguier, 2016, Halpern and Vilaça, 2016]. However, a closer look on these definitions reveals that they have some shortcomings: (i) they are too strict for practice, and mark some secure mechanisms as insecure, (ii) they are not designed against the presence of multiple coalitions. These observations led us to further investigate k-resiliency and (k, t)-robustness definitions to provide improved and practical definitions taking into account potential multiple coalitions' coexistence in the system. We list the main contributions of this chapter as follows:

- To resolve the issue (i) with k-resiliency and (k, t)-robustness definitions, we propose ℓ-repellence against the presence of a single coalition (replacing k-resiliency). If additionally there are arbitrarily deviating players, our proposed (ℓ, t)-resistance replaces (k, t)-robustness. Our definitions incorporate transferable utility assumption in game theory, as it is realistic in many distributed and multi-party computing settings.

- To cope with the issue (ii), we propose m-stability definition against the presence of multiple coalitions. Inspired by threshold security in cryptography [De Santis et al., 1994, Desmedt, 2011], our m-stability definition protects the system against deviation of any number of rational coalitions as long as none of them has more than m members. Note that this is novel, as previous definitions considered only a single deviating coalition existing in the system.

- On three already existing mechanisms, one for incentivized cloud computing, one for forwarding data packages in ad hoc networks, and one for con-

nectivity in ad hoc networks, we separately show applicability and advantage of our novel definitions. We note that none of these works has been analyzed against coalitions previously.

- Regarding incentivized cloud computing, our definitions improve the multi-party mechanism of [Küpçü, 2017] by fine tuning parameter selection to achieve security against large coalitions of size up to one less than all the contractors involved. More concretely, we set tighter relative bounds for rewards and bounties given to the contractors.

6.2 Related Work

Security against Coalitions. For coalition-proofness in the realm of distributed systems and multi-party computation, conventionally k-resiliency and its extended version (k, t)-robustness in presence of Byzantine parties [Abraham et al., 2006] are used. A line of works [Wang et al., 2012, Abraham et al., 2013, Brenguier, 2016, Halpern and Vilaça, 2016] confronts to satisfy this definition. There also exist models as in [Li et al., 2006, Heller, 2008] based on simpler coalitional abilities such as "cheap talk" [Farrell and Rabin, 1996]. In this work, we propose ℓ-repellence, (ℓ, t)-resistance, and m-stability definitions as more flexible proof definitions, instead of the conventional k-resiliency and the related (k, t)-robustness [Abraham et al., 2006] definitions.

6.3 Background on Game Theory and Mechanism Design

An n-player game can be defined by a set of players $N = (P_1, \ldots, P_n)$, the m_i possible actions $\mathcal{A}_i = (A_i^1, \ldots, A_i^{m_i})$ of each player $i \in N$ and a utility function U_i of each player $i \in N$ which assigns a real-value for each possible action vector A that specifies one action for every player in the game. The goal of each player is to maximize her utility. A strategy s_i of player i is a probability distribution over the possible actions of the player i. We denote by $s_i(a)$ the probability the strategy

s_i assigns to the action a. Note that mechanisms are designed to incentivize each player to play a particular strategy. In these mechanisms, a specific action can also be referred to as a strategy. Moreover, we denote by $s_D = (s_{P_a}, \ldots, s_{P_b})$ a strategy profile (an ordered set) comprising the strategies of all the players in the set $D = (P_a, \ldots, P_b)$. We highlight that in many games, some of the strategies that can be chosen by separate players are named the same for simplicity, but as their players are different, they are different and still can be combined in a set as separate elements. Further, we denote by $s_{-i} = s - (s_i)$ the strategy profile of all the players excluding the player i. Similarly, s_{-D} denotes the strategy profile of all the players in a game that are not in D. Given a strategy profile s_N, we denote by $U_i(s_N)$ the expected utility of the player i, if the players in the game play the strategy profile s_N.

Nash Equilibrium. Game theory provides various tools for predicting the outcomes of a game, among which the most commonly used one is the Nash equilibrium which is defined as follows:

Definition 10 (Nash Equilibrium)**.** *For any game with the set N of players, a strategy set $s_N = (s_{P_1}, \ldots, s_{P_n})$ is a Nash equilibrium if $\forall i \in N\ \forall s'_i \neq s_i\ U_i(s_N) \geq U_i(s'_i \cup s_{-i})$.*

Weakly Dominant Strategy. Another useful notion in the analysis of the games is weakly dominant strategy, which is defined for a player as follows:

Definition 11 (Weakly Dominant Strategy of a Player)**.** *A strategy s_i of a player i is its weakly dominant strategy if for all strategies $s'_i \neq s_i$ of i and for all strategy profiles s'_{-i} of players other than i, $U_i(s_i \cup s'_{-i}) \geq U_i(s'_i \cup s'_{-i})$.*

Unlike a Nash equilibrium, a weakly dominant strategy may not always exist in a game. Yet, a strategy set $s_N = (s_{P_1}, \ldots, s_{P_n})$ where each s_i is the weakly dominant strategy of i is a Nash equilibrium, in which case it can be referred to as weakly dominant strategy equilibrium. The definition of weakly dominant strategy of a coalition follows the definition of coalition utility below.

Transferable Utility. In many games involving cooperations, it would be safe to assume that players can engage in binding agreements for how to share the

outcome of a game. In particular, if there exists an available currency to the participants, and this currency is valued equally among them, then this assumption becomes more realistic. We stress that due to the increasing use of cryptocurrencies along with smart contracts, nowadays this assumption is realistic in many problems arising in the realm of computer science. The conventional name given to this assumption is transferable utility assumption [Peters, 2008]. This simplifies the analysis by permitting to define a single utility for a coalition as a whole, and by abstracting out how this utility is shared among the participants internally. The utility of a coalition is defined as follows:

Definition 12 (Coalition Utility)**.** *Let s be a strategy set for a game Π played by the players in a mechanism. Then, the utility $U_C(s)$ of a coalition C is defined as $U_C(s) = \sum_{i \in C} U_i(s)$.*

Weakly dominant strategy of a coalition is defined similar to that of a player as below:

Definition 13 (Weakly Dominant Strategy of a Coalition)**.** *A strategy profile s_C of a coalition C is its weakly dominant strategy if for all strategy profiles $s'_C \neq s_C$ of C and for all strategy profiles s'_{-C} of the players outside C, the following holds: $U_C(s_C \cup s'_{-C}) \geq U_i(s'_C \cup s'_{-C})$.*

We stress that the transferable utility assumption is also useful in cases where a rational adversarial party joins a protocol with multiple identities, e.g., as in Sybil attack in peer-to-peer systems [Douceur, 2002].

Mechanism Design. Mechanism Design (ME) can be considered as an application of game theory to achieve a goal by incentivizing the rational players for a particular strategy set s_N, where N is the set of all players. Given some desired functionality $\mathcal{F}$, we say that (Π, s_N) is a mechanism for $\mathcal{F}$ if the outcome of (Π, s_N) satisfies $\mathcal{F}$ and the players are incentivized to play s_N.

BAR Model. First, proposed by [Aiyer et al., 2005], Byzantine (B), altruistic (A), and rational (R) model (or BAR model) is the commonly used model in mechanism design for distributed systems. The model defines three types of players:

the Byzantine ones (i.e., the ones that can choose any possible strategy, no matter what their utilities are), the altruistic ones (i.e., the ones that always choose the honest strategy, no matter what their utilities are), and the rational ones (i.e., the ones that always choose the strategy that maximizes their utilities). The model itself does not provide a distribution for these types of players in a given game, but rather is useful for asserting and proving statements such as a desirable functionality can be achieved as long as the number of rational and or Byzantine players are below certain bounds.

***k*-Resiliency.** [Abraham et al., 2006] proposes the k-resilient, t-immune, and (k,t)-robust mechanism definitions to protect the mechanism outcome against coalitions and Byzantine players as follows:

Definition 14 (k-Resilient Mechanism)**.** *Given a player set $C \subseteq N$, s_C is a group best response for C to s_{-C}, if for all strategies s'_C played by C and $\forall i \in C$, we have $U_i(s_C \cup s_{-C}) \geq U_i(s'_C \cup s_{-C})$. A joint strategy s_N is a k-resilient equilibrium, if $\forall C \subseteq N$ with $|C| \leq k$, s_C is a group best response for C to s_{-C}, where $s_N = s_C \cup s_{-C}$. Given some desired functionality $\mathcal{F}$, we say that (Π, s_N) is a k-resilient mechanism for $\mathcal{F}$, if s_N is a k-resilient equilibrium of Π and the outcome of (Π, s_N) satisfies $\mathcal{F}$.*

Definition 15 (t-Immune Mechanism)**.** *A joint strategy s_N is a t-immune equilibrium, if $\forall T \subseteq N$ with $|T| \leq t$, for all strategies s'_T played by the players in T, and $\forall i \notin T$, we have $U_i(s'_T \cup s_{-T}) \geq U_i(s_N)$, where $s_N = s_T \cup s_{-T}$. Given some desired functionality $\mathcal{F}$, we say that (Π, s_N) is a t-immune mechanism for $\mathcal{F}$, if s_N is a t-resilient equilibrium of Π and the outcome of (Π, s_N) satisfies $\mathcal{F}$.*

Definition 16 ((k,t)-Robust Mechanism)**.** *A joint strategy s_N is a (k,t)-robust equilibrium, if $\forall C, T \subseteq N$ s.t. $C \cap T = \emptyset$, $|C| \leq k$, and $|T| \leq t$, for all strategies s'_T played by the players in T, and for all strategies s'_C played by C, and $\forall i \in C$, we have $U_i(s_{-T} \cup s'_T) \geq U_i(s_{-(C \cup T)} \cup s'_C \cup s'_T)$, where $s_N = s_C \cup s_T \cup s_{-(C \cup T)}$. Given some desired functionality $\mathcal{F}$, we say that (Π, s_N) is a (k,t)-robust mechanism for $\mathcal{F}$, if s_N is a (k,t)-robust equilibrium of Π and the outcome of (Π, s_N) satisfies $\mathcal{F}$.*

6.4 Shortcomings of Existing Definitions

In this section, we show some limitations of the existing k-resiliency and (k,t)-robustness definitions for security against coalitions. That is, we provide some example hypothetical games where these definitions indeed fail to satisfy expectations by making it too hard to satisfy (i.e., resulting unnecessary hardness in practice). Also, we elaborate on the limitations of some well-known notions from cooperative game theory for use in security, i.e., they fail to capture coalition strategies well enough (i.e., resulting in security breaches). None of the games provided in this section has been deduced from any specific known game, instead, we have composed them to clarify our argument. Yet, due to the simplicity of these games, they are likely to appear in mechanisms from real life or literature. We show the drawbacks of k-resiliency with existing mechanisms from computer science literature in Section 6.6.

k-Resiliency is too Strict. Suppose that one needs a mechanism played by a player set N with a desired strategy set s_N. The mechanism has already shown to be Nash equilibrium. Yet, it is needed to be stable against potential coalitions of any 2 players. W.l.o.g., we are interested in the coalition strategies of two particular players, named Alice and Bob. Assuming all the other players play the honest strategy $s_{-(A\cup B)}$, Table 6.1 shows the utilities (u_A, u_B) of Alice and Bob from honest strategies s_A and s_B, and deviant strategies s'_A and s'_B, respectively. Notice that the weakly dominant strategies of Alice and Bob are (s_A, s_B).

Alice ╲ Bob	s_B	s'_B
s_A	$(5,5)$	$(7,2)$
s'_A	$(2,7)$	$(4,4)$

Table 6.1: The utilities (u_A, u_B) of Alice and Bob from honest strategies s_A and s_B, and deviant strategies s'_A and s'_B.

According to the k-resiliency definition given in Definition 14, this mecha-

nism is not even 2-resilient as there exists a coalition $C = (\text{Alice,Bob})$, with at least one deviant coalition strategy s'_C such that at least one member of the coalition has a greater utility than the one obtained from (s_A, s_B). In fact, there exist two such strategies (s_A, s'_B) and (s'_A, s_B). That is, compared to the honest strategy, the former delivers greater utility for Alice, and the latter is more beneficial to Bob. However, assuming both Alice and Bob are in the coalition for rational purposes, neither Alice would play s_A, nor Bob would play s_B, as otherwise their utilities would decrease. Therefore, there exist games where k-resiliency is too strict to achieve, yet still secure against a coalition based on individual rationality assumption.

The Definitions from Cooperative Game Theory are too Gentle. The cooperative game theory and mechanism design overcomes this issue by notions such as "strong Nash equilibrium" [Aumann, 1959] and "coalition-proof Nash equilibrium" [Bernheim et al., 1987]. These notions are based on Pareto-optimality, i.e., for their satisfaction there should not be any coalition C with a strategy s'_C that delivers at least the same utility as s_C to each player in C and greater utility than s_C to at least one player in C. We could corporate a definition based on these, but we identify the following issue. Let us change the utility matrix of Alice and Bob in the above mentioned game as in Table 6.2, assuming all the other parameters are kept unchanged. Notice that the weakly dominant strategies of Alice and Bob are still (s_A, s_B).

Alice ╲ Bob	s_B	s'_B
s_A	$(5,5)$	$(\mathbf{10},2)$
s'_A	$(2,7)$	$(4,4)$

Table 6.2: The utilities (u_A, u_B) of Alice and Bob from honest strategies s_A and s_B, and deviant strategies s'_A and s'_B.

The problem with this mechanism is that Alice can offer his coalition partner Bob a transfer of value 4 from his account to hers, if he plays s'_B. Then, she

would play s_A and their utility would become $(6, 6)$, which beats the utility $(5, 5)$ from the honest strategy. Unfortunately, due to the following reasons, this example issue is significant for mechanism design for multi-party protocols and distributed systems. First, with advent and prevalence of cryptocurrencies, the smart contract schemes that can enforce such binding agreements are now available to anyone who can connect to internet. Second, many of the arising problems involve utilities strictly built upon costs, fines, rewards, etc., which can be easily converted to monetary utilities.

This issue is named transferable utility and handled with the notion "core property" [Shapley and Shubik, 1969] in cooperative game theory. Essentially, a strategy set has core property if no coalition can have a greater total utility from another strategy. Our definitions combine this idea with threshold security [De Santis et al., 1994, Desmedt, 2011] idea as in k-resiliency, and improve upon it by including Byzantine parties that can arbitrarily deviate.

6.5 Our Security Definitions Against Coalitions

In this section, we propose our definitions to replace k-resiliency and (k, t)-robustness, due to the shortcomings of them mentioned in Section 6.4.

6.5.1 Security Definitions Against a Single Coalition

First we provide ℓ-repellent mechanism definition, as a direct replacement of k-resiliency.

Definition 17 (ℓ-Repellent Mechanism)**.** *Given a player set $C \subseteq N$, s_C is a best collective response for C to s_{-C}, if for all strategies s'_C played by C, we have $U_C(s_C \cup s_{-C}) \geq U_C(s'_C \cup s_{-C})$. A joint strategy s_N is an ℓ-repellent equilibrium, if $\forall C \subseteq N$ with $|C| \leq \ell$, s_C is a best collective response for C to s_{-C}, where $s_N = s_C \cup s_{-C}$. Given some desired functionality $\mathcal{F}$, we say that (Π, s_N) is an ℓ-repellent mechanism for $\mathcal{F}$, if s_N is an ℓ-repellent equilibrium of Π and the outcome of (Π, s_N) satisfies $\mathcal{F}$.*

In line with the previous (k,t)-robustness definition, we also provide (ℓ,t)-resistant mechanism definition to comprise cases where a collaboration of a set of players and a set of arbitrarily acting players coexist. The following definition is expected to be used in mechanism design instead of (k,t)-robustness.

Definition 18 ((ℓ,t)-Resistant Mechanism)**.** *A joint strategy s_N is an (ℓ,t)-resistant equilibrium, if $\forall C,T \subseteq N$ s.t. $C \cap T = \emptyset$, $|C| \leq \ell$, and $|T| \leq t$, for all strategies s'_T played by the players in T, and for all strategies s'_C played by C, we have $U_C(s_{-T} \cup s'_T) \geq U_C(s_{-(C\cup T)} \cup s'_C \cup s'_T)$, where $s_N = s_C \cup s_T \cup s_{-(C\cup T)}$. Given some desired functionality $\mathcal{F}$, we say that (Π, s_N) is an (ℓ,t)-resistant mechanism for $\mathcal{F}$, if s_N is an (ℓ,t)-resistant equilibrium of Π and the outcome of (Π, s_N) satisfies $\mathcal{F}$.*

6.5.2 Extension Against Multiple Coalitions

We extend our ℓ-repellence and (ℓ,t)-resistance definitions for systems where multiple coalitions can form. The extension that we provide here is inspired by threshold cryptography, and allows coalitions up to certain thresholds.

Definition 19 (m-Stable Mechanism)**.** *A joint strategy s_N is an m-stable equilibrium, if for all natural numbers $p \leq |N|$ and for all coalitions $C_1, ..., C_p$ satisfying following conditions:*

- $1 \leq |C_1|, \ldots, |C_p| \leq m$,
- $C_1 \cup \ldots \cup C_p = N$,
- $\forall i,j \in (1,\ldots,p)$ *s.t.* $i \neq j$, $C_i \cap C_j = \emptyset$;

we have that for each $i = 1, ..., p$, the strategy s_{C_i} is weakly dominant for the coalition C_i and that $s_{C_1} \cup \ldots \cup s_{C_p} = s_N$. Given some desired functionality $\mathcal{F}$, we say that (Π, s_N) is an m-stable mechanism for $\mathcal{F}$, if s_N is an m-stable equilibrium of Π and the outcome of (Π, s_N) satisfies $\mathcal{F}$.

We stress that m-stability implies m-repellence. This follows from that in an m-stable mechanism, for any coalition of size up to m the honest strategy is the

weakly dominant strategy. As a consequence, if a coalition in this size range deviates from the honest strategy, it ends up with a utility that is less or equal.

In some sense, one may consider ℓ-repellence as similar to Nash equilibrium, and m-stability as similar to weakly dominant strategy equilibrium. ℓ-repellence disincentivizes a rational coalition, only in case the coalition believes that all other parties will play the honest strategy. m-stability, on the other hand, further disincentivizes a coalition from deviating, in case the coalition knows some other deviating coalitions. Even if the coalition anticipates that other players may make honest mistakes (e.g., due to miscalculation) in choosing or playing their strategies, the coalition is still incentivized for honesty. This is why ℓ-repellence does not imply ℓ-stability.

Remark 3. *m-stability necessitates each strategy s_i to be the weakly dominant strategy for C_i, i.e., no matter what all other coalitions or players choose, C_i is always better by choosing s_i. Therefore, unlike extension of ℓ-repellence to (ℓ, t)-resistance, an extended definition for involving Byzantine parties in m-stability is redundant. Recall that t-immunity prevents any harm from arbitrarily deviating parties to players of the honest strategy, hence it may still be separately used.*

6.5.3 Comparison to Previous Definitions

Although compared to k-resiliency definition (i.e., Definition 14) ℓ-repellence is weaker, it is sufficient in cases where the players are rational in choosing to be part of a coalition. We consider that this is a reasonable assumption in many systems where mechanism design ideas are applied, as the main target of these systems is to enforce the rational players towards the system goal.

As long as ℓ-repellence definition is satisfied in a mechanism, no matter the utility distribution among the coalition players is, either all of them obtain the same utility from a deviant strategy as the one from desired strategy or at least one player obtains less utility. Regarding the former case, k-resiliency also does not provide any protection as well. Regarding the latter case, ℓ-repellence unavoidably entails that particular player not to obey the coalition strategy, which

provides enough protection for the mechanism's desired functionality.

We emphasize that k-resiliency and (k,t)-robustness imply k-repellence and (k,t)-resistance, respectively. The proofs of these statements are intuitive, hence we will not provide them here. However, the converses of these statements are not true.

We acknowledge that we are unable to detect trivial implications between t-immunity and our definitions, hence one may need to prove a mechanism separately for them. The only trivial implication is for zero-sum mechanisms given as follows:

Theorem 2. *If (Π, s_N) is a zero-sum t-immune mechanism, then it is also a t-repellent mechanism.*

Proof. Assume that (Π, s_N) is a zero-sum t-immune mechanism. Also, assume that (Π, s_N) is not a t-repellent mechanism. Then, there exists a player set T with some strategy profile s'_T s.t. $|T| \leq t$ and $U_T(s'_T \cup s_{-T}) > U_T(s_N)$. Due to zero-sum property, this results in less total utility for players that are not in T. Therefore, there exists a player i that is not in T, for which $U_i(s'_T \cup s_{-T}) < U_i(s_N)$. This means (Π, s_N) is not a zero-sum t-immune mechanism, which contradicts the assumption in the beginning of this proof. □

Regarding m-stability, we have the following implications for (ℓ, t)-resistance. Note that as above mentioned, m-stability trivially implies m-repellence.

Lemma 4. *If (Π, s_N) is an m-stable mechanism, then it is a $(m, |N| - m)$-resistant mechanism.*

Proof. Assume that (Π, s_N) is an m-stable mechanism. Also, assume that it is not an $(m, |N| - m)$-resistant mechanism. Then, there exist sets $C, T \subseteq N$ such that $C \cap T = \emptyset$, $|C| \leq m$, $|T| \leq |N| - m$, $U_C(s'_C \cup s'_T \cup s_{-(C \cup T)}) > U_C(s'_T \cup s_{-T})$. Therefore, $s_C \subseteq s_N$ is not a weakly dominant strategy for C. This means (Π, s_N) is not an m-stable mechanism, which contradicts the assumption in the beginning of this proof. □

Others ╲ This	Diligent	Lazy
All diligent	$r - \mathsf{cost}(1)$	$rq - \big(f + b(n-1)\big)(1-q) - \mathsf{cost}(q)$
k lazy	$r + b(1-q) - \mathsf{cost}(1)$	$rq - \big(f + \frac{b(n-k-1)}{k+1}\big)(1-q) - \mathsf{cost}(q)$
All lazy	$r + b(1-q) - \mathsf{cost}(1)$	$r - \mathsf{cost}(q)$

Table 6.3: The expected utility of each contractor from choosing diligent or lazy with respect to the other players' chosen strategies (where $0 < k < n$) in the outsourced computation mechanism of [Küpçü, 2017].

Theorem 3. *If (Π, s_N) is an m-stable mechanism, then $\forall \ell \leq m$ it is an $(\ell, |N| - \ell)$-resistant mechanism.*

Proof. Assume that (Π, s_N) is an m-stable mechanism. Also, assume that $\exists \ell \leq m$ s.t. it is not an $(\ell, |N| - \ell)$-resistant mechanism. Then by Lemma 4, (Π, s_N) is not an ℓ-stable mechanism. As $\ell \leq m$, this means (Π, s_N) is not an m-stable mechanism, which contradicts the assumption in the beginning of this proof. □

6.6 Example Mechanisms from Literature

In this section, in order to show the usefulness of our definitions, we apply them to some example problems from cloud computing and network security literature. For comparison and reference, we also analyze them with the previous k-resiliency definition.

6.6.1 Example 1: Incentivized Outsourced Computation

We consider the setting provided in [Küpçü, 2017]. We briefly describe it as follows. The setting involves a boss and, as players, n rational contractors. The boss has a costly algorithm, of which execution she wants to outsource to the n contractors. Each contractor can choose either the diligent strategy or the lazy strategy. The former means that it runs the correct algorithm whose cost is denoted as $\mathsf{cost}(1)$. The latter means that it runs a less costly deterministic algorithm (called "q algorithm") which gives the correct output with probability q and has

a cost denoted as $\mathsf{cost}(q)$. Further, according to [Küpçü, 2017], the q algorithm run by all lazy players is assumed to be the same. If all contractors return the same output, the boss just accepts it as the correct one and gives the reward r to each of them. Otherwise, the diligent contractors execute a protocol with the boss to catch the lazy ones. In this case, the diligent ones receive the reward r and a bounty b. A share for the total bounties of the diligent contractors plus a fixed fine f is collected from each contractor. Table 6.3 provides the resulting expected utility matrix. We note that for this mechanism to be meaningful, it is necessary that $\mathsf{cost}(q) < \mathsf{cost}(1) < r$. [Küpçü, 2017] has shown that the boss should set $b > r/(1-q)$ to have all diligent as the unique Nash equilibrium (without any restriction on the fine f) if no coalition is allowed. While [Küpçü, 2017] suggests setting $b \approx r$ is sufficient for practical purposes, our findings in Theorem 5 show that it can even be considered close to optimal, as $b \leq r(n-1)/(n-2)$ guarantees security against large coalitions. Therefore, applying our definitions lead to better understanding of the existing mechanisms not only from a theoretical viewpoint, but also with practical importance in setting the system parameters.

This mechanism is a good example for the limitation of the previous k-resiliency definition, and how it can mark as insecure a mechanism secure against large coalitions (i.e., up to one less than all participants) by our definitions.

Theorem 4. *The incentivized outsourced computation mechanism of [Küpçü, 2017] is not 2-resilient.*

Proof. Let C be a coalition of two arbitrary players i and j. Let s'_C be the strategy where i and j play the lazy and the diligent strategies, respectively. The utility of j becomes $U_j(s'_C \cup s_{-C}) = r + b(1-q) - \mathsf{cost}(1)$, which is greater than $U_j(s_N) = r - \mathsf{cost}(1)$. □

Regarding our definitions, we only show that $(n-1)$-stability of this mechanism, which implies $(n-1)$-repellence. Further, by Theorem 3, for all $\ell \leq n-1$ it implies$(\ell, n-\ell)$-resistance. The theorem given below concerns only mechanisms with size $n > 2$, since for $n = 2$ and $b > r/(1-q)$, this outsourced com-

putation mechanism is already 1-stable without any upper bound for b (i.e., for each player, the diligent strategy has been shown as weakly dominant by [Küpçü, 2017]).

Theorem 5. *For $n > 2$, if the boss sets the reward and bounty as $r(n-1)/(n-2) \geq b > r/(1-q)$, the incentivized outsourced computation mechanism of [Küpçü, 2017] is $(n-1)$-stable.*

Proof. Let C be a coalition of size $|C| \leq n-1$. W.l.o.g, we need to show the all diligent strategy s_C is its weakly dominant strategy.

Case 1. Assume all the players outside C play diligent strategy. Let s_{-C} denote their strategy profile. Also, let $s_N = s_C \cup s_{-C}$. Then, $U_C(s_N) = |C| \cdot \big(r - \mathsf{cost}(1)\big)$. Let s'_C be a strategy profile for C s.t. k players in C play lazy. If the q algorithm returns the correct output (i.e., with probability q), the coalition utility compared to honest strategy only changes due to the decrease in the cost of the executed algorithm by the lazy players. More concretely, the utility of the coalition becomes $U_C(s'_C \cup s_{-C}) = U_C(s_N) + k \cdot \big(r - \mathsf{cost}(q)\big) - k \cdot \big(r - \mathsf{cost}(1)\big) = U_C(s_N) + k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big)$. If the q algorithm returns some incorrect output (i.e., with probability $1-q$), the coalition utility compared to honest strategy changes due to failed rewards and paid fines by the lazy players, bounties paid to players outside of the coalition, and the decrease in the cost of the executed algorithm. More concretely, the utility of the coalition becomes $U_C(s'_C \cup s_{-C}) = U_C(s_N) - kr - kf - b(n - |C|) + k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big)$. We deduce the expected utility of the coalition as $U_C(s'_C \cup s_{-C}) = U_C(s_N) + k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big) + (1-q)\big(-kr - kf - b(n - |C|)\big)$. Then, we calculate $k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big) + (1-q)\big(-kr - kf - b(n - |C|)\big) \leq kr + (1-q)\big(-kr - kf - b(n - |C|)\big) \leq kr + (1-q)\big(-kr - b(n - |C|)\big) \leq kb(1-q) + (1-q)\big(-kr - b(n - |C|)\big) = -(1-q)kr + b(1-q)(k - n + |C|)$. For $r \geq \frac{b(k-n+|C|)}{k}$, we obtain $U_C(s'_C \cup s_{-C}) \leq U_C(s_N)$. As these inequalities need to hold for all $k \leq |C|$ and all $|C| < n$, we need to find the maximum of the lower bound of r. For this purpose, we first set $|C| = n-1$, which can be done, since for any possible value of k, the value of $|C|$ can be $n-1$. Then we obtain $r \geq \frac{b(k-1)}{k}$. Again, for the maximum of the lower bound of r, we set $k = n-1$, which can

be done, since we have set $|C| = n-1$ and still have $k \leq |C|$. At the end, we obtain the lower bound of r as $r \geq \frac{b(n-2)}{n-1}$, or alternatively the upper bound of b as $b \leq \frac{r(n-1)}{n-2}$ for $n > 2$.

Case 2. Assume $y > 0$ and $z > 0$ players outside C play lazy and diligent strategies, respectively, s.t. we have $y + z + |C| = n$. Let s'_{-C} denote their strategy profile. Let s'_C be a strategy profile for C s.t. k players in C play lazy. If the q algorithm returns the correct output (i.e., with probability q), the utility of the coalition change due to the decrease in the cost of the executed algorithm by the lazy players in the coalition. More concretely, the utility of the coalition becomes $U_C(s'_C \cup s'_{-C}) = U_C(s_C \cup s'_{-C}) + k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big)$. Note that no bounty is received by any player, as the lazy outside the coalition also benefit from the correct output of the q algorithm (as all the players assumed to be running the same deterministic q algorithm). If the q algorithm returns some incorrect output (i.e., with probability $1-q$), the coalition utility changes due to receiving less bounties and rewards, fines paid by the lazy players, bounties paid to the players outside of the coalition, and the decrease in the cost of the executed algorithm by the lazy players. More concretely, the utility of the coalition becomes $U_C(s'_C \cup s'_{-C}) = U_C(s_C \cup s'_{-C}) - kb - kr - kf - bz \cdot \frac{k}{k+y} + k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big)$. We deduce the expected utility of the coalition as $U_C(s'_C \cup s'_{-C}) = U_C(s_C \cup s'_{-C}) - (1-q)\Big(kb + kr + kf + bz \cdot \frac{k}{k+y}\Big) + k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big)$. Then, we calculate $-(1-q)\Big(kb + kr + kf + bz \cdot \frac{k}{k+y}\Big) + k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big) \leq -(1-q)\Big(kb + kr + kf + bz \cdot \frac{k}{k+y}\Big) + kr \leq -(1-q)\Big(kb + kr + kf + bz \cdot \frac{k}{k+y}\Big) + kb(1-q) = -(1-q)\Big(kr + kf + bz \cdot \frac{k}{k+y}\Big) \leq 0$. For all k and $|C|$ in the relevant range, we obtain $U_C(s'_C \cup s'_{-C}) \leq U_C(s_C \cup s'_{-C})$. We stress that the main difference from Case 1 occurs due to loss of the bounties by lazy when playing s'_{-C} in this case.

Case 3. Assume all the players outside C play lazy strategy. Let s''_{-C} denote their strategy profile. We check the deviation strategies in two separate subcases below.

Let s'_C be a strategy profile for C s.t. $k < |C|$ players in C play lazy. If the q algorithm returns the correct output (i.e., with probability q), the coalition utility

changes due to the decrease in the cost of the executed algorithm by the lazy players in the coalition. More concretely, the utility of the coalition becomes $U_C(s'_C \cup s''_{-C}) = U_C(s_C \cup s''_{-C}) + k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big)$. Note that no bounty is received by any player, as the lazy outside the coalition also benefit from the correct output of the q algorithm (as all the players assumed to be running the same deterministic q algorithm). If the q algorithm returns some incorrect output (i.e., with probability $1-q$), the coalition utility changes due to receiving less bounties and rewards, fines and bounties paid by the lazy players, and the decrease in the cost of the executed algorithm by the lazy players. More concretely, the utility of the coalition becomes $U_C(s'_C \cup s''_{-C}) = U_C(s_C \cup s''_{-C}) - kb - kr - kf - \frac{(|C|-k)bk}{|N-C|+k} + k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big)$. Hence, the expected utility of the coalition is $U_C(s'_C \cup s''_{-C}) = U_C(s_C \cup s''_{-C}) - (1-q)\Big(kb + kr + kf + \frac{(|C|-k)bk}{|N-C|+k}\Big) + k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big)$. We calculate $-(1-q)\Big(kb + kr + kf + \frac{(|C|-k)bk}{|N-C|+k}\Big) + k \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big) \leq -(1-q)(kb + kr + kf) + kr \leq -(1-q)(kb + kr + kf) + kb(1-q) = -(1-q)(kr + kf) \leq 0$. Therefore, for all k and $|C|$ in the relevant range, we obtain $U_C(s'_C \cup s''_{-C}) \leq U_C(s_C \cup s''_{-C})$.

Now, let s'_C be a strategy profile for C s.t. all players in C play lazy. We use the coalition utility $U_C(s_N)$ from all-diligent as reference to obtain $U_C(s'_C \cup s''_{-C}) = U_C(s_N) + |C| \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big) = U_C(s_C \cup s''_{-C}) - |C|b(1-q) + |C| \cdot \big(\mathsf{cost}(1) - \mathsf{cost}(q)\big)$. As $\mathsf{cost}(1) - \mathsf{cost}(q) \leq r \leq b(1-q)$, for all $|C|$ in the relevant range, we have $U_C(s'_C \cup s''_{-C}) \leq U_C(s_C \cup s''_{-C})$. □

Theorem 6. *The incentivized outsourced computation mechanism of [Küpçü, 2017] is $(n-1)$-immune.*

Proof. Since there exists at least one player acting honestly, the all lazy utility cannot be achieved. If all the players act diligently, the utility of an honest player is $r - \mathsf{cost}(1)$. For any other outcome, the utility of an honest player turns out $r + b(1-q) - \mathsf{cost}(1)$, which is greater than the former. □

Remark 4. *The incentivized outsourced computation mechanism of [Küpçü, 2017] analyzed above cannot trivially be altered to satisfy k-resiliency by replacing the all diligent*

strategy utility (i.e., $r - \mathsf{cost}(1)$) with ($r + b(1-q) - \mathsf{cost}(1)$), since this means that the boss hands $r + b(1-q)$ as reward to each contractor in case of consensus of the output. In turn, this would enforce the all lazy utility to become $r + b(1-q) - \mathsf{cost}(q)$, providing another Nash equilibrium to the system, i.e., all lazy. Yet, the proposed mechanism of [Küpçü, 2017] with setting $r(n-1)/(n-2) \geq b > r/(1-q)$ satisfies our proposed definitions, and provides strong security against coalitions with up to $n-1$ rational players.

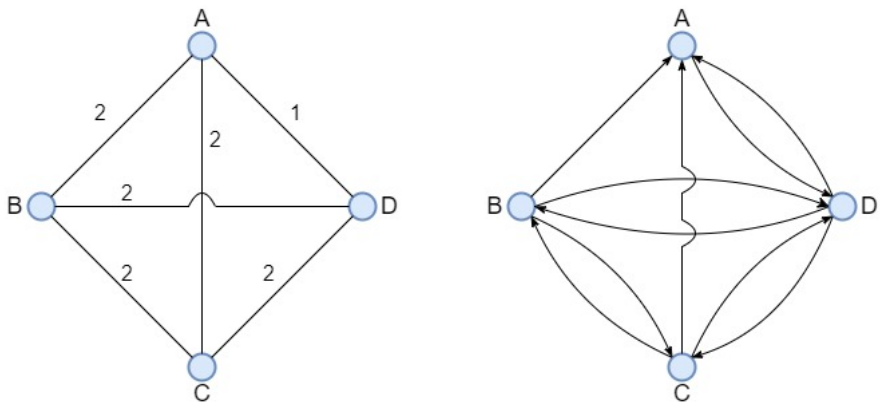
6.6.2 Example 2: Forwarding Dilemma

We consider the "forwarding dilemma" introduced by [Naserian and Tepe, 2009] to model forwarding of a flooded packet in a wireless ad hoc network. We briefly describe it as follows. In this game, players are network nodes who has received the same flooded packet. Regarding this packet, each player has two strategies: forward it or drop it. [Naserian and Tepe, 2009] shows that desirable strategies of this game are those with one player that plays forward, while the rest plays drop. Here, the number of forwarding players needs limiting to 1 in order to avoid excess bandwidth overhead.

There are two values that affect the utility of each player, namely the network gain factor g and the forwarding cost c. The utilities of the players are defined according to her and other players' strategies as in Table 6.4. Obviously, $c < g$ is required for incentivizing one forward and the rest drop.

Others ╲ This	forward	drop
All drop	$g - c$	0
At least one forward	$g - c$	g

Table 6.4: The utility of each player from choosing forward or drop with respect to the other players' chosen strategies in the forwarding dilemma of [Naserian and Tepe, 2009].



(a) Initial graph H (b) Induced graph G

Figure 6.1: Graphs related to an instance of the strong connectivity game of [Eidenbenz et al., 2003]. The one given in (a) defines the game, and the one given in (b) shows the induced graph if the players A, B, C, and D choose radii 1, 2, 2, and 2, respectively.

In the beginning of each game a mediator assigns forwarding to one of the player and dropping to the remaining players [1]. According to [Naserian and Tepe, 2009], this strategy profile is a Nash equilibrium. Let N be the set of players with $|N| = n$ and P be the player who is assigned forwarding by the mediator.

We show that according to k-resiliency definition, this mechanism is not incentive compatible even in the presence of coalitions with 2 players.

Theorem 7. *The given mechanism of forwarding dilemma game is not 2-resilient.*

Proof. Let $C \subseteq N$ consist of P and another player, say P'. If P drops and P'

[1] In mechanism design, a mediator is generally used for coordination of players in playing an equilibrium, in case there exist multiple equilibria. The mediator's job is just to assign a strategy profile for all players in the beginning of a game. We highlight that it is not an authority and that it cannot enforce any particular strategy to any player. A mediator can even be implemented as a deterministic algorithm that is executed by each player on some common input before the game starts. Regarding the forwarding dilemma, the algorithm may select one of the players for forward strategy (e.g., a linear search for the player with the smallest address or id encoding). Clearly, choice of the mediator and its existence does not affect our analysis and results.

forwards, the utility of P increases to g from $g - c$. Thus, this mechanism is not 2-resilient. □

On the other hand, this mechanism is secure in the presence of a single coalition, according to our ℓ-repellent definition.

Theorem 8. *The given mechanism of forwarding dilemma game is n-repellent.*

Proof. We show that any coalition cannot receive more utility than the honest strategy. W.l.o.g, let $C \subseteq N$ be a coalition of size at most n. Assume players outside of the coalition follow mediator.

Case 1. Assume P is not in C. Then since each player in the C already receives g, the payoff of the coalition cannot increase by any change of the strategy of the coalition.

Case 2. Assume P is in C. If they follow the honest strategy, the payoff of the coalition is $|C| \cdot g - c$. If nobody in the coalition forwards, the payoff of the coalition is 0. If more than one player in the coalition forwards, the payoff of the coalition is at most $|C| \cdot g - 2 \cdot c$. Thus, there is no strategy for the coalition C such that the coalition receives greater payoff than the honest strategy. □

We again observe a good example mechanism that is marked completely insecure against a coalition (not even 2-resilient) by the definition of [Abraham et al., 2006]. However, by our ℓ-repellence definition, it is secure no matter what the size of the coalition is. For completeness, we also show that unfortunately this game does not satisfy our other definitions and t-immunity.

Theorem 9. *The given mechanism of forwarding dilemma game is not* $(1, 1)$*-resistant.*

Proof. Let C be any coalition of size 1 and $P \notin C$. Also, let $T = (P)$. If P deviates and chooses to drop, then C receives 0 by following the mediator while its utility becomes $g - c$, when the player in the coalition deviates and forwards. □

Theorem 10. *The given mechanism of forwarding dilemma game is not* 1*-immune.*

Proof. If P deviates by dropping, then the utility of each player from the honest strategy decreases to 0. □

Theorem 11. *The given mechanism of forwarding dilemma game is not* 1*-stable.*

Proof. As neither forwarding nor dropping is a weakly dominant strategy, clearly we do not have 1-stability in this game. □

6.6.3 Example 3: Strong Connectivity

We consider the "strong connectivity" game investigated by [Eidenbenz et al., 2003] for connectivity problems in an ad hoc network where nodes are selfish. We summarize their results related to our work as follows. They observe wireless networks with omni-directional antennas. The network topology can be represented by an undirected, weighted, complete graph $H(V, E', w)$, where V denotes the vertices with $|V| = n$, E' denotes edges and w is a weight vector with w_e is weight of an edge $e \in E'$.

The players in this game are the nodes of the graph H. Strategy of each player is choosing a radius r. Radii choices of the nodes induce a directed, unweighted graph $G(V, E)$ such that $e = (u, v) \in E$ if and only if $r_u \geq w_e$. We say that a node u can reach another node v, if there exists a path between them in the induced graph. Then, the utility of a node v is defined as $-r_v^\alpha$ if v can reach every other node for some constant distance power gradient α. If there is a node that v cannot reach, its utility is $-\infty$.

The main goal of the network designer here is strong connectivity, i.e. each node can reach any other node. Here, we consider an instance of strong connectivity game with four nodes $\mathsf{A}, \mathsf{B}, \mathsf{C}$ and D and the initial graph shown in Figure 6.1*a*. There exists a mediator who assigns a radius to each player [2]. Specifically, here it assigns radii $(1, 2, 2, 2)$ to $\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}$. The induced graph G for this game is given in Figure 6.1b.

Theorem 12. *The given mechanism of strong connectivity game is not 2-resilient.*

[2] In line with the mediator of forwarding dilemma, here it can again be a trivial deterministic algorithm executed by each player on some common input in the beginning.

Proof. Consider the coalition $C = (\mathsf{A}, \mathsf{D})$. If this mechanism were 2-resilient, A choosing 1 and D choosing 2 as radius would be a group best response. For this strategy, utility of D is -2^{α}. On the other hand, if D chooses 1 and A chooses 2 as radius, utility of D becomes -1^{α}, which is greater than -2^{α}. Hence, the former group strategy is not a group best response. Consequently, this mechanism is not 2-resilient. □

Theorem 13. *The given mechanism of strong connectivity game is* 4*-repellent.*

Proof. Assume that this mechanism is not 4-repellent. Then, there exists a coalition C with a size up to 4 s.t. there exists a strategy that makes the utility of C higher. We note that in this mechanism, any node other than D cannot decrease her utility without losing its all connections. Thus, D must be a part of the coalition, and must decrease her radius to 1. Note that D cannot choose a radius lower than 1, since it would prevent her from connecting to any other node. When D decreases her radius to 1, she can only connect to A. Thus, for them to reach other nodes A must be a part of the coalition and it should change her strategy as well. The only reasonable joint deviation is A choosing 2 and D choosing 1. However, for this deviation the utility of the coalition does not change. By contradiction we have that this mechanism is 4-repellent. □

Observe that this mechanism is not 2-resilient, yet it is repellent against even a coalition of all the players. For completeness, we also analyze it with our other definitions and t-immunity.

Theorem 14. *The given mechanism of strong connectivity game is not* $(1, 1)$*-resistant.*

Proof. Let D be a Byzantine player and choose 1 as radius. Then, A cannot connect to B or C by choosing radius 1, which results with a payoff $-\infty$ for A. On the other hand, if A chooses 2 as radius, her payoff is -2^{α} which is greater than $-\infty$. As a coalition can consist of only A, this mechanism is not $(1, 1)$-resistant. □

Theorem 15. *The given mechanism of strong connectivity game is not* 1*-immune.*

Proof. Let D be an arbitrarily acting player. Let D choose 1 as radius. Then, A cannot connect to B or C by choosing radius 1, which results with a payoff $-\infty$ for A. Thus, this mechanism is not t-immune for any $t \geq 1$. □

Theorem 16. *The given mechanism of strong connectivity game is not* 1*-stable.*

Proof. For each of the players A and D, there exists no weakly dominant strategy. □

Chapter 7

DELEGATE: COALITION, SYBIL, AND COPY PROOF INCENTIVIZED OUTSOURCED COMPUTATION WITH SMART CONTRACTS

7.1 Introduction

A client can improve its computation capability and memory capacity by outsourcing computation to benefit from cloud providers. A project can be outsourced as a whole, yet there also exist distributed projects where large computations are broken into smaller pieces for outsourcing to volunteer nodes. An early example of such a distributed computation mechanism is the SETI@home project[1] that aims at finding intelligence in outer space.

In both cases, to securely and dependably outsource a heavy computation to third parties, there has been a long line of approaches ranging from the ones based on cryptography [Gennaro et al., 2010, Parno et al., 2012, Thaler et al., 2012, Goldwasser et al., 2015, Sadeghi et al., 2010] to those based on game theory [Belenkiy et al., 2008, Küpçü, 2017, Dong et al., 2017, Jiang and Tian, 2019, Avizheh et al., 2019, Dong et al., 2020, Küpçü and Safavi-Naini, 2021]. Among the latter, game-theory based *incentivized outsourced computation*, where the same computational task is outsourced to more than one contractor, is an interesting arising subject studied in the two-party [Belenkiy et al., 2008, Avizheh et al., 2019, Küpçü and Safavi-Naini, 2021] and multi-party [Küpçü, 2017, Biçer et al., 2021] settings. In this approach, a boss plans to outsource a computation on an input (while neither the computation nor the input being necessarily private), and is interested in receiving the correct computation result. If the boss

[1] `https://setiathome.ssl.berkeley.edu`

runs the computation from scratch to verify the correctness of the computation, outsourcing it would not result in targeted efficiency improvement. Although specifically for NP problems there exist polynomial-time verification algorithms, we are interested in practical efficiency, which may not be achieved by high degree polynomial verification.

Regarding the multiple contractor setting, the probability of all the contractors being maliciously driven is lowered. The previous work [Küpçü, 2017] and its extension [Biçer et al., 2021] provide a mechanism where the honest contractors help the outsourcer catch and punish lazy contractors. The former achieves all contractors diligently executing the computation as the Nash equilibrium, while the latter fine-tunes the mechanism for ensuring security against a coalition with up to all involved contractors except one. However, none of these works analyze the full collaboration of contractors for sharing the computation load.

A recent approach in incentivized outsourced computation has been using *smart contracts* [Dong et al., 2017, Jiang and Tian, 2019, Avizheh et al., 2019, Dong et al., 2020, Küpçü and Safavi-Naini, 2021]. Each party deposits some amount to the smart contract at the beginning of the protocol execution. The smart contract collects the results on behalf of the boss and returns to each party the corresponding outcome of the protocol, e.g., the computation result for the boss and a reward or a fine to each contractor. This way, the use of smart contracts allows a higher efficiency for the resource-restrained boss and protects a diligent contractor from the boss's potential malicious actions, i.e., not paying the reward. However, none of these protocols are in the multi-contractor setting or provide any security guarantee against Sybils or coalitions.

Avizheh et al. proposed the *copy attack* in the smart contract setting [Avizheh et al., 2019]. As the contractors need to send the computation result publicly to the smart contract for verification, a copy attacker can copy a response previously submitted instead of executing the computation herself. Although [Küpçü and Safavi-Naini, 2021] provides a "Match Check" protocol to resolve this problem, there still exists a need for analyzing complicated outcomes that combine

the copy attack with various other strategies and resolving it in the multi-party setting.

Another and maybe more general issue is the lack of well-developed models for bridging game theory and cryptography. For example, we detect a tricky issue in previous works [Belenkiy et al., 2008, Küpçü, 2017, Küpçü and Safavi-Naini, 2021, Biçer et al., 2021], due to the involvement of cryptographic protocols in outsourcing an unbounded computation. The analyses of these works show that they do not bound contractors in the execution of outsourced computations, but bound them when it comes to breaking the security of the involved cryptographic primitive (Merkle hash tree).

7.1.1 Overview of Our Techniques

In this work, we target to resolve the above-mentioned issues with incentivized outsourced computation. The setting that we consider is a combination of those of [Küpçü, 2017] and [Küpçü and Safavi-Naini, 2021] such that a boss outsources the computation to $n > 1$ contractors and delegates the work of checking the final result to a smart contract. We model each contractor as a type of adversary that we name as *bounded rational* adversary, which is rational by having the main interest to maximize its utility and is bounded to execute only a PPT ITM. This can be considered as inspired by the time-and-budget-limited adversary model of [Premnath and Haas, 2015].

At the start of the execution, the boss deposits to the smart contract (SC) a sufficient amount of currency for rewarding all the contractors. Each contractor deposits to the SC a sufficient amount of currency for the fine. Then, the contractors are given a deadline τ_{SC} for executing the computation and submitting their commitments to the computation results and the Merkle hash tree root of the computation steps. After τ_{SC}, they have another deadline τ_F to submit the openings of their commitments. We create a response submission protocol that is universally composable (UC) [Canetti, 2001] based on non-interactive reusable-CRS (common reference string) commitments. Our response submission protocol

can potentially be used beyond our outsourcing mechanism.

At τ_F, the SC gets activated and checks whether there exists a consensus in the submitted results from the contractors. If this is the case, then SC returns the result to the boss, gives back the deposit of fines to the contractors, and distributes the rewards to the contractors. Otherwise, it runs the MismatchCheck protocol of [Küpçü, 2017] to determine the final result. Each MismatchCheck execution takes place among two contractors and SC, and assigns one of the contractors as the winner and the other one as the loser (never both are assigned as the winner or the loser). It is designed in a way that it will always assign a diligent contractor as the winner. The algorithm of SC ensures that there is an incentive for a diligent contractor to participate in this protocol.

To involve the used cryptographic protocols in the game theoretical analysis, regarding the Merkle hash tree, the bounded rational adversary may take any malicious action with negligible probability. Let the utility from this attack be a polynomial $poly_a(\lambda)$ of the security parameter λ if this attack is successful. Then the expected utility due to this attack action is a negligible utility computed as $poly_a(\lambda) \cdot \epsilon$, where ϵ is the negligible probability of success.

Regarding the involvement of the UC-secure response submission protocol, we consider that each participant in the protocol is controlled by a player and the possible messages by the environment to that participant are essentially actions of each player. Although we do not assume the existence of the ideal functionality, any distinguishable outcome results in a breach for UC-security, therefore can only succeed with negligible probability. As in the case with the Merkle hash tree construction, the bounded rational adversary can achieve only an additional negligible utility.

We highlight that the equilibrium and coalition-proofness concepts in our game theoretical analysis are computational versions of the generic definitions. We derive them by allowing a deviation to receive a higher utility only by a negligible function, which is then compatible with the bound rational adversaries.

7.1.2 Our Contributions

- In Section 7.4, we provide a UC-secure [Canetti, 2001] response submission protocol for smart-contract-assisted multi-party incentivized outsourced computation based on UC commitments [Canetti and Fischlin, 2001, Damgård and Groth, 2003, Fischlin et al., 2011, Fujisaki, 2014, Fujisaki, 2016]. Our protocol prevents the copy attack [Avizheh et al., 2019], and eliminates the need for further analysis of strategies involving this attack, as long as the submitter of a response does not choose to leak it on purpose, which can be considered as forming a coalition. Being UC-secure, our proposal has strong security in scenarios where multiple protocols run concurrently, which is an immediate concern in a blockchain-based distributed system.

- In Section 7.5, we propose our smart-contract-assisted n-party incentivized outsourced computation mechanism: `Delegate`. Compared to previous n-party protocols [Küpçü, 2017, Biçer et al., 2021], our proposed protocol incentivizes each contractor not only to execute the computation diligently but also to submit the correct computation output as soon as he can and to help catch the cheating contractors. Further, our proposed n-party mechanism achieves a low fine-to-reward ratio f/r, which, according to [Belenkiy et al., 2008], should be kept low for incentivization of contractors to join the system. This becomes a particularly important issue in a smart-contract-managed system, as each contractor is expected to deposit a fine at the beginning of the execution. More concretely, we achieve a fine-to-reward ratio ($\frac{f}{r} \approx \frac{1}{n-1}$) much lower than that of [Küpçü, 2017, Biçer et al., 2021] ($1 \leq \frac{f}{r} \leq n-1$).

- In Section 7.6, we analyze our mechanism in the setting where contractors are allowed to form coalitions and to collaborate for sharing the computation load and the payment of fines and rewards. We show that in this

setting, our mechanism is $n-1$-repellent according to the ℓ-repellence definition of [Biçer et al., 2021]. This shows that as long as all contractors in the system do not establish a coalition (i.e., there exists a separate party outside the coalition), the boss is guaranteed to receive the correct output since rational contractors will compute the task diligently.

7.2 Related Work

Outsourced Computation.

Previous works [Golle and Mironov, 2001, Szajda et al., 2003, Du et al., 2010] utilize "inner state hashes" to verify the computation result cheaply for the outsourcer. In fact, the technique utilized in this chapter for verification is also an extension of the ones [Belenkiy et al., 2008, Küpçü, 2017, Küpçü and Safavi-Naini, 2021, Biçer et al., 2021], by letting the smart contract to handle the verification task, as in [Küpçü and Safavi-Naini, 2021].

The previous incentivized outsourced computation works that have the closest settings and goals to ours can be listed as [Belenkiy et al., 2008, Küpçü, 2017, Dong et al., 2017, Jiang and Tian, 2019, Dong et al., 2020, Küpçü and Safavi-Naini, 2021]. Our setting essentially differs from them as follows:

- [Jiang and Tian, 2019, Dong et al., 2020] are in a single-contractor setting, while [Belenkiy et al., 2008, Dong et al., 2017, Küpçü and Safavi-Naini, 2021] are in a two-contractor setting. Our work essentially differs from these works by outsourcing to multiple contractors.

- [Küpçü, 2017, Biçer et al., 2021] are in a multi-contractor setting as our work. They also outsource the same computation to multiple contractors and incentivize them to submit the correct responses. [Biçer et al., 2021] further achieves security against coalitions. Yet, our proposal further incentivizes submitting the correct and early computation output and helping catch cheaters. Also, note that they do not use a smart contract and let the boss do the verification of responses.

7.3 Preliminaries and Subprotocols

Negligible Function. A function $\mathsf{n}(\cdot) : \mathbb{N} \to \mathbb{R}$ is *negligible*, if for each positive polynomial $poly(\cdot)$ there exists an integer $N > 0$ such that for all $\lambda > N$ we have $|\mathsf{n}(\lambda)| < \frac{1}{poly(\lambda)}$. If a negligible function takes as input the security parameter to output a probability or a utility, we may call it negligible probability or negligible utility, respectively.

Smart Contract. A smart contract (SC) is utilized for establishing binding contracts in cryptocurrencies (e.g., Ethereum [Buterin, 2013]). SC is publicly executed by the underlying consensus protocol. It collects the submitted transactions to the network and appends them to the blockchain. We assume SC as a trusted party just as in other smart contract based protocols [Kothapalli et al., 2017, Afanasev et al., 2018, Avizheh et al., 2019, Küpçü and Safavi-Naini, 2021].

7.3.1 Outsourced Computation Primitives

We briefly describe useful primitives in incentivized outsourced computation as provided in [Canetti et al., 2011, Küpçü and Safavi-Naini, 2021].

Merkle Hash Tree. A Merkle hash tree is a binary tree generated by hashing a set of data elements $E = (e_1, \ldots, e_s)$. Each leaf with index i of the tree is constructed as $H(e_i)$ where $H : \{0,1\}^* \to R$ is a collision resistant hash function, and R is a set whose each element can be encoded as a bit string and whose number of elements is $O(poly(\lambda))$. Each other node of the tree is constructed by $H(i,j)$ where i and j are the children of that node. The root hash is denoted by $z = MHRoot(E)$. We denote the Merkle proof of element e_i as $p_i = MHproof(E, e_i)$. This proof is composed of the hash values of the sibling nodes along the path between $H(e_i)$ and the root, and is used for proving the inclusion of an element in z with one minus negligible probability. $VerifyMHProof(z, i, e_i, p_i)$ is the verification algorithm that runs on input the root hash z, an index i, and the indexed element e_i and its proof p_i. This algorithm can be computed in complexity $O(\log s)$ to obtain an output *Success* or *Failure* depending on the success of the verifica-

tion.

Computation trace. The computation is represented by a Turing machine $TM()$ with an input tape initially storing the input string. The reduced configuration rc_j of the computation state at the step j is assigned as $\big(s_j, h_j, v_j, MHRoot(t_j)\big)$, where s_j , h_j, v_j, and t_j are the state, the head position, the tape at the given head, and the tape of the related TM configuration [Canetti et al., 2011]. We let the set $RC = (rc_1, \ldots, rc_s)$ and the hash root of the computation $z = MHRoot(RC)$ as in [Küpçü and Safavi-Naini, 2021].

MismatchCheck **protocol.** Proposed for use in two-party incentivized outsourced computation [Küpçü and Safavi-Naini, 2021], this protocol is run among a checker (the smart contract in our setting) and two contractors C_i and C_j who are assigned to execute the same computation TM on some input x. The aim is to catch the cheater (set to lose) if only one of C_i and C_j is diligent (set to win). We note that this protocol always makes at least one contractor lose if their inputs are different (i.e., mismatching). Before the protocol is run, the checker is supposed to receive the responses (y_i, z_i) and (y_j, z_j) from the contractors. The execution flow of *MismatchCheck* is given in Protocol 7.1. The protocol takes $O(\log s)$ time where s is the number of computation steps. In our proposal, we use multiple executions of this protocol to obtain the final result in case of the absence of consensus in the submitted responses by the contractors. Note that the checker should already have the first reduced configuration rc_1 in the protocol start, although it is not explicitly mentioned in [Küpçü and Safavi-Naini, 2021]. Otherwise, the protocol may not set a contractor with the correct submission to win, if the contractors differ even in the first computation step.

7.3.2 The Universal Composability (UC) Framework

We now provide a brief introduction to the Universal Composability (UC) framework [Canetti, 2001]. To define a protocol for a given task, first, we need to formalize an ideal world process for carrying out the task. In the ideal world, there exists no communication among the parties. Instead, they have access to an ideal

MismatchCheck

The checker receives the computation steps s_i and s_j from C_i and C_j.

The checker runs $CommittedBinarySearch(s, z_i, z_j)$ (the procedure is given below) with C_i and C_j to find the smallest k such that $rc_{k,i} = rc_{k,j}$ and $rc_{k+1,i} = rc_{k+1,j}$ where $rc_{k,i}$ denotes rc_k of C_i's computation and $s = min(s_1, s_2)$.

The checker sends k to C_i and C_j. For $a = 1, 2$, C_a sends to the checker $(rc_{k,a}, MHProof(RC_a, rc_{k,a}), rc_{k+1,a}, MHProof(RC_a, rc_{k+1,a}), p_{k,a})$.

The checker runs $VerifyCommittedStep$ (the procedure is given below) with C_b for $b = i, j$. If any contractor is not set to lose, he is set to win.

$CommittedBinarySearch(s, z_i, z_j)$

Set $low \leftarrow 0$, $high \leftarrow s$
repeat
  $Middle \leftarrow \lfloor (high - low)/2 \rfloor + low$
  The checker sends *middle* to C_i and C_j.
  For $a = 1, 2$, C_a sends to the checker $rc_{middle,a}, MHproof(RC_a, rc_{middle,a})$.
  if $VerifyMHProof(z_a, middle, rc_{middle,a}, MHproof(RC_a, rc_{middle,a})) = Failure$
   Set C_a to lose, Exit
  if $(rc_{middle,i} = rc_{middle,j})$ **then** $low \leftarrow middle$
  else $high \leftarrow middle$
until $high = low + 1$
return low

$VerifyCommittedStep(rc_a, MHProof(RC, rc_a), rc_{a+1}, MHProof(RC, rc_{a+1}), p_a)$

if $VerifyMHProof(z_b, a, rc_a, MHProof(RC, rc_a)) = failure$ **or** $VerifyMHProof(z_b, a + 1, rc_{a+1}, MHProof(RC, rc_{a+1})) = failure$
  Set C_b to lose, Exit
The checker parses $(s_a, h_a, v_a, MHRoot(t_a)) \leftarrow rc_a$,
$MHProof(t_a, h_a) \leftarrow p_a$.
if $VerifyMHProof(MHRoot(t_j), h_a, v_a, MHProof(t_a, h_a)) = failure$
  Set C_b to lose, Exit
The checker generates $(s'_{a+1}, h'_{a+1}, v'_{a+1})$ by simulating Turing Machine on (s_a, h_a, v_a).
The checker parses $(s_{a+1}, h_{a+1}, v_{a+1}, MHRoot(t_{a+1})) \leftarrow rc_{a+1}$.
if $(s'_{a+1}, h'_{a+1}, v'_{a+1}) \neq (s_{a+1}, h_{a+1}, v_{a+1})$
  Set C_b to lose, Exit

Protocol 7.1: *MismatchCheck* protocol given by [Küpçü and Safavi-Naini, 2021].

functionality, which is essentially a trusted party that captures the desired requirements of the task. A real life protocol securely realizes a task if running the protocol "emulates" the defined ideal process.

We model each protocol participant, ideal functionality, and adversarial party as an interactive Turing machine (ITM) [Goldwasser et al., 1985]. The input and output tapes are utilized for communicating with other programs running on the same machine, and the communication tapes are utilized for communicating with other ITMs on the network. Moreover, we restrict each ITM to run in probabilistic polynomial time (PPT).

Protocol execution in the real world. We briefly describe the execution of a given protocol π (run by parties $P_1, \ldots, P_n$) in the presence of a PPT adversary $\mathcal{A}$ and a PPT environment $\mathcal{Z}$. The execution flows by a sequence of activations. In each activation, a single ITM is activated for reading its input and incoming communication tapes, executing its code, and possibly writing to its output and outgoing communication tapes. Further, $\mathcal{Z}$ can write on the input tapes of the parties and can read their output tapes. $\mathcal{A}$ carries out the communication between protocol participants by reading messages from the outgoing message tapes of the sender, and by copying them to the incoming message tapes of the recipient. We highlight that $\mathcal{A}$ cannot modify or duplicate these messages. If $\mathcal{A}$ corrupts a party, it learns the internal information of that party and fully controls that party afterward.

When the execution starts, $\mathcal{Z}$ is activated first. Upon activation, it may write information on the input tape of only one other party or only the adversary. After the activation of $\mathcal{Z}$ is complete, the next ITM is activated. Once a party completes its activation, $\mathcal{Z}$ is activated again. Whenever $\mathcal{A}$ delivers a message to some party P_i in some activation, then P_i is the one activated next. Once P_i's activation is complete, $\mathcal{Z}$ is activated again. If in some activation $\mathcal{A}$ does not deliver any messages, then $\mathcal{Z}$ is activated after $\mathcal{A}$ finishes its activation. We note that via this mechanism, $\mathcal{Z}$ and $\mathcal{A}$ can freely exchange information, between any two activations of some party. The output of the protocol execution is the output of $\mathcal{Z}$.

The ideal process. The ideal process involves an ideal functionality $\mathcal{F}$, an ideal adversary $\mathcal{H}$, an environment $\mathcal{Z}$ and a set of dummy parties $\tilde{P}_1, \ldots, \tilde{P}_n$. Whenever $\tilde{P}_i$ is activated with an input x, it forwards x to $\mathcal{F}$. Whenever $\tilde{P}_i$ is activated with incoming message y from $\mathcal{F}$, it copies y to its output. The ideal adversary $\mathcal{H}$ is responsible for delivering messages from $\mathcal{F}$ to the dummy parties. It can also corrupt dummy parties in order to control them and to learn what they know. The order of events in the ideal process differs from the real world execution by not being involved in the messages from a dummy party to $\mathcal{F}$. We highlight that in the ideal process, no communication takes place among the dummy parties. The output is the output of $\mathcal{Z}$.

Securely realizing an ideal functionality. We say that a protocol π securely realizes an ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$, there exists an ideal adversary $\mathcal{H}$ such that no environment $\mathcal{Z}$, can distinguish between a real world protocol execution and ideal process except for some negligible probability. Formally,

Definition 20 (Computational Indistinguishability)**.** *We say that two distributions* $\{X(\lambda, a)\}_{\lambda \in N, a \in \{0,1\}^*}$ *and* $\{Y(\lambda, a)\}_{\lambda \in N, a \in \{0,1\}^*}$ *are computationally indistinguishable (denoted by* $\{X(\lambda, a)\}_{\lambda \in N, a \in \{0,1\}^*} \approx_c \{Y(\lambda, a)\}_{\lambda \in N, a \in \{0,1\}^*}$*), if there exists a negligible function* $\mathsf{n}(\cdot)$ *such that* $|Pr(X(\lambda, a) = 1) - Pr(Y(\lambda, a) = 1)| < \mathsf{n}(\lambda)$.

Definition 21 (Realizing a Functionality)**.** *Let* $\mathcal{F}$ *be an ideal functionality,* π *be an n-party real world protocol,* $REAL_{\pi,\mathcal{A},\mathcal{Z}}$ *be the output of* $\mathcal{Z}$ *in the real world protocol execution, and* $IDEAL_{\mathcal{F},\mathcal{H},\mathcal{Z}}$ *be the output of* $\mathcal{Z}$ *in an ideal process. We say that* π *securely realizes* $\mathcal{F}$ *if for any adversary* $\mathcal{A}$ *there exists an ideal adversary* $\mathcal{H}$ *such that for any environment* $\mathcal{Z}$*, we have* $REAL_{\pi,\mathcal{A},\mathcal{Z}} \approx_c IDEAL_{\mathcal{F},\mathcal{H},\mathcal{Z}}$.

Composition. Thanks to the composition theorem [Canetti, 2001], one can utilize a functionality $\mathcal{F}$ in the design of another protocol π'. The parties may communicate with an unbounded number of copies of $\mathcal{F}$, each of which is identified via a unique session identifier *sid*. Each copy of $\mathcal{F}$ mimics the ideal process. Although the adversary is responsible for delivering the messages from the copies

of $\mathcal{F}$ to the parties, it cannot access their contents. Also, the environment cannot have direct access to the copies of $\mathcal{F}$. If one proves that π' securely realizes a functionality $\mathcal{F}'$ in the hybrid model of computation with access to $\mathcal{F}$ (i.e., $\mathcal{F}$-hybrid model), then any protocol π that has been shown to realize $\mathcal{F}$ can replace $\mathcal{F}$ as follows. When the first message is sent to a copy *sid* of $\mathcal{F}$, that message gets inputted to an invocation of a new copy of π. All subsequent messages to *sid* are replaced with an activation of the corresponding copy of π, with the same message content. All outputs from a copy of *pi* are treated as a message received from *sid*.

7.3.3 *Non-Interactive and Reusable-*CRS *Commitment Schemes*

Commitment schemes (Commit, Open) are useful cryptographic primitives that enable Alice to commit to a value (usually, a bit or bit string) for Bob via Commit operation, and later open it via Open operation. The two traditional requirements for commitment schemes are being hiding (i.e., Commit does not reveal the committed value without Open) and binding (i.e., once Commit takes place, Alice cannot execute Open for different values). A commitment scheme is non-interactive if both Commit and Open costs one round, i.e., for each of them, one string is sent to Bob. The way that we utilize the commitment scheme in Section 7.4 requires commitments from multiple parties, and possibly many similar protocols may run concurrently. We stress that commitment schemes in the standard model are not secure by default in this setting as other users' commitments and their openings can be utilized for copying or generating different commitments. Instead, we expect the commitment scheme to satisfy the ideal functionality given in Protocol 7.2, i.e., the string commitment version of the multiple bit commitment ideal functionality given in [Canetti and Fischlin, 2001]. Unlike the ones in standard model, $\mathcal{F}_{\text{MCOM}}$ ensures each commitment is generated as a completely blinded envelop and is bound to its generator.

UC-secure commitments are shown to be non-existent without a common reference string CRS (the ideal functionality is given in Protocol 7.3) by [Canetti and

Functionality $\mathcal{F}_{\text{MCOM}}$

Running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$, $\mathcal{F}_{\text{MCOM}}$ proceeds as follows.

1. On receiving a message $(\mathsf{Commit}, sid, cid, P_i, P_j, s)$ from P_i, record the tuple (cid, P_i, P_j, s) and send the message $(Receipt, sid, cid, P_i, P_j)$ to P_j and $\mathcal{S}$, where sid is the session id, cid is the commitment id, and $s \in \{0,1\}^*$ is the committed string. Ignore subsequent $(\mathsf{Commit}, sid, cid, \cdot)$ messages.

2. On receiving a message $(\mathsf{Open}, sid, cid, P_i, P_j)$ from P_i, proceed as follows: If there exists a record for the tuple (cid, P_i, P_j, s), then send the message $(Open, sid, cid, P_i, P_j, s)$ to P_j and $\mathcal{S}$. Otherwise, do nothing.

Protocol 7.2: The ideal functionality for multiple UC commitments.

Functionality $\mathcal{F}_{\mathsf{CRS}}$

Parameterized by a distribution D, $\mathcal{F}_{\mathsf{CRS}}$ proceeds as follows.

1. When activated for the first time on input $(value, sid)$, choose a value $d \leftarrow_R D$ and send d back to the activating party. In each other activation, return d to the activating party.

Protocol 7.3: The ideal functionality for the common reference string.

Fischlin, 2001]. Therefore the ideal functionality given in Protocol 7.2 is the one with reusable-CRS for efficiency. Any non-interactive commitment scheme realizing $\mathcal{F}_{\text{MCOM}}$, e.g., [Canetti and Fischlin, 2001, Damgård and Groth, 2003, Fischlin et al., 2011, Fujisaki, 2014, Fujisaki, 2016], can be used for realizing our protocol. Note that concurrent commitment schemes [Pass and Rosen, 2005, Pass and Wee, 2010, Lin and Pass, 2011, Goyal et al., 2016, Khurana, 2017, Lin et al., 2017, Khurana and Sahai, 2017, Bitansky and Lin, 2018] are also shown to UC-secure by [Lin et al., 2009], therefore, the non-interactive ones [Khurana and Sahai, 2017, Lin et al., 2017, Bitansky and Lin, 2018] are also eligible. However, stand-alone non-malleable commitments [Dolev et al., 1991, Di Crescenzo et al., 1998] should be

shown to realize $\mathcal{F}_{\text{MCOM}}$ before their use.

7.4 Our Response Submission Protocol

The copy attack [Avizheh et al., 2019] induces a special obstacle for smart contract based outsourced computation, as the output of the computation needs to be published on SC earlier than the deadline. In this section, we describe our proposed UC-secure response submission protocol against the copy attack. The protocol that we propose utilizes the ideal functionality $\mathcal{F}_{\text{MCOM}}$ for reusable-CRS commitments.

Threat Model. Inline with the smart contract based incentivized outsourced computation protocol requirements, in our model, there is no guaranteed delivery of messages. The network is asynchronous, i.e., under adversarial control. Further, the communication can be eavesdropped by the adversary but cannot be modified (i.e., they are authenticated). In addition, each party has a unique identity. The adversary can actively and adaptively corrupt any number of parties. Although in `Delegate` protocol definition we model the SC as a trusted party, in this section we model it as a protocol participant that can be fully corrupted by the adversary. As each message received by SC is public, we cannot consider it a trusted party in this sense. We stress that instead of modeling SC with precise and detailed adversarial risks, we achieve a stronger security guarantee by allowing its full corruption. We also note that this is more flexible as the response submission protocol can be useful in further settings than the one in `Delegate`, where the adversary can eclipse parties to mimic SC for them.

The ideal functionality $\mathcal{F}_{\text{RS}}$. Protocol 7.4 provides the ideal functionality $\mathcal{F}_{\text{RS}}$ for the response submissions of the contractors. Response messages from the contractors to $\mathcal{F}_{\text{RS}}$ need to be sent before the time τ_{SC}. A contractor may send Leak messages to $\mathcal{F}_{\text{RS}}$, to model the action of leaking the response to another contractor. Yet, apart from the contractor's own action, the response does not get leaked to any other contractor or SC. Also, we show that leaking is disincentivized in Section 7.6. Once the response is submitted, it cannot be altered or re-submitted.

Protocol $\approx_{\mathbf{RS}}$. Protocol 7.5 provides our proposed protocol $\approx_{\text{RS}}$ to realize $\mathcal{F}_{\text{RS}}$ in the $\mathcal{F}_{\text{MCOM}}$-hybrid model. $\approx_{\text{RS}}$ runs with a single copy of $\mathcal{F}_{\text{MCOM}}$. Until the current time $\tau_C = \tau_F$, SC just records the obtained messages and then gets activated to run the algorithm for the output. We highlight that in the implementation of this protocol the value CRS will be required for the realization of the functionality $\mathcal{F}_{\text{MCOM}}$, which needs to be submitted to SC by the boss before the start of the protocol execution as part of the auxiliary information aux.

Functionality $\mathcal{F}_{\text{RS}}$

Running with contractors $P_1, \ldots, P_n$, the smart contract SC, and an adversary $\mathcal{S}$, $\mathcal{F}_{\text{RS}}$ proceeds as follows.

1. When the current time $\tau_C < \tau_{SC}$, on receiving a message $(\mathsf{Response}, sid, P_i, (y, z))$ from P_i for the first time, record the tuple $(P_i, sid, (y, z))$. Send the message $(Receipt, sid, P_i)$ to $\mathcal{S}$. Ignore other $(\mathsf{Response}, sid, P_i, \cdot)$ tuples. On receiving a message $(\mathsf{Leak}, sid, P_i)$ from P_i, if there exists a record $(P_i, sid, (y, z))$, send $(Leak, sid, P_i, (y, z))$ to $\mathcal{S}$. Otherwise do nothing.

2. When $\tau_{SC} = \tau_C$, send each recorded $(P_i, sid, (y, z))$ tuple to $\mathcal{S}$.

3. When $\tau_C = \tau_F$, send each recorded $(P_i, sid, (y, z))$ tuple to SC.

Protocol 7.4: The ideal functionality for response submission of contractors.

Theorem 17. *The protocol π_{RS} given in Protocol 7.5 securely emulates the ideal functionality $\mathcal{F}_{RS}$ given in Protocol 7.4 in $\mathcal{F}_{MCOM}$-hybrid model.*

Proof. We construct an ideal world PPT adversary $\mathcal{H}$ that interacts with the environment $\mathcal{Z}$, the dummy parties $P_0, \ldots, P_n$, and SC, and the ideal functionality $\mathcal{F}_{\text{RS}}$ in an ideal execution. $\mathcal{H}$ is divided into two internal parts, a copy of the real world PPT adversary $\mathcal{A}$ in a black-box fashion and a PPT simulator $\mathcal{S}$. Throughout the ideal execution, $\mathcal{S}$ simulates a real execution of $\approx_{\text{RS}}$ for $\mathcal{A}$ by acting as an interface between $\mathcal{A}$ and the outside of $\mathcal{H}$. $\mathcal{S}$ is allowed to modify the messages

The Response Submission Protocol π_{RS}

Running with contractors $C_1, \ldots, C_n$, the smart contract SC, and an adversary $\mathcal{H}$, π_{RS} proceeds as follows in the presence of one copy of $\mathcal{F}_{\text{MCOM}}$ functionality with session id sid_{MCOM} running with parties $P_1, \ldots, P_n$, and SC.

1. When the current time $\tau_{\text{C}} < \tau_{SC}$, an honest P_i and an honest SC proceed as follows. Upon the first input $(\mathsf{Response}, sid, P_i, (y, z))$, P_i picks a unique cid and sends the message $(\mathsf{Commit}, sid_{\text{MCOM}}, cid, P_i, \text{SC}, (y, z))$ to $\mathcal{F}_{\text{MCOM}}$. Then, $\mathcal{F}_{\text{MCOM}}$ sends $(\mathsf{Commit}, sid_{\text{MCOM}}, cid, P_i, \text{SC})$ to SC. P_i ignores other $(\mathsf{Response}, *)$ inputs. Upon the input $(\mathsf{Leak}, sid, P_i)$, P_i sends the message $(\mathsf{Open}, sid_{\text{MCOM}}, cid, P_i, \text{SC})$ to $\mathcal{F}_{\text{MCOM}}$, which in turn sends $(Open, sid_{\text{MCOM}}, cid, P_i, \text{SC}, (y, z))$ to SC. SC just keeps appending all the messages coming from $\mathcal{F}_{\text{MCOM}}$ to its database.

2. When $\tau_{SC} \leq \tau_{\text{C}} < \tau_{\text{F}}$, an honest P_i and SC proceed as follows. P_i sends the message $(\mathsf{Open}, sid_{\text{MCOM}}, cid, P_i, \text{SC})$ to $\mathcal{F}_{\text{MCOM}}$. If there is a related record, $\mathcal{F}_{\text{MCOM}}$ sends $(\mathsf{Open}, sid_{\text{MCOM}}, cid, P_i, \text{SC}, (y, z))$ to SC. Again, SC just passively keeps appending all the messages coming from $\mathcal{F}_{\text{MCOM}}$ to its database.

3. When $\tau_{\text{C}} = \tau_{\text{F}}$, on its database, SC deduces each P_i's first $(\mathsf{Commit}, sid_{\text{MCOM}}, cid, P_i, \text{SC})$ message among the ones sent before τ_{SC} on its database. These messages are combined to obtain a table T with up to n entries. Then, SC generates another table T' with entries $(P_i, sid, (y, z))$ obtained from the corresponding $(\mathsf{Open}, sid_{\text{MCOM}}, cid, P_i, \text{SC}, (y, z))$ entries sent in the interval $(\tau_{SC}, \tau_{\text{F}})$ for each $(P_i, sid_{\text{MCOM}}, cid)$ on T. For the ones that are found, SC outputs each entry of T'.

Protocol 7.5: The $\mathcal{F}_{\text{MCOM}}$-hybrid protocol for response submission of contractors.

between $\mathcal{A}$ and the outside world, with the exception that whenever $\mathcal{A}$ outputs a message to $\mathcal{Z}$, $\mathcal{S}$ just forwards this message to $\mathcal{Z}$. More precisely,

1. Throughout the whole execution $\mathcal{S}$ runs an internal copy $\mathcal{F}'_{\text{MCOM}}$ of $\mathcal{F}_{\text{MCOM}}$ with the session id sid_{MCOM}.

2. When the current time $\tau_{\text{C}} < \tau_{SC}$:

   (a) If at some point $\mathcal{S}$ receives $(Receipt, sid, P_i)$ from $\mathcal{F}_{\text{RS}}$, $\mathcal{S}$ picks a unique cid and gives the tuple $(Receipt, sid_{\text{MCOM}}, cid, P_i, \text{SC})$ to $\mathcal{A}$. $\mathcal{S}$ also adds (P_i, cid) to a list L.

   (b) If at some point $\mathcal{S}$ receives $(Leak, sid, P_i, (y, z))$ from $\mathcal{F}_{\text{RS}}$, $\mathcal{S}$ gives the tuple $(Open, sid_{\text{MCOM}}, cid, P_i, \text{SC}, (y, z))$ to $\mathcal{A}$.

   (c) $\mathcal{S}$ simulates a corrupted party P_i as follows. For any $(\mathsf{Commit}, sid_{\text{MCOM}}, cid, P_i, \text{SC}, (y, z))$ command, if $(P_i, \cdot) \notin L$, $\mathcal{S}$ sends $(\mathsf{Response}, sid, P_i, (y, z)$ to $\mathcal{F}_{\text{RS}}$. For any $(\mathsf{Open}, sid_{\text{MCOM}}, cid, P_i, \text{SC})$

command, if $(P_i, cid) \in L$, $\mathcal{S}$ sends $(\mathsf{Leak}, sid, P_i$ to $\mathcal{F}_{\text{RS}}$. For all other cases, $\mathcal{S}$ utilize $\mathcal{F}'_{\text{MCOM}}$ to simulate $\mathcal{A}$'s view.

(d) $\mathcal{S}$ simulates internal transcripts of a corrupted SC via a database generated by appending all messages of the form $(Receipt, \cdot, \cdot, \cdot, \text{SC})$ $(Open, \cdot, \cdot, \cdot, \text{SC}, \cdot)$ that $\mathcal{S}$ gives to $\mathcal{A}$. Internal transcripts of any other corrupted party P_i are simulated by using the input tape of the corresponding ideal dummy party (i.e., for the messages to $\mathcal{F}_{\text{RS}}$) and the *cid* values generated for that party by $\mathcal{S}$.

3. When $\tau_C = \tau_{SC}$, $\mathcal{S}$ receives each recorded $(P_i, sid, (y, z))$ tuple from $\mathcal{F}_{\text{RS}}$.

4. When $\tau_{SC} < \tau_C < \tau_F$:

   (a) $\mathcal{S}$ simulates an honest party P_i's $(\mathsf{Open}, sid_{\text{MCOM}}, cid, P_i, \text{SC})$ call, by giving to $\mathcal{A}$ the corresponding message, if it received a related record.

   (b) For all other queries to $\mathcal{F}_{\text{MCOM}}$ by corrupted parties, $\mathcal{S}$ keeps simulating $\mathcal{A}$'s view via its internal copy of $\mathcal{F}_{\text{MCOM}}$.

   (c) Again, internal transcripts are simulated as above.

5. When $\tau_C = \tau_{SC}$, $\mathcal{S}$ simulates a corrupted SC by running the algorithms given in the 3-rd Step of Protocol 7.5. Note that an honest SC does not make any call for any outside functionality at this step.

The view of $\mathcal{A}$ in the above simulation is computationally indistinguishable from the one in a real world protocol execution. Also, the view of $\mathcal{Z}$ in the ideal world interacting with $\mathcal{H}$ is computationally indistinguishable from the one in the real protocol execution interacting with $\mathcal{A}$.

□

7.5 Our Incentivized Outsourced Computation Proposal

7.5.1 System Model

Essentially, the setting that we are interested in is a multi-party version of the two-party incentivized outsourced computation setting [Belenkiy et al., 2008, Küpçü and Safavi-Naini, 2021] and the inclusion of smart contracts and rational coalitions on top of [Küpçü, 2017], as we describe in what follows.

There exist three types of entities: (i) a *boss*, who is interested in outsourcing the computation of a deterministic polynomial time[2] Turing machine $TM()$ on an input tape x, (ii) a set N of n *contractors* that execute the computation to obtain the reward, and (iii) a *Smart Contract (SC)* that rules the system interactions between the parties on behalf of the boss, and handles the transactions. We can consider the boss as an honest designer of the mechanism rather than a player involved in it, as in [Belenkiy et al., 2008, Küpçü, 2017, Küpçü and Safavi-Naini, 2021]. She outsources the computation to the contractors and generates SC at the beginning of the game. Her goal is to obtain the correct computation result from the contractors at the end of the protocol, and she is willing to pay a total reward R for this goal.

We model a contractor as a rational party whose main interest is to maximize its utility. We bound each contractor to execute only a probabilistic polynomial time (PPT) interactive Turing machine (ITM). This can be considered as inspired by the time-and-budget-limited adversary model of [Premnath and Haas, 2015] but differs from it by limiting the adversary as being able to run a PPT algorithm instead of limiting it in terms of a budget.

Non-Collaborating and Collaborating Coalitions. The contractors may establish coalitions; thus the boss is naturally advised to pick the contractors in a manner that lowers the probability of coalitions. For simplicity, as usual [Belenkiy et al., 2008, Küpçü, 2017, Küpçü and Safavi-Naini, 2021], we assume the

[2]If it is a randomized algorithm, it should be converted to a deterministic one by providing the randomization script (e.g., pseudorandom seed) as part of the input.

contractors are picked randomly from a pool. We identify two types of possible coalitions among the contractors: (i) the *non-collaborative setting*, where the members of a coalition together decide the strategy that each of them individually plays, among the available strategies, and (ii) the *collaborative setting*, where the members of a coalition together decide the strategy that each of them plays, and can go beyond the individual strategies. In particular, they can simply decide that only one of them performs the computation (honestly or not), and then all of them submit the same result. We note that the previous work [Biçer et al., 2021] only analyzes the non-collaborative setting. In both settings, following the transferable utility assumption, we assume a single total utility for the coalition. This allows our setting to also cover Sybil contractors.

SC is an ITM that is trusted for running the program that it has been given, but its computation and storage are public, so it cannot protect any private values. The communication with SC is over authenticated public channels.

Goals of The Protocol. We briefly list our system goals, which are built upon the previous works [Belenkiy et al., 2008, Küpçü, 2017, Küpçü and Safavi-Naini, 2021, Biçer et al., 2021], as follows:

1. With high probability, the boss receives the correct computation result (considered by [Belenkiy et al., 2008, Küpçü, 2017, Küpçü and Safavi-Naini, 2021, Biçer et al., 2021]).

2. The contractors are not discouraged by setting a high fine to reward ratio (considered by [Belenkiy et al., 2008]).

3. There is an incentive for contractors to act honestly, i.e., an honestly acting contractor gets rewarded (considered by [Belenkiy et al., 2008, Küpçü, 2017, Küpçü and Safavi-Naini, 2021, Biçer et al., 2021]).

4. There is an incentive for contractors to submit the response early (not considered before).

5. Contractors playing the honest strategy are incentivized to help catch deviating contractors.

6. The mechanism is safe against collaborating contractors, i.e., the workload of computation can be shared within the coalition (only non-collaborating coalitions are considered by [Biçer et al., 2021]).

7. The workload to SC in case of mismatching submissions is kept low for the budget of the boss (considered by [Küpçü and Safavi-Naini, 2021]).

7.5.2 *Our `Delegate` Protocol*

To prepare for outsourcing, the boss specifies SC for the involved contractors, the computation $TM()$ to be executed, and the input x. SC collects deposits from the boss and the contractors. In accordance with the protocol $\approx_{\mathrm{RS}}$ (see Protocol 7.5), until a predetermined time τ_{SC}, SC passively collects computation results from the contractors. The computation results are submitted as Commit to (y_i, z_i) by each contractor P_i, where y_i is the computation result and z_i is the Merkle hash root of all computation steps (see Section 7.3). Then, until another predetermined time τ_F, the contractors send Open of their submitted responses to SC. After τ_F, SC checks all the submissions, determines the result of the computation, and delivers fines and rewards. It also returns the boss's residual deposit if there is any remaining. Finally, it sends the computation result to the boss. Protocol 7.6 shows the high-level flow of `Delegate`.

Notation. We partly use the notation of [Küpçü and Safavi-Naini, 2021]:

- $TM()$: The function to be computed, picked by the boss.

- x: The input to the function $TM()$, picked by the boss.

- aux: The auxiliary information submitted to SC at the beginning of the protocol by the boss.

- y: The computation result submitted by a contractor.
- z: The Merkle root of the computation steps, submitted by a contractor.
- r: The reward of a contractor, if her result is deemed correct by SC.
- f: The fine charged on a contractor, if her result is deemed incorrect by SC.
- τ_{SC}: The deadline given to the contractors for committing to their response (response includes both the computation result y and the Merkle root z). τ_{SC} should be large enough for the correct computation to be completed.
- τ_F: The deadline given to the contractors for opening their committed response.
- τ_C: The current time.

Algorithm 18 shows the algorithm of SC throughout the protocol. The algorithm activates at τ_F and collects fine deposits and responses submitted by the contractors. If no consensus in the submitted responses exists, SC executes the for loop in the mismatch resolution part. At each round of the for loop, the algorithm sets two of the submitted responses to compete by *MismatchCheck* executions (see Section 7.3.1). In each of these executions, submitters of the two responses are involved. When a submitter of a response loses, the algorithm continues by the next submitter of that response, until all submitters of a response are set to lose. Then that response gets eliminated. This way we make sure a submitted response can only get eliminated after all of its submitters are involved in *MismatchCheck* executions as a submitter of a response can be part of a hidden separate coalition and can lose *MismatchCheck* on purpose. After one response gets eliminated the for loop continues to the next round by picking another response to compete with the winner of the previous round.

Unlike the previous works [Küpçü, 2017, Küpçü and Safavi-Naini, 2021, Biçer et al., 2021], fine distribution to diligent contractors is optimized for the follow-

ing. The fine deposits by cheaters will be delivered to the diligent contractors that get actively involved in catching the cheaters in *MismatchCheck* executions. The number of executions that a diligent contractor gets involved in is less than or equal to the number of fine gains that he will receive. We consider this as a further incentivization to get involved and to stay online, as long as f is greater than the *MismatchCheck* execution cost for the diligent contractor. Further, the algorithm takes into account the response submission order in assigning *MismatchCheck* executions, as an incentivization of early submission by the diligent contractors. If a diligent contractor submits earliest, stays online, and joins each *MismatchCheck* asked by SC, he gets all the fines of the cheaters. In this case, the remaining diligent contractors only receive the reward and their deposited fine back. Yet, if the early submitter misses these duties, the remaining ones can receive more. An honest contractor algorithm is provided for completeness as Algorithm 19.

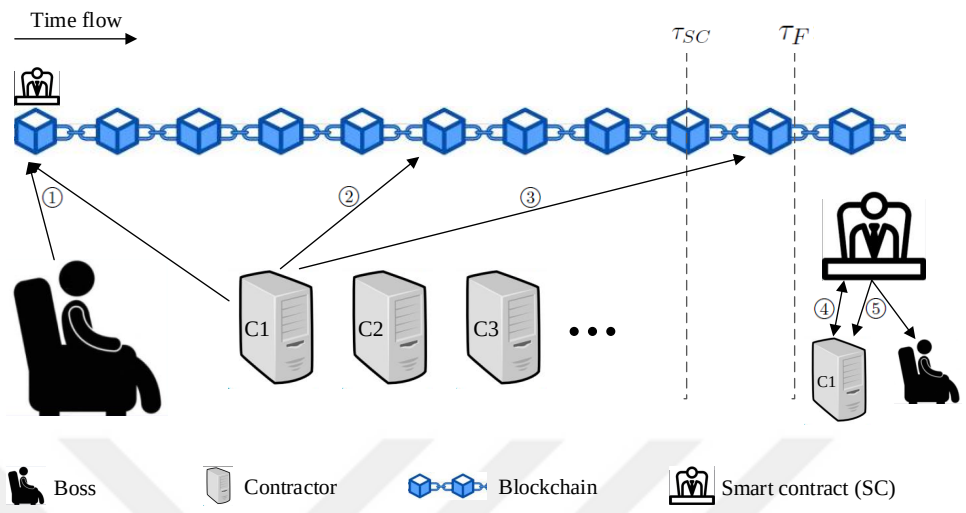
7.6 Game Theoretical Analysis of `Delegate`

In this section, we list the strategies that can be played throughout the `Delegate`, and analyze them from game theoretical perspective in absence of coalitions. We then extend our analysis to allow leaking and copying actions during submission and in presence of both collaborating and non-collaborating coalitions. We highlight that having a UC-secure response submission protocol means that the copy attack can succeed with negligible probability if the underlying commitment scheme is UC-secure.

7.6.1 Strategies of Contractors Outside Coalition

We define the strategy set in the setting without any coalition based on the strategies given in [Belenkiy et al., 2008, Küpçü, 2017, Küpçü and Safavi-Naini, 2021]. The available actions for each contractor are briefly as follows.

- Regarding computation of the task:



① The boss generates SC. She also submits $TM()$, x, aux, reward r, fine f, deadlines τ_{SC} and τ_F to SC. The boss also deposits the amount $n \cdot r$ as her commitment to paying the rewards. Each contractor deposits the amount f.

② Each contractor C_i submits Commit to his response (y_i, z_i) until the deadline τ_{SC} to SC.

③ Each contractor C_i submits Open of its committed response until the deadline τ_F to SC.

④ SC gets activated. SC collects the submitted responses by the contractors. SC verifies commitment openings. SC then checks whether there exists a consensus (y, z) in the submitted responses. If no consensus is reached, SC runs the *MismatchCheck* protocol with the contractors until obtaining a single result (y, z) (see Algorithm 18).

⑤ SC returns to the boss the computation result y. If there exists a consensus in the submitted responses, SC transfers the reward r to the balance of each contractor. It also returns each deposit f to its depositor. Otherwise, SC publishes the diligent contractors and the cheaters and transfers the reward r to the balance of each diligent contractor. It also returns the deposit f to each diligent contractor. Further, it distributes the collected fines from cheaters to the diligent contractors (see Algorithm 18 for details of the distribution) and sends the remaining reward deposit to the boss.

Protocol 7.6: Our `Delegate` protocol flow.

Algorithm 18 Complete algorithm of SC.

```
Activate at τ_F
Initial Check:
Check the sent blinded responses and their openings and prepare a table for each contractor with entries (P_i, (y_i, z_i))
where (y_i, z_i) is the first submitted commitment by P_i.
for each contractor P_i do
    if P_i did not submit a commitment or its opening then Cheaters.add(P_i)
    elseResponse(y_i, z_i).add(P_i)
    end if
end for
if ∀i, j (y_i, z_i) = (y_j, z_j) then
    ResultingResponse ← Response(y, z)
    Go to Delivery
end if
Mismatch resolution:    // Executed if there is a mismatch in the submitted responses
Initialize ResultingResponse as Response(y, z) with the greatest number of contractors
ResultingResponse[0].count ← Cheaters.size    // Start by assigning the cheaters' fines to this contractor.
for each Response(y, z) do
    i ← 0, j ← 0    // When competition of two responses starts, initialize indices
    Execute MismatchCheck with ResultingResponse[i] and Response(y, z)[j], during execution:
    if ResultingResponse[i] fails to comply or loses MismatchCheck then
        if i < ResultingResponse.size then
            i ← i + 1, restart MismatchCheck
        else
            Cheaters.add(ResultingResponse)
            Response(y, z)[j].count ← Cheaters.size
            ResultingResponse ← Response(y, z)
        end if
    else if Response(y, z)(j) fails to comply or loses MismatchCheck then
        if j < Response(y, z).size then
            j ← j + 1, restart MismatchCheck
            ResultingResponse[j].count+ = Response(y, z).size,
            Cheaters.add(Response(y, z))
        end if
    end if
end for
Delivery: Publish ResultingResponse, Cheaters, Quitters
Deliver (n − ResultingResponse.size) · r to the boss
Deliver r + f + j.count · f to each j ∈ ResultingResponse
```

Algorithm 19 Algorithm of Honest Contractor.

Activate at the protocol start: Deposit f to SC and learn $TM()$ and x. Compute $TM(x)$ diligently to obtain (y, z). Submit Commit to the response (y, z).

Activate at τ_{SC}: Submit Open of the commitment.

Activate at τ_F: Comply with each *MismatchCheck* execution requested by SC.

1. *Diligent computation.* This is the preferred action by the boss. The contractor runs the correct computation honestly and submits the result in the given time. The cost of the computation is denoted as $\mathsf{cost}(d)$.

2. *Lazy algorithm computation.* This action corresponds to choosing a lazy algorithm from the set $L = (l_1, \ldots, l_m)$ where $m \in O(poly(\lambda))$. In a lazy strategy with the algorithm l_i, the contractor runs l_i whose cost is defined as $\mathsf{cost}(l_i) < \mathsf{cost}(d)$. In presence of a single diligent contractor in the system, the probability of the lazy algorithm l_i producing a successful result is defined as $q_i < \mathsf{n}(\lambda)$ for some negligible function $\mathsf{n}(\cdot)$, thanks to the *MismatchCheck* protocols executed by SC. We highlight that this is a generalization of the previous "lazy q-algorithm" of [Küpçü, 2017, Küpçü and Safavi-Naini, 2021, Biçer et al., 2021] and "Guess" strategy by [Küpçü and Safavi-Naini, 2021]. We also highlight that, unlike the previous works, we do not limit the contractors to play a single lazy algorithm. Note that the algorithms with $\mathsf{cost}(l_i) \geq \mathsf{cost}(d)$ are simply dominated by the diligent algorithm and excluded from the consideration.

- Regarding response submission (based on actions in $\mathcal{F}_{RS}$):

  1. *Honest submission.* This corresponds to submitting the output of the computation algorithm (can be a lazy one).

  2. *Leaking.* This is a deviant submission action. The submitter of a response deliberately leaks the submitted response. We disincentivize leaking.

  3. *Copy strategies.* The copy attack [Avizheh et al., 2019] is that the contrac-

tor just waits for some response from the other contractors on the SC, then copies and submits that response, as if it is the contractor's own result. [Küpçü and Safavi-Naini, 2021] describes combined strategies including copy attack and diligent and lazy strategies. In this work, we proposed a UC-secure response submission protocol that allows any PPT contractor running to conduct this attack with at most a negligible probability, as long as the original submitter does not leak it.

- Regarding *Mismatch* protocol executions:

  1. *Not complying.* This includes trying to cheat during a *Mismatch* execution or failing to send messages in time.
  2. *Complying.* Honestly preparing and sending the messages in time per the protocol requirements.

7.6.2 *Analysis of Leaking and MismatchCheck Actions*

Now we briefly analyze leaking and *MismatchCheck* from the utility perspective. Table 7.1 shows the utility matrix of a diligent contractor based on the leaking decision. Not leaking is computationally dominant, as it can be better if there exist non-diligent contractors. In such a case, the contractor can obtain an extra reward by cheater hunting. We note that in this case, leaking results in utility $r - \mathsf{cost}(d)$ due to copiers.

Others ╲ This	Do not leak	Leak
All Diligent and Honest Submission	$r - \mathsf{cost}(d)$	$r - \mathsf{cost}(d)$
Some Lazies and Copiers	$\geq r - \mathsf{cost}(d)$	$r - \mathsf{cost}(d)$

Table 7.1: Utility matrix of a diligent contractor based on leaking.

We stress that not complying during a *MismatchCheck* results in a loss for a diligent contractor with non-negligible probability. This means it is already

disincentivized. When the diligent contractors comply, since the responses of the lazy ones are different, the SC identifies the diligent as the winner and the rest as the loser. It then rewards and fines them, respectively.

7.6.3 Analysis Without Coalitions

We have showed leaking and and not complying is disincentivized in the previous subsection. Now we allow each contractor to choose only Diligent and Lazy as strategies (i.e., honest submission and complying is ensured), and disallow any group of contractors from forming coalitions. Therefore, this setting is also interesting if there is no SC and the boss is managing the verification, as in [Belenkiy et al., 2008, Küpçü, 2017, Biçer et al., 2021]. We start by showing that by adjusting the parameters, one can ensure all-diligent is the unique computational Nash equilibrium in this setting. Table 7.2 provides the utility matrix of the contractor.

Others ╲ This	Diligent	Lazy l_i
All Diligent	$r - \mathsf{cost}(d)$	$r \cdot q_i - f(1 - q_i) - \mathsf{cost}(l_i)$
k Diligent	$\geq r - \mathsf{cost}(d)$	$\leq r \cdot q_i - f(1 - q_i) - \mathsf{cost}(l_i)$
All Lazy L'	$r + f \sum_{l_j \in L'}(1 - q_j) - \mathsf{cost}(d)$	*depends on l_i and L'

Table 7.2: Utility matrix of a contractor if no coalition exists, only honest submission and complying is allowed. k should satisfy $0 < k < n - 1$.

Theorem 18. *If the hash function utilized in Merkle hash tree is collision resistant and the boss sets the fine and reward as* $f > \frac{r}{(n-1)\left(1-\mathsf{n}(\lambda)\right)}$ *for all negligible functions* $\mathsf{n}(\cdot)$ *where* λ *is a security parameter, then in the setting described in this subsection, the strategy with diligent computation, honest submission, and complying during MismatchCheck for each player is the unique computational Nash equilibrium.*

Proof. There exist two main cases that we need check: (1) whether all diligent, honest submission, complying is a computational Nash equilibrium, (2) whether

there exist any other computational Nash equilibrium than all diligent. Regarding the case (2), we use subcases to show no other computational Nash equilibria exist in the system.

Case (1): Check whether all diligent, honest submission, complying is a computational Nash equilibrium. If all the other contractors choose the diligent strategy, w.l.o.g, the utility of a contractor from the diligent strategy and that from a lazy strategy l_i are $r - \mathsf{cost}(d)$ and $r \cdot q_i - f(1 - q_i) - \mathsf{cost}(l_i)$, respectively. Regarding the latter utility, we have $r \cdot q_i - f(1 - q_i) - \mathsf{cost}(l_i) < -f - \mathsf{cost}(l_i) + v_1$ for some negligible utility v_1. Hence, if $r - \mathsf{cost}(d) > -f - \mathsf{cost}(l_i) + v_1$, we have the utility of diligent strategy to be greater for P. We calculate $r - \mathsf{cost}(d) + \mathsf{cost}(l_i) > r - r = 0 > -f + v_1$. Therefore, for all negligible utilities v, it is sufficient to have $f \geq v$ to achieve this setting as the computational Nash equilibrium.

Case (2): Check for existence of any other computational Nash equilibrium. If k of the contractors choose the diligent strategy where $0 < k < n$, the utility of a lazy P from the diligent strategy and that from any lazy strategy l_i are lower bounded by $r - \mathsf{cost}(d)$ and upper bounded by $r \cdot q_i - f(1 - q_i) - \mathsf{cost}(l_i) + v_2$ for some negligible utility v_2, respectively. The former utility may increase compared to Case (1), because the contractor may gain some of the fines. Also, the latter utility may *only* increase compared to Case (1), if the lazy algorithm outputs a correct value and P gain some of the fines. As this occurs with negligible probability, the maximum increase in utility is bounded by a negligible v_2. Regarding the latter utility, we have $r \cdot q_i - f(1 - q_i) - \mathsf{cost}(l_i) + v_2) < -f - \mathsf{cost}(l_i) + \epsilon_1 + v_2 = -f - \mathsf{cost}(l_i) + v_3$ for some negligible utility v_3. The rest of the calculation is simply carried as in Case (1). Therefore, for all negligible utilities v, it is sufficient to have $f > v$ for diligent utility to dominate the lazy one and spoil any computational Nash equilibrium.

For the remaining, we need to show there is no computational Nash equilibrium in which no contractor plays the diligent strategy. Suppose there exists a computational Nash equilibrium in which no contractor plays the diligent strategy. If all the contractors are playing the same lazy strategy l_i, for

each contractor the inequality $r + (n-1)(1-q_i)f - \mathsf{cost}(d) \leq r - \mathsf{cost}(l_i)$ is necessary, as otherwise this is not a computational Nash equilibrium. Then, if $(n-1)(1-q_i)f > \mathsf{cost}(d) - \mathsf{cost}(l_i)$, the computational Nash equilibrium is spoiled. It is straightforward to deduce that $f > \frac{r}{(n-1)\left(1-\mathsf{n}(\lambda)\right)}$ for all negligible function $\mathsf{n}(\cdot)$ is sufficient. If all the contractors are not playing the same lazy strategy, then one of these strategies (say l_i) would result in a utility lower bounded $r - \mathsf{cost}(l_i)$, and each l_j of the others would result in a utility $-f - \mathsf{cost}(l_j) + v_4$ where v_4 is a negligible utility resulting from the multiplication of negligible probability that l_j outputs the same response with l_i and the resulting rewards and fine gains. Therefore, for all negligible utilities v, it is sufficient to have $f > v$, for allowing any contractor to convert l_i and receive a utility increase. This conflicts the supposed computational Nash equilibrium. □

7.6.4 *Completion of Analysis Without Coalitions*

In this subsection, we consider the leaking strategy to extend our analysis in Section 7.6.3. We show that all diligent computation is still a computational Nash equilibrium and that there does not exist any Nash equilibrium where no contractor is honest. We allow ourselves by not requiring all diligent computation and honest submission without leaking strategy as the unique computational Nash equilibrium anymore. This is forced as leaking does not decrease the utility of any diligent player. Yet, it is justified, as the boss still obtains the correct output (guaranteed by the Theorems 19 and 20). More concretely, we still have the following theorems:

Theorem 19. *If the hash function utilized in Merkle hash is collision resistant, and the response submission protocol π_{RS} given in Protocol 7.5 securely emulates the ideal functionality $\mathcal{F}_{RS}$ given in Protocol 7.4; then in the setting described in this subsection, each player following diligent computation, honest submission, and complying without leaking is a computational Nash equilibrium.*

Proof. If all the other contractors follow the strategy diligent computation and

honest submission without leaking, compared to honest submission by copy attack the contractor can only increase its utility by a negligible utility. Further, in this case, by leaking his response the contractor does not change his utility. □

Theorem 20. *If the hash function utilized in Merkle hash is collision resistant, and the response submission protocol π_{RS} given in Protocol 7.5 securely emulates the ideal functionality $\mathcal{F}_{RS}$ given in Protocol 7.4, and the boss sets the fine and reward as $f > \frac{r}{(n-1)\left(1-\mathsf{n}(\lambda)\right)}$ for all negligible functions $\mathsf{n}(\cdot)$ where λ is a security parameter; then in the setting described in this subsection, there exists no computational Nash equilibrium where at least one player does not follow the diligent computation and honest submission with or without leaking strategies.*

Proof. Suppose there exists a computational Nash equilibrium in which no contractor plays the diligent strategy. Then some contractors play the same lazy strategy l_i, each obtaining utility $r - \mathsf{cost}(l_i)$, whereas others may play copy thanks to some leaking players, each obtaining the utility r. However, if one of these latter player plays the diligent computation and honest submission without leaking, he will obtain $(n-1)(1-q_i)f$. If $(n-1)(1-q_i)f > \mathsf{cost}(d) - \mathsf{cost}(l_i) + \mathsf{n}(\lambda)$ for all non-negligible functions $\mathsf{n}(\cdot)$, the computational Nash equilibrium is spoiled. Again it is straightforward to deduce that $f > \frac{r}{(n-1)\left(1-\mathsf{n}(\lambda)\right)}$ for all negligible function $\mathsf{n}(\cdot)$ is sufficient as in the proof of Theorem 7.6.3. □

7.6.5 Analysis With Non-Collaborating Coalitions

In this subsection, by using the computational ℓ-repellence definition, we show that our mechanism is secure against coalitions with up to $n-1$ contractors involved. The individual strategies of each player do not vary from the ones in Section 7.6.1. We define the desirable strategy from a coalition as follows:

Definition 22 (non-collaborating diligent strategy)**.** *Non-collaborating diligent strategy is defined as a non-collaborating coalition strategy where the diligent algorithm is executed by each player in the coalition, the output is submitted honestly, and during MismatchCheck the first submitter of the response complies.*

Theorem 21. *If the hash function utilized in Merkle hash is collision resistant, and the response submission protocol π_{RS} given in Protocol 7.5 securely emulates the ideal functionality $\mathcal{F}_{RS}$ given in Protocol 7.4; then in the setting provided in this subsection, all non-collaborating diligent strategy is computationally $(n-1)$-repellent.*

Proof. Given that all the contractors outside the coalition play the diligent strategy, the utility of the coalition G from all non-collaborating diligent strategy is calculated as $|G|(\cdot r - \mathsf{cost}(d))$. We stress that there must be at least 1 player outside that plays the diligent strategy as theorem covers the coalitions up to size $n-1$. If k people in the coalition submit a response obtained by a lazy algorithm, the coalition utility becomes upper bounded by $(|G|-k)\cdot r - (|G|-k)\mathsf{cost}(d) + v'$, obtained by setting the costs of algorithms run by k deviant strategy players as 0 (for maximum utility), fines paid by them staying inside coalition (again, for maximum utility), and v' as the maximum utility a lazy algorithm can achieve. Clearly, all non-collaborating diligent strategy utility is greater than this one for any $0 < k \leq |G|$ as $r > \mathsf{cost}(d)$. We highlight that obtaining a higher utility requires that some of the players do not compute the function but still submit the correct response. This may succeed either by copying an outsider's response or computing the correct Merkle hash without having the correct computation result, both occurring with negligible probability. □

7.6.6 *Analysis With Collaborating Coalitions*

A coalition may benefit from reducing computation costs by parallelizing the task (or letting only one player handle it), resulting in an only assumed total cost of $\mathsf{cost}(d)$ for a coalition of size k, instead of $k \cdot \mathsf{cost}(d)$ to play the all diligent strategy. Assuming the boss is only interested in obtaining a consensus on the correct result (not in every player executing the computation) as in Section 7.6.5, we obtain an equilibrium where the result extends by the following theorem.

Definition 23 (Collaborating Diligent Strategy)**.** *Collaborating diligent strategy is defined as a collaborating coalition strategy where the diligent algorithm is executed once*

IOC scheme	Number n of Contractors	Smart Contract	Computation Cost to Boss/SC	Ratio f/r	Fast Submission	Cheater Hunt	Coalition/Sybil Proofness
[Belenkiy et al., 2008]	2	N/A	$O(s)$	~ 0	No incentive	N/A	N/A
[Küpçü, 2017]	≥ 2	N/A	$O(s)$	$\sim (n-1)$	No incentive	Volunteered	Via [Biçer et al., 2021]
*	2	Yes	$O(\log s)$	~ 0	No incentive	Forced	N/A
[Biçer et al., 2021]	≥ 2	N/A	$O(s)$	$\sim (n-1)$	No incentive	Volunteered	$n-1$, non-collaborating
`Delegate`	≥ 2	Yes	$O(\log s)$	$\sim 1/(n-1)$	Incentivized	Incentivized	$n-1$, collaborating

Table 7.3: Detailed comparison of our work with the state-of-the-art in terms of our goals listed in Section 7.5.1. * refers to the work of [Küpçü and Safavi-Naini, 2021]. We highlight that lower f/r ratio is better for incentivization of contractors [Belenkiy et al., 2008]. In none of the given mechanisms, computation cost to the boss or SC is incurred when there exists a consensus in the submitted responses. Coalitions in [Biçer et al., 2021] were not considered as collaborating for computation.

in the coalition, the output is submitted honestly by all players in the coalition without leaking, and during MismatchCheck the first submitter of the response complies.

We acknowledge that this is a slight abuse of the previous computational ℓ-repellence definition, where the desired coalition strategy was expected to be the collection of the desired strategies of each contractor inside it. Yet, it is justified due to the following reasons: (1) cost cancellations due to interactions of coalitional partners are inevitable as the same computation is outsourced, (2) the boss only cares about the outcome, i.e., obtaining the correct result.

Theorem 22. *If the hash function utilized in Merkle hash is collision resistant, and the response submission protocol π_{RS} given in Protocol 7.5 securely emulates the ideal functionality $\mathcal{F}_{RS}$ given in Protocol 7.4; then in the setting provided in this subsection, all collaborating diligent strategy is computationally $(n-1)$-repellent.*

Proof. Given that all the contractors outside the coalition play the diligent strategy, the utility of the coalition G from all collaborating diligent strategy is calculated as $|G| \cdot r - \mathsf{cost}(d)$. We stress that there must be at least 1 player outside that plays the diligent strategy as theorem covers the coalitions up to size $n-1$. If k contractors in the coalition submit a response obtained by a lazy algorithm

and a contractors run the diligent algorithm, the coalition utility becomes upper bounded by $(|G| - k) \cdot r - a \cdot \mathsf{cost}(d) + v'$, obtained by setting the costs of algorithms run by k deviant strategy players as 0 (for maximum utility), fines paid by them staying inside coalition (again, for maximum utility), and v' as the maximum utility a lazy algorithm can achieve. Clearly, all collaborating diligent strategy utility is greater than this one for any $0 < k \leq |G|$. We highlight that obtaining a higher utility requires that none of the players compute the function. This may succeed either by copying an outsider's response or computing the correct Merkle hash without having the correct computation result, both occurring with negligible probability. □

7.7 Comparison to State-of-The-Art

We compare our proposal with the existing mechanisms in Table 7.3 concerning the goals listed in Section 7.5.1. We acknowledge that these previous works do not target our listed goals, and it may be possible to further improve them to achieve some of the given features (e.g., it may be possible to improve [Küpçü, 2017, Biçer et al., 2021] by allowing the boss to employ a smart contract and to use our response submission protocol).

The computation cost to the boss or SC only occurs if there does not exist a consensus. of our mechanism in the online case without failures is $O(n \log s)$ but in the worst case, it is $O(n^2 \log s)$ if failures exist. Yet, n or n^2 terms may not be considered as the bottleneck in practical settings and do not grow asymptotically, as for a given value of the correct computation to the boss, the cost of the honest computation inversely bounds the number of contractors. Thus the cost of the computation dominates.

We highlight that a lower f/r ratio is better for incentivization of contractors [Belenkiy et al., 2008]. Therefore, we consider the minimum ratio for each protocol, as this is the amount to be deposited to the SC by each contractor at the beginning.

Chapter 8

CONCLUSION

This thesis provides methods for applying blockchain to realistic problems with the help of the cryptography and incentivization. Usually, practical applications will necessarily involve some combination of both as provable security notion of cryptography is not trivially realizable due to complication of systems and inefficiency. We can briefly mention our contributions as follows:

Regarding VABS (in Chapter 2). We provide a novel security framework for VABS. Our proposal can be used and extended for further ABS schemes that provides similar or more features. Also, we provide the first provably secure VABS construction with modular design. Our scheme is efficient as for a VABS with up to 20 attributes, Sign and Verify operations take below 0.3 and 0.2 sec, respectively, and the generated signature size is below 89 KB). It is also modular as one can choose turning on or off revocation and traceability features.

Regarding e-donation (in Chapter 3). We propose the novel, secure, and decentralized e-donation framework and definitions for its security properties. Our proposal is useful and extendable for further e-donation schemes that provides similar or more features. Also, we provide the first e-donation construction with provable unforgeability and balance, fair distribution of donations, donation and recipient privacy. Our implementation results show that for many cases, donating takes less than 1.7 min and 6 kB, while receiving donations takes around 1.7 min and less than 90 kB.

Regarding BlockSim-Net (in Chapter 4). We convert the BlockSim's [Alharby and van Moorsel, 2020] local simulation to a network simulation. We optimize the simulation by switching to another miner's blockchain when a block is received and by requiring miners only send the last block of their longest local

blockchain to the admin server for consensus. Our simulations show that for Bitcoin, BlockSim-Net results in a reward deviation of 4.85, and BlockSim results in that of 9.16, and for Litecoin, BlockSim-Net results in a reward deviation of 13.29, while BlockSim results in that of 7.59.

Regarding `Fortis` (in Chapter 5). We introduce Bold mining and Oracle mining attacks on timestamp based blockchain defense mechanisms. To analyze SM-type attacks, we propose a useful generic formulas that can be utilized for revenue calculations in selfish mining type of attacks. Also, we propose our selfish mining mitigation algorithm `Fortis` [Biçer and Küpçü, 2020b] based on a global synchronous clock. We provide the simulation results for showing practicality and security of our proposed defense mechanism, even in absence of a fully syncronous clock (i.e., due to realistic delays and glitches).

Regarding m-Stability (in Chapter 6). We propose ℓ-repellence, (ℓ, t)-resistance, and m-stability definitions, to replace the previous definitions of [Abraham et al., 2006]. On three already existing mechanisms, we separately show applicability and advantage of our novel definitions. We improve the multi-party mechanism of [Küpçü, 2017] by fine tuning parameter selection to achieve security against large coalitions.

Regarding `Delegate` (in Chapter 7). We provide a UC-secure [Canetti, 2001] response submission protocol to prevent the copy attack [Avizheh et al., 2019]. We propose our smart-contract-assisted n-party incentivized outsourced computation mechanism: `Delegate`. We analyze and show incentive compatibility of our mechanism in the setting where contractors are allowed to form coalitions and to collaborate for sharing the computation load and the payment of fines and rewards.

Future Work. We have already mentioned some future work for the given problems within the related chapters. Here we will discuss two possible forward steps for overall blockchain security.

An important problem that has not been addressed by this thesis is a complete inclusion of possible adversarial actions in a provable security technique, such as

inclusion of utilities in a universal composability framework. We highlight that Rational Protocol Design of [Garay et al., 2013] is a step towards this purpose, but it is limited by considering a single adversary against the protocol designer. In many cases, the assumption of a single entity taking control of many parties is too restrictive and may rule out normally dependable protocols. A more generic solution to this problem would take into account existence of multiple coalitions (as in the case of m-stability) and set playing the desired strategy an equilibrium such as our ℓ-repellence or m-stability.

Another considerable open problem is the requirement of an adversarial model capable of conducting any attack both on the blockchain and its applications. The state-of-the-art deals with such attacks by separating blockchain security and application security via abstraction of the former as a public ledger. It is quite interesting to investigate this attacker's possible damages when these take place all together, which is exactly the current case. Obviously, right now this seems complicated, yet a single technique may be developed to simplify the system. An adversarial model taking into account all of these threats could be incorporated in the above-mentioned problem or a further work.

BIBLIOGRAPHY

[Eni, 2019] (2019). Enigma web site. `https://enigma.co/`. Accessed: 2020-03-05.

[Abraham et al., 2006] Abraham, I., Dolev, D., Gonen, R., and Halpern, J. (2006). Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation. PODC '06.

[Abraham et al., 2013] Abraham, I., Dolev, D., and Halpern, J. Y. (2013). Distributed protocols for leader election: A game-theoretic perspective. In *DISC '12*.

[Afanasev et al., 2018] Afanasev, M. Y., Fedosov, Y. V., Krylova, A. A., and Shorokhov, S. A. (2018). An application of blockchain and smart contracts for machine-to-machine communications in cyber-physical production systems. In *IEEE ICPS '18*.

[Aiyer et al., 2005] Aiyer, A. S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.-P., and Porth, C. (2005). Bar fault tolerance for cooperative services. *ACM SIGOPS Oper. Syst. Rev.*, 39(5):45–58.

[Alharbi, 2020] Alharbi, M. (2020). Blocksim implementation. (accessed: 01.11.2020).

[Alharby and van Moorsel, 2020] Alharby, M. and van Moorsel, A. (2020). Blocksim: An extensible simulation tool for blockchain systems. *Frontiers in Blockchain*, 3.

[Alm, 2002] Alm, S. E. (2002). *Simple random walk*. `http://www2.math.uu.se/~sea/kurser/stokprocmn1/slumpvandring_eng.pdf`.

[Andrychowicz et al., 2014] Andrychowicz, M., Dziembowski, S., Malinowski, D., and Mazurek, L. (2014). Secure multiparty computations on bitcoin. In *IEEE S&P '14*.

[Andrychowicz et al., 2016] Andrychowicz, M., Dziembowski, S., Malinowski, D., and Mazurek, L. (2016). Secure multiparty computations on bitcoin. *Commun. ACM*, 59(4).

[Aoki et al., 2019] Aoki, Y., Otsuki, K., Kaneko, T., Banno, R., and Shudo, K. (2019). Simblock: A blockchain network simulator. IEEE INFOCOM 2019.

[Arita and Tsurudome, 2009] Arita, S. and Tsurudome, K. (2009). Construction of threshold public-key encryptions through tag-based encryptions. In *ACNS '09*.

[Atzei et al., 2017] Atzei, N., Bartoletti, M., and Cimoli, T. (2017). A survey of attacks on ethereum smart contracts sok. In *POST '17*.

[Atzei et al., 2018] Atzei, N., Bartoletti, M., Cimoli, T., Lande, S., and Zunino, R. (2018). Sok: Unraveling bitcoin smart contracts. In *POST '18*.

[Au et al., 2013] Au, M. H., Susilo, W., Mu, Y., and Chow, S. S. M. (2013). Constant-size dynamic k-times anonymous authentication. *IEEE Systems Journal*, 7(2):249–261.

[Aumann, 1959] Aumann, R. J. (31 Dec. 1959). *16. Acceptable Points in General Cooperative n-Person Games*. Princeton University Press.

[Avizheh et al., 2019] Avizheh, S., Nabi, M., Safavi-Naini, R., and Venkateswarlu K., M. (2019). Verifiable computation using smart contracts. In *ACM CCSW '19*.

[Axon and Goldsmith, 2017] Axon, L. and Goldsmith, M. (2017). Pb-pki: A privacy-aware blockchain-based pki. In *SECRYPT ICETE '17*.

[Badertscher et al., 2018a] Badertscher, C., Garay, J., Maurer, U., Tschudi, D., and Zikas, V. (2018a). But why does it work? a rational protocol design treatment of bitcoin. In *EUROCRYPT '18*.

[Badertscher et al., 2018b] Badertscher, C., Gazi, P., Kiayias, A., Russell, A., and Zikas, V. (2018b). Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *ACM CCS '18*.

[Bag et al., 2017] Bag, S., Ruj, S., and Sakurai, K. (2017). Bitcoin block withholding attack: Analysis and mitigation. *IEEE Transactions on Information Forensics and Security*, 12(8):1967–1978.

[Bahack, 2013] Bahack, L. (2013). Theoretical bitcoin attacks with less than half of the computational power (draft). *arXiv preprint arXiv:1312.7013*.

[Barić and Pfitzmann, 1997] Barić, N. and Pfitzmann, B. (1997). Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT '97*.

[Belenkiy et al., 2008] Belenkiy, M., Chase, M., Erway, C. C., Jannotti, J., Küpçü, A., and Lysyanskaya, A. (2008). Incentivizing outsourced computation. In *NetEcon '08*.

[Bellare and Goldreich, 1993] Bellare, M. and Goldreich, O. (1993). On defining proofs of knowledge. In *CRYPTO' 92*.

[Bellare et al., 2005] Bellare, M., Shi, H., and Zhang, C. (2005). Foundations of group signatures: The case of dynamic groups. In *CT-RSA '05*.

[Bentov and Kumaresan, 2014] Bentov, I. and Kumaresan, R. (2014). How to use bitcoin to design fair protocols. In *CRYPTO '14*.

[Bernheim et al., 1987] Bernheim, B. D., Peleg, B., and Whinston, M. (1987). Coalition-proof nash equilibria i. concepts. *Journal of Economic Theory*, 42(1):1–12.

[Biçer and Küpçü, 2019] Biçer, O. and Küpçü, A. (2019). Versatile abs: Usage limited, revocable, threshold traceable, authority hiding, decentralized attribute based signatures. `https://eprint.iacr.org/2019/203`.

[Bichsel et al., 2014] Bichsel, P., Camenisch, J., Dubovitskaya, M., Enderlein, R. R., Krenn, S., Krontiris, I., Lehmann, A., Neven, G., Nielsen, J. D., Paquin, C., Preiss, F.-S., Rannenberg, K., Sabouri, A., and .Stausholm, M. (2014). D2.2 - Architecture for Attribute-based Credential Technologies - Final Version. Technical report. https://abc4trust.eu/download/Deliverable_D2.2.pdf.

[Bissias and Levine, 2020] Bissias, G. and Levine, B. (2020). Bobtail: Improved blockchain security with low-variance mining. In *NDSS '20*.

[Bitansky and Lin, 2018] Bitansky, N. and Lin, H. (2018). One-message zero knowledge and non-malleable commitments. In *TCC '18*.

[Biçer and Küpçü, 2020a] Biçer, O. and Küpçü, A. (2020a). Anonymous, attribute based, decentralized, secure, and fair e-donation. In *PETS/PoPETS '20*.

[Biçer and Küpçü, 2020b] Biçer, O. and Küpçü, A. (2020b). Fortis: Selfish mining mitigation by (for)geable (ti)me(s)tamps. Cryptology ePrint Archive, Paper 2020/1290. `https://eprint.iacr.org/2020/1290`.

[Biçer and Küpçü, 2022] Biçer, O. and Küpçü, A. (2022). Delegate: Coalition, sybil, and copy proof incentivized outsourced computation with smart contracts. In *Submission to EUROCRYPT*.

[Biçer et al., 2021] Biçer, O., Yildiz, B., and Küpçü, A. (2021). m-stability: Threshold security meets transferable utility. *ACM CCSW '21*.

[Blass and Kerschbaum, 2017] Blass, E.-O. and Kerschbaum, F. (2017). Strain: A secure auction for blockchains. https://eprint.iacr.org/2017/1044.

[Bobba et al., 2006] Bobba, R., Fatemieh, O., Khan, F., Gunter, C. A., and Khurana, H. (2006). Using attribute-based access control to enable attribute-based messaging. In *ACSAC '06*.

[Bobba et al., 2010] Bobba, R., Fatemieh, O., Khan, F., Khan, A., Gunter, C. A., Khurana, H., and Prabhakaran, M. (2010). Attribute-based messaging: Access control and confidentiality. *ACM TISSEC*, 13(4).

[Boneh et al., 2006] Boneh, D., Boyen, X., and Halevi, S. (2006). Chosen ciphertext secure public key threshold encryption without random oracles. In *CT-RSA*.

[Bonneau et al., 2014] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J., and Felten, E. (2014). Mixcoin: Anonymity for bitcoin with accountable mixes. In *FC '14*.

[Bootle et al., 2016] Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., and Groth, J. (2016). Foundations of fully dynamic group signatures. In *ACNS '16*.

[Boudot, 2000] Boudot, F. (2000). Efficient proofs that a committed number lies in an interval. In *EUROCRYPT '00*.

[Bowe et al., 2020] Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., and Wu, H. (2020). Zexe: Enabling decentralized private computation. In *IEEE SP '20*.

[Boyen, 2007] Boyen, X. (2007). Mesh signatures. In *EUROCRYPT '07*.

[Bracha and Toueg, 1985] Bracha, G. and Toueg, S. (1985). Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4).

[Breeze, 2013] Breeze, B. (2013). How donors choose charities: The role of personal taste and experiences in giving decisions. *Voluntary Sector Review*,

4:165–183. https://www.kent.ac.uk/sspssr/philanthropy/documents/How%20Donors%20Choose%20Charities%2018%20June%202010.pdf.

[Brenguier, 2016] Brenguier, R. (2016). Robust equilibria in mean-payoff games. In *FoSSaCS '16*.

[Bünz et al., 2019] Bünz, B., Agrawal, S., Zamani, M., and Boneh, D. (2019). Zether: Towards privacy in a smart contract world. Cryptology ePrint Archive, Report 2019/191.

[Buterin, 2013] Buterin, V. (2013). Ethereum white paper: A next-generation smart contract and decentralized application platform.

[Bünz et al., 2018] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. (2018). Bulletproofs: Short proofs for confidential transactions and more. In *IEEE SP '18*.

[CAF, 2018] CAF (2018). World giving index. https://www.cafonline.org/docs/default-source/about-us-publications/caf_wgi2018_report_webnopw_2379a_261018.pdf?sfvrsn=c28e9140_4.

[Camenisch et al., 2016] Camenisch, J., Drijvers, M., and Hajny, J. (2016). Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs. In *ACM WPES '16*.

[Camenisch et al., 2020] Camenisch, J., Drijvers, M., Lehmann, A., Neven, G., and Towa, P. (2020). Short threshold dynamic group signatures. In *SCN '20*.

[Camenisch and Groß, 2012] Camenisch, J. and Groß, T. (2012). Efficient attributes for anonymous credentials. *ACM Trans. Inf. Syst. Secur.*, 15(1).

[Camenisch and Groth, 2005] Camenisch, J. and Groth, J. (2005). Group signatures: Better efficiency and new theoretical aspects. In *Security in Communication Networks*.

[Camenisch et al., 2006] Camenisch, J., Hohenberger, S., Kohlweiss, M., Lysyanskaya, A., and Meyerovich, M. (2006). How to win the clonewars: Efficient periodic n-times anonymous authentication. In *ACM CCS '06*.

[Camenisch and Lysyanskaya, 2001] Camenisch, J. and Lysyanskaya, A. (2001). An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT '01*.

[Camenisch and Lysyanskaya, 2002] Camenisch, J. and Lysyanskaya, A. (2002). Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO '02*.

[Camenisch and Lysyanskaya, 2003] Camenisch, J. and Lysyanskaya, A. (2003). A signature scheme with efficient protocols. In *SCN '03*.

[Camenisch and Michels, 1999] Camenisch, J. and Michels, M. (1999). Proving in zero-knowledge that a number is the product of two safe primes. In *EUROCRYPT '99*.

[Camenisch and Stadler, 1997] Camenisch, J. and Stadler, M. (1997). Efficient group signature schemes for large groups. In *CRYPTO '97*.

[Canetti, 2001] Canetti, R. (2001). Universally composable security: A new paradigm for cryptographic protocols. In *IEEE FOCS '01*.

[Canetti and Fischlin, 2001] Canetti, R. and Fischlin, M. (2001). Universally composable commitments. In *CRYPTO '01*.

[Canetti and Goldwasser, 1999] Canetti, R. and Goldwasser, S. (1999). An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *EUROCRYPT*.

[Canetti et al., 2011] Canetti, R., Riva, B., and Rothblum, G. N. (2011). Practical delegation of computation using multiple servers. In *ACM CCS '11*.

[Cao et al., 2012] Cao, D., Zhao, B., Wang, X., and Su, J. (2012). Flexible multi-authority attribute-based signature schemes for expressive policy. *Mob. Inf. Syst.*, 8(3).

[Carlsten et al., 2016] Carlsten, M., Kalodner, H., Weinberg, S. M., and Narayanan, A. (2016). On the instability of bitcoin without the block reward. In *ACM CCS '16*.

[Cashlib, 2010] Cashlib (2010). `https://github.com/brownie/cashlib`. Accessed: 2020-02-15.

[Chainlink, 2021] Chainlink (2021). Chainlink's smart contract research. (accessed: 01.06.2021).

[Chaum, 1982] Chaum, D. (1982). Blind signatures for untraceable payments. In *CRYPTO '82*.

[Chaum et al., 1988] Chaum, D., Crépeau, C., and Damgard, I. (1988). Multiparty unconditionally secure protocols. In *STOC '88*.

[Chicarino et al., 2020] Chicarino, V., Albuquerque, C., Jesus, E., and Rocha, A. (2020). On the detection of selfish mining and stalker attacks in blockchain networks. *Ann. Telecommun.*, 75:143–152.

[Chow, 2009] Chow, S. S. M. (2009). Real traceable signatures. In *SAC '09*.

[Company, 2018] Company, E. C. (2018). Zcash "sapling" cryptography. `https://github.com/zcash-hackworks/sapling-crypto`. Accessed: 2020-02-28.

[Cramer et al., 1994] Cramer, R., Damgård, I., and Schoenmakers, B. (1994). Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO '94*.

[Croman et al., 2016] Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Gün Sirer, E., Song, D., and Wattenhofer, R. (2016). On scaling decentralized blockchains. In *FC '16*.

[Damgård and Groth, 2003] Damgård, I. and Groth, J. (2003). Non-interactive and reusable non-malleable commitment schemes. In *ACM STOC '03*.

[Dani et al., 2011] Dani, V., Movahedi, M., Rodriguez, Y., and Saia, J. (2011). Scalable rational secret sharing. In *ACM PODC '11*.

[David et al., 2019] David, A., Maciej, S., Lorenzino, V., and Francesco, P. (2019). Blockchain for digital government. Technical report, Publications Office of the European Union.

[Davidson and Diamond, 2020] Davidson, M. and Diamond, T. (2020). On the profitability of selfish mining against multiple difficulty adjustment algorithms. Cryptology ePrint Archive, Report 2020/094.

[De Santis et al., 1994] De Santis, A., Desmedt, Y., Frankel, Y., and Yung, M. (1994). How to share a function securely. In *STOC '94*.

[Decker and Wattenhofer, 2013] Decker, C. and Wattenhofer, R. (2013). Information propagation in the bitcoin network. In *IEEE P2P '13*.

[Delerablée and Pointcheval, 2008] Delerablée, C. and Pointcheval, D. (2008). Dynamic threshold public-key encryption. In *CRYPTO '08*.

[Derler et al., 2015] Derler, D., Hanser, C., and Slamanig, D. (2015). A new approach to efficient revocable attribute-based anonymous credentials. In *IMACC '15*.

[Desmedt, 2011] Desmedt, Y. (2011). *Threshold Cryptography*. Springer US.

[Deuber et al., 2018] Deuber, D., Maffei, M., Malavolta, G., Rabkin, M., Schröder, D., and Simkin, M. (2018). Functional credentials. In *PETS/PoPETs '18*.

[Di Crescenzo et al., 1998] Di Crescenzo, G., Ishai, Y., and Ostrovsky, R. (1998). Non-interactive and non-malleable commitment. In *ACM STOC '98*.

[Dodis and Yampolskiy, 2005] Dodis, Y. and Yampolskiy, A. (2005). A verifiable random function with short proofs and keys. In *PKC '05*.

[Dolev et al., 1991] Dolev, D., Dwork, C., and Naor, M. (1991). Non-malleable cryptography. In *ACM STOC '91*.

[Dong et al., 2020] Dong, B., Wang, H., Monreale, A., Pedreschi, D., Giannotti, F., and Guo, W. (2020). Authenticated outlier mining for outsourced databases. *IEEE TDSC*, 17(2):222–235.

[Dong et al., 2017] Dong, C., Wang, Y., Aldweesh, A., McCorry, P., and van Moorsel, A. (2017). Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In *ACM CCS '17*.

[Dong et al., 2019] Dong, X., Wu, F., Faree, A., Guo, D., Shen, Y., and Ma, J. (2019). Selfholding: A combined attack model using selfish mining with block withholding attack. *Elsevier Computers & Security*, 87.

[Douceur, 2002] Douceur, J. R. (2002). The sybil attack. In *IPTPS '02*.

[Drăgan et al., 2018] Drăgan, C.-C., Gardham, D., and Manulis, M. (2018). Hierarchical attribute-based signatures. In *CNS '18*.

[Du et al., 2004] Du, W., Jia, J., Manish Mangal, and Mummoorthy Murugesan (2004). Uncheatable grid computing. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*

[Du et al., 2010] Du, W., Murugesan, M., and Jia, J. (2010). *Uncheatable Grid Computing*. Chapman & Hall/CRC.

[Dwork and Naor, 1993] Dwork, C. and Naor, M. (1993). Pricing via processing or combatting junk mail. In *CRYPTO '92*.

[Eidenbenz et al., 2003] Eidenbenz, S., Kumar, V. S. A., and Zust, S. (2003). Equilibria in topology control games for ad hoc networks. In *DIALM-POMC '03*.

[El Kaafarani et al., 2014a] El Kaafarani, A., Chen, L., Ghadafi, E., and Davenport, J. (2014a). Attribute-based signatures with user-controlled linkability. In *CNS '14*.

[El Kaafarani and Ghadafi, 2017] El Kaafarani, A. and Ghadafi, E. (2017). Attribute-based signatures with user-controlled linkability without random oracles. In *Cryptography and Coding*.

[El Kaafarani et al., 2014b] El Kaafarani, A., Ghadafi, E., and Khader, D. (2014b). Decentralized traceable attribute-based signatures. In *CT-RSA '14*.

[Elliott, 2019] Elliott, S. A. (2019). Nash equilibrium of multiple, non-uniform bitcoin block withholding attackers. *ICDIS '19*.

[Emura et al., 2017] Emura, K., Hayashi, T., and Ishida, A. (2017). Group signatures with time-bound keys revisited: A new model and an efficient construction. In *ASIA CCS '17*.

[Etemad and Küpçü, 2015] Etemad, M. and Küpçü, A. (2015). Efficient key authentication service for secure end-to-end communications. ProvSec '15.

[Eyal and Sirer, 2018] Eyal, I. and Sirer, E. G. (2018). Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM*, 61(7).

[Farrell and Rabin, 1996] Farrell, J. and Rabin, M. (1996). Cheap talk. *Journal of Economic Perspectives*, 10(3):103–118.

[Feldman et al., 2004] Feldman, M., Papadimitriou, C., Chuang, J., and Stoica, I. (2004). Free-riding and whitewashing in peer-to-peer systems. In *ACM SIGCOMM Workshop*.

[Fiat and Shamir, 1987] Fiat, A. and Shamir, A. (1987). How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO '86*.

[Fischlin et al., 2011] Fischlin, M., Libert, B., and Manulis, M. (2011). Non-interactive and re-usable universally composable string commitments with adaptive security. In *ASIACRYPT '11*.

[Frikken et al., 2006] Frikken, K. B., Li, J., and Atallah, M. J. (2006). Trust negotiation with hidden credentials, hidden policies, and policy cycles. In *NDSS '06*.

[Fuchsbauer et al., 2019] Fuchsbauer, G., Hanser, C., and Slamanig, D. (2019). Structure-preserving signatures on equivalence classes and constant-size anonymous credentials. *J Cryptol*, 32.

[Fuchsbauer et al., 2010] Fuchsbauer, G., Katz, J., and Naccache, D. (2010). Efficient rational secret sharing in standard communication networks. In *TCC '10*.

[Fujisaki, 2014] Fujisaki, E. (2014). All-but-many encryption: A new framework for fully-equipped uc commitments. In *ASIACRYPT '14*.

[Fujisaki, 2016] Fujisaki, E. (2016). Improving practical uc-secure commitments based on the ddh assumption. In *SCN '16*.

[Fujisaki and Okamoto, 1997] Fujisaki, E. and Okamoto, T. (1997). Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*.

[Gan et al., 2013] Gan, Y., Wang, L., Wang, L., Pan, P., and Yang, Y. (2013). Efficient construction of cca-secure threshold pke based on hashed diffie-hellman assumption. *Computer Journal*, 56:1249–1257.

[Garay et al., 2013] Garay, J., Katz, J., Maurer, U., Tackmann, B., and Zikas, V. (2013). Rational protocol design: Cryptography against incentive-driven adversaries. In *FOCS '13*.

[Garay et al., 2015] Garay, J. A., Kiayias, A., and Leonardos, N. (2015). The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT '15*.

[Garay et al., 2017] Garay, J. A., Kiayias, A., and Leonardos, N. (2017). The bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO '17*.

[Gardham and Manulis, 2022] Gardham, D. and Manulis, M. (2022). Revocable hierarchical attribute-based signatures from lattices. In *ACNS '22*.

[Garman et al., 2014] Garman, C., Green, M., and Miers, I. (2014). Decentralized anonymous credentials. In *NDSS '14*.

[Gennaro et al., 2010] Gennaro, R., Gentry, C., and Parno, B. (2010). Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO '10*.

[Gennaro et al., 2007] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. (2007). Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol.*, 20(1).

[Gervais et al., 2016] Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., and Capkun, S. (2016). On the security and performance of proof of work blockchains. *ACM CCS '16*.

[Ghadafi, 2014] Ghadafi, E. (2014). Efficient distributed tag-based encryption and its application to group signatures with efficient distributed traceability. In *LATINCRYPT '14*.

[Ghadafi, 2015] Ghadafi, E. (2015). Stronger security notions for decentralized traceable attribute-based signatures and more efficient constructions. In *CT-RSA '15*.

[Goldreich et al., 1991] Goldreich, O., Micali, S., and Wigderson, A. (1991). Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728.

[Goldwasser et al., 2015] Goldwasser, S., Kalai, Y. T., and Rothblum, G. N. (2015). Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4).

[Goldwasser et al., 1985] Goldwasser, S., Micali, S., and Rackoff, C. (1985). The knowledge complexity of interactive proof-systems. In *ACM STOC '85*.

[Golle and Mironov, 2001] Golle, P. and Mironov, I. (2001). Uncheatable distributed computations. In *CT-RSA '01*.

[Google, 2021] Google (2021). Certificate transperancy project. (accessed: 21.03.2021).

[Gordon and Katz, 2006] Gordon, S. D. and Katz, J. (2006). Rational secret sharing, revisited. In *SCN '06*.

[Goyal et al., 2016] Goyal, V., Pandey, O., and Richelson, S. (2016). Textbook non-malleable commitments. In *ACM STOC '16*.

[Groth and Sahai, 2012] Groth, J. and Sahai, A. (2012). Efficient noninteractive proof systems for bilinear groups. *SIAM Journal on Computing*, 41(5).

[Göbel et al., 2016] Göbel, J., Keeler, H., Krzesinski, A., and Taylor, P. (2016). Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay. *Performance Evaluation*, 104:23–41.

[Halpern and Vilaça, 2016] Halpern, J. Y. and Vilaça, X. (2016). Rational consensus: Extended abstract. In *ACM PODC '16*.

[Hampiholi et al., 2015] Hampiholi, B., Alpár, G., v. d. Broek, F., and Jacobs, B. (2015). Towards practical attribute-based signatures. In *Security, Privacy, and Applied Cryptography Engineering*.

[Harper, 2021] Harper, C. (2021). Are you running a bitcoin node? `https://www.coindesk.com/are-you-running-a-bitcoin-node` (accessed: 02.02.2021).

[Hassanzadeh-Nazarabadi et al., 2021] Hassanzadeh-Nazarabadi, Y., Küpçü, A., and Özkasap, O. (2021). Lightchain: Scalable dht-based blockchain. *IEEE TPDS*, 32(10):2582–2593.

[Heilman, 2014] Heilman, E. (2014). One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner (poster abstract). In *FC '14*.

[Heilman et al., 2015] Heilman, E., Kendler, A., Zohar, A., and Goldberg, S. (2015). Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX SEC '15*.

[Heller, 2008] Heller, Y. (2008). Ex-ante and ex-post strong correlated equilbrium. MPRA Paper 7717, University Library of Munich, Germany.

[Hopwood et al., 2020] Hopwood, D., Bowe, S., Hornby, T., and Wilcox, N. (2020). Zcash protocol specification. `https://raw.githubusercontent.com/zcash/zips/master/protocol/protocol.pdf`.

[Hou et al., 2021] Hou, C., Zhou, M., Ji, Y., Daia, P., Tramèr, F., Fanti, G., and Juels, A. (2021). Squirrl: Automating attack analysis on blockchain incentive mechanisms with deep reinforcement learning. In *NDSS '21*.

[Huang et al., 2021] Huang, Z., Lin, Z., Chen, Q., Zhou, Y., and Huang, H. (2021). Outsourced attribute-based signatures with perfect privacy for circuits in cloud computing. *Concurrency and Computation: Practice and Experience*, 33.

[Hwang et al., 2015] Hwang, J. Y., Chen, L., Cho, H. S., and Nyang, D. (2015). Short dynamic group signature scheme supporting controllable linkability. *IEEE Transactions on Information Forensics and Security*, 10(6):1109–1124.

[Jiang and Tian, 2019] Jiang, X. and Tian, Y. (2019). How to construct rational protocols with nash equilibrium consistency in the uc framework. `https://eprint.iacr.org/2019/1398`.

[Kalodner et al., 2018] Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S. M., and Felten, E. W. (2018). Arbitrum: Scalable, private smart contracts. In *USENIX SEC'18*.

[Kamara and Küpçü, 2018] Kamara, S. and Küpçü, A. (2018). Dogfish: Decentralized optimistic game-theoretic file sharing. In *ACNS '18*.

[Katz, 2008] Katz, J. (2008). Bridging game theory and cryptography: Recent results and future directions. In *TCC '08*.

[Katz and Lindell, 2007] Katz, J. and Lindell, Y. (2007). *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2nd edition.

[Kerber et al., 2020] Kerber, T., Kiayias, A., and Kohlweiss, M. (2020). Kachina - foundations of private smart contracts. In *IEEE CSF '21*. Available at `https://eprint.iacr.org/2020/543`.

[Khan et al., 2021] Khan, K. M., Arshad, J., and Khan, M. M. (2021). Empirical analysis of transaction malleability within blockchain-based e-voting. *Elsevier Computers & Security*, 100.

[Khurana, 2017] Khurana, D. (2017). Round optimal concurrent non-malleability from polynomial hardness. In *TCC17*.

[Khurana and Sahai, 2017] Khurana, D. and Sahai, A. (2017). How to achieve non-malleability in one or two rounds. In *IEEE FOCS '17*.

[Kiayias et al., 2016a] Kiayias, A., Koutsoupias, E., Kyropoulou, M., and Tselekounis, Y. (2016a). Blockchain mining games. In *ACM EC '16*.

[Kiayias et al., 2016b] Kiayias, A., Koutsoupias, E., Kyropoulou, M., and Tselekounis, Y. (2016b). Blockchain mining games. In *ACM EC '16*.

[Kiayias et al., 2017] Kiayias, A., Russell, A., David, B., and Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO '17*.

[Kiayias et al., 2016c] Kiayias, A., Zhou, H., and Zikas, V. (2016c). Fair and robust multi-party computation using a global transaction ledger. In *EUROCRYPT '16*.

[King and Nadal, 2012] King, S. and Nadal, S. (2012). Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. Available at `http://www.peercoin.net/`.

[Kosba et al., 2016] Kosba, A., Miller, A., Shi, E., Wen, Z., and Papamanthou, C. (2016). Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE SP '16*.

[Kothapalli et al., 2017] Kothapalli, A., Miller, A., and Borisov, N. (2017). Smartcast: An incentive compatible consensus protocol using smart contracts. In *FC '17*.

[Koutsoupias et al., 2019] Koutsoupias, E., Lazos, P., Ogunlana, F., and Serafino, P. (2019). Blockchain mining games with pay forward. In *WWW '19*.

[Kubilay et al., 2018] Kubilay, M. Y., Kiraz, M. S., and Mantar, H. A. (2018). Certledger: A new PKI model with certificate transparency based on blockchain. *arXiv preprint arXiv:1806.03914*. http://arxiv.org/abs/1806.03914.

[Kumaresan and Bentov, 2014] Kumaresan, R. and Bentov, I. (2014). How to use bitcoin to incentivize correct computations. In *ACM CCS '14*.

[Kumaresan et al., 2016] Kumaresan, R., Vaikuntanathan, V., and Vasudevan, P. N. (2016). Improvements to secure computation with penalties. In *ACM CCS '16*.

[Küpçü, 2017] Küpçü, A. (2017). Incentivized outsourced computation resistant to malicious contractors. *IEEE TDSC*, 14(6):633–649.

[Küpçü and Safavi-Naini, 2021] Küpçü, A. and Safavi-Naini, R. (2021). Smart contracts for incentivized outsourcing of computation. In *ESORICS CBT '21*.

[Kılınç-Alper, 2019] Kılınç-Alper, H. (2019). Network time with a consensus on clock. https://eprint.iacr.org/2019/1348.

[Lamport et al., 1982] Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3).

[Leelavimolsilp et al., 2019] Leelavimolsilp, T., Tran-Thanh, L., Stein, S., and Nguyen, V. H. (2019). Selfish mining in proof-of-work blockchain with multiple miners: An empirical evaluation. *arXiv preprint arXiv:1905.06853*.

[Li et al., 2006] Li, H., Clement, A., Wong, E., Napper, J., Roy, I., Alvisi, L., and Dahlin, M. (2006). Bar gossip. In *OSDI '06*.

[Li et al., 2010] Li, J., Au, M. H., Susilo, W., Xie, D., and Ren, K. (2010). Attribute-based signature and its applications. In *ACM ASIACCS '10*.

[Libert and Yung, 2011] Libert, B. and Yung, M. (2011). Adaptively secure non-interactive threshold cryptosystems. In *ICALP '11*.

[Lin and Pass, 2011] Lin, H. and Pass, R. (2011). Constant-round nonmalleable commitments from any one-way function. In *ACM STOC '11*.

[Lin et al., 2017] Lin, H., Pass, R., and Soni, P. (2017). Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In *IEEE FOCS '17*.

[Lin et al., 2009] Lin, H., Pass, R., and Venkitasubramaniam, M. (2009). A unified framework for concurrent security: Universal composability from stand-alone non-malleability. In *ACM STOC '09*.

[Lin and Sun, 2013] Lin, X. J. and Sun, L. (2013). Efficient cca-secure threshold public-key encryption scheme. Cryptology ePrint Archive, Report 2013/749.

[Litecoin, 2011] Litecoin (2011). `https://litecoin.org/`. Accessed: 2020-01-15.

[Liu et al., 2018] Liu, H., Ruan, N., Du, R., and Jia, W. (2018). On the strategy and behavior of bitcoin mining with n-attackers. In *ASIACCS '18*.

[Liu and Wong, 2005] Liu, J. K. and Wong, D. S. (2005). Linkable ring signatures: Security models and new schemes. In *ICCSA '05*.

[Lorek and Markowski, 2018] Lorek, P. and Markowski, P. (2018). Conditional gambler's ruin problem with arbitrary winning and losing probabilities with applications. *arXiv preprint arXiv:1812.00687*.

[Lueks et al., 2015] Lueks, W., Alpar, G., Hoepman, J.-H., and Vullers, P. (2015). Fast revocation of attribute-based credentials for both users and verifiers. volume 455, pages 463–478.

[Lysyanskaya and Triandopoulos, 2006] Lysyanskaya, A. and Triandopoulos, N. (2006). Rationality and adversarial behavior in multi-party computation. In *CRYPTO '06*.

[Maggs et al., 2012] Maggs, M., O'Keefe, S., and Thiel, D. (2012). Consensus clock synchronization for wireless sensor networks. *Sensors Journal, IEEE*, 12:2269–2277.

[Maji et al., 2011] Maji, H. K., Prabhakaran, M., and Rosulek, M. (2011). Attribute-based signatures. In *CT-RSA '11*.

[Manshaei et al., 2013] Manshaei, M. H., Zhu, Q., Alpcan, T., Bacşar, T., and Hubaux, J.-P. (2013). Game theory meets network security and privacy. *ACM Comput. Surv.*, 45(3).

[Marmolejo-Cossío et al., 2019] Marmolejo-Cossío, F. J., Brigham, E., Sela, B., and Katz, J. (2019). Competing (semi-)selfish miners in bitcoin. ACM AFT '19.

[Mashamba-Thompson and Crayton, 2020] Mashamba-Thompson, T. and Crayton, E. (2020). Blockchain and artificial intelligence technology for novel coronavirus disease 2019 self-testing. *Diagnostics*, 10(198).

[Meiklejohn et al., 2010] Meiklejohn, S., Erway, C. C., Küpçü, A., Hinkle, T., and Lysyanskaya, A. (2010). Zkpdl: A language-based system for efficient zero-knowledge proofs and electronic cash. USENIX Security '10.

[Miers et al., 2013] Miers, I., Garman, C., Green, M., and Rubin, A. D. (2013). Zerocoin: Anonymous distributed e-cash from bitcoin. In *IEEE S&P*.

[Molnar, 2000] Molnar, D. (2000). The seti@home problem. ACM Crossroads. `http://www.acm.org/crossroads/columns/onpatrol/september2000.html`.

[Monero, 2014] Monero (2014). `https://www.getmonero.org/`. Accessed: 2020-01-15.

[Möser et al., 2018] Möser, M., Soska, K., Heilman, E., Lee, K., Heffan, H., Srivastava, S., Hogan, K., Hennessey, J., Miller, A., Narayanan, A., and Christin, N. (2018). An empirical analysis of traceability in the monero blockchain. *PoPETs*.

[Murthy et al., 2020] Murthy, C., Shri, M. L., Kadry, S., and Lim, S. (2020). Blockchain based cloud computing: Architecture and research challenges. *IEEE Access*, 8.

[Nakamoto, 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

[Narayanan et al., 2016] Narayanan, A., Bonneau, J., Felten, E., Miller, A., and Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies*, volume 1 of *Fundamental Algorithms*. Princeton University Press, 3rd edition. (book).

[Naserian and Tepe, 2009] Naserian, M. and Tepe, K. (2009). Game theoretic approach in routing protocol for wireless ad hoc networks. *Ad Hoc Networks*, 7:569–578.

[Nayak et al., 2016] Nayak, K., Kumar, S., Miller, A., and Shi, E. (2016). Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. IEEE EuroS&P 2016.

[Nayak et al., 2016] Nayak, K., Kumar, S., Miller, A., and Shi, E. (2016). Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *IEEE EuroS&P '16*.

[Nguyen, 2005] Nguyen, L. (2005). Accumulators from bilinear pairings and applications. In *CT-RSA '05*.

[Nisansala et al., 2017] Nisansala, M., Perera, S., and Koshiba, T. (2017). Fully secure lattice-based group signatures with verifier-local revocation. In *IEEE, AINA '17*.

[Okamoto and Takashima, 2013] Okamoto, T. and Takashima, K. (2013). Decentralized attribute-based signatures. In *PKC '13*.

[Optimal selfish mining strategies implementation, 2017] Optimal selfish mining strategies implementation (2017). `https://github.com/nirenzang/Optimal-Selfish-Mining-Strategies-in-Bitcoin`. Accessed: 2022-03-29.

[Park et al., 2019] Park, S., Im, S., Seol, Y., and Paek, J. (2019). Nodes in the bitcoin network: Comparative measurement study and survey. *IEEE Access*, 7:57009–57022.

[Parno et al., 2012] Parno, B., Raykova, M., and Vaikuntanathan, V. (2012). How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC '12*.

[Pass and Rosen, 2005] Pass, R. and Rosen, A. (2005). Concurrent non-malleable commitments. In *IEEE FOCS '05*.

[Pass et al., 2017] Pass, R., Seeman, L., and Shelat, A. (2017). Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT '17*.

[Pass and Wee, 2010] Pass, R. and Wee, H. (2010). Constant-round non-malleable commitments from sub-exponential one-way functions. In *EUROCRYPT '10*.

[Pedersen, 1991a] Pedersen, T. P. (1991a). A threshold cryptosystem without a trusted party. In *EUROCRYPT '91*.

[Pedersen, 1991b] Pedersen, T. P. (1991b). A threshold cryptosystem without a trusted party. In *EUROCRYPT*.

[Pedersen, 1992] Pedersen, T. P. (1992). Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91*.

[Peters, 2008] Peters, H. (2008). *Cooperative Games with Transferable Utility*. Springer Berlin Heidelberg.

[Popov, 2017] Popov, S. (2017). The tangle. Available at `http://www.iota.org/IOTA_Whitepaper.pdf`.

[Premnath and Haas, 2015] Premnath, S. N. and Haas, Z. J. (2015). Security and privacy in the internet-of-things under time-and-budget-limited adversary model. *IEEE Wireless Commun. Lett.*, 4(3).

[Ramachandran et al., 2022] Ramachandran, P., Agrawal, N., Biçer, O., and Küpçü, A. (2022). Blocksim-net: a network-based blockchain simulator. *Turkish Journal of Electrical Engineering and Computer Sciences*, 30.

[Ramani et al., 2019] Ramani, S. K., Tourani, R., Torres, G., Misra, S., and Afanasyev, A. (2019). Ndn-abs: Attribute-based signature scheme for named data networking. In *ICN '19*.

[Ritz and Zugenmaier, 2018] Ritz, F. and Zugenmaier, A. (2018). The impact of uncle rewards on selfish mining in ethereum. In *IEEE Euro S&P '18 Workshop*.

[Rivest et al., 2006] Rivest, R., Shamir, A., and Tauman, Y. (2006). How to leak a secret: Theory and applications of ring signatures. In *TCS '06*.

[Roby, 2017] Roby, N. (2017). Application of blockchain technology in online voting. `https://www.rsaconference.com/writable/files/About/application_of_blockchain_technology_in_online_voting.pdf`.

[Ruffing et al., 2014] Ruffing, T., Moreno-Sanchez, P., and Kate, A. (2014). Coinshuffle: Practical decentralized coin mixing for bitcoin. In *ESORICS '14*.

[Saad et al., 2019] Saad, M., Njilla, L., Kamhoua, C. A., and Mohaisen, A. (2019). Countering selfish mining in blockchains. *ICNC '19*.

[Sadeghi et al., 2010] Sadeghi, A.-R., Schneider, T., and Winandy, M. (2010). Token-based cloud computing. In *TRUST '10*.

[Sakai et al., 2016] Sakai, Y., Attrapadung, N., and Hanaoka, G. (2016). Attribute-based signatures for circuits from bilinear map. In *PKC '16*.

[Sakai et al., 2015] Sakai, Y., Emura, K., Schuldt, J., Hanaoka, G., and Ohta, K. (2015). Dynamic threshold public-key encryption with decryption consistency from static assumptions. In *ISP '15*.

[Sakai et al., 2018] Sakai, Y., Katsumata, S., Attrapadung, N., and Hanaoka, G. (2018). Attribute-based signatures for unbounded languages from standard assumptions. In *ASIACRYPT '18*.

[Sakai et al., 2012] Sakai, Y., Schuldt, J. C. N., Emura, K., Hanaoka, G., and Ohta, K. (2012). On the security of dynamic group signatures: Preventing signature hijacking. In *PKC '12*.

[Sanders, 2020] Sanders, O. (2020). Efficient redactable signature and application to anonymous credentials. In *PKC '20*.

[Sapirshtein et al., 2017] Sapirshtein, A., Sompolinsky, Y., and A., Z. (2017). Optimal selfish mining strategies in bitcoin. Financial Cryptography 2017.

[Sapirshtein et al., 2016] Sapirshtein, A., Sompolinsky, Y., and Zohar, A. (2016). Optimal selfish mining strategies in bitcoin. In *FC '16*.

[Sarmenta, 2002] Sarmenta, L. F. G. (2002). Sabotage-tolerance mechanisms for volunteer computing systems. *Future Gener. Comput. Syst.*, 18(4).

[Sasson et al., 2014] Sasson, E. B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., and Virza, M. (2014). Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE SP '14*.

[Schnorr, 1991] Schnorr, C. P. (1991). Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3).

[Schreiberr, 1973] Schreiberr, F. R. (1973). *Sybil*. Warner Books.

[Shahandashti and Safavi-Naini, 2009] Shahandashti, S. F. and Safavi-Naini, R. (2009). Threshold attribute-based signatures and their application to anonymous credential systems. In *AFRICACRYPT '09*.

[Shapley and Shubik, 1969] Shapley, L. and Shubik, M. (1969). On market games. *Journal of Economic Theory*, 1(1):9–25.

[Shi et al., 2020] Shi, S., He, D., Li, L., Kumar, N., Khan, M. K., and Choo, K.-K. R. (2020). Applications of blockchain in ensuring the security and privacy of electronic health record systems: A survey. *Elsevier Computers & Security*, 97.

[Shultz, 2015] Shultz, B. (2015). Certification of witness: Mitigating blockchain fork attacks. Available at `http://bshultz.com/paper/ShultzThesis.pdf`.

[Singh et al., 2019] Singh, P. K., Singh, R., Nandi, S. K., and Nandi, S. (2019). Managing smart home appliances with proof of authority and blockchain. In *I4CS '19*.

[Slamanig et al., 2014] Slamanig, D., Spreitzer, R., and Unterluggauer, T. (2014). Adding controllable linkability to pairing-based group signatures for free. In *ISC '14*.

[Solat and Potop-Butucaru, 2017] Solat, S. and Potop-Butucaru, M. (2017). Zeroblock: Timestamp-free prevention of block-withholding attack in bitcoin. In *SSS '17*.

[Sompolinsky et al., 2017] Sompolinsky, Y., Lewenberg, Y., and Zohar, A. (2017). Spectre : Serialization of proof-of-work events : Confirming transactions via recursive elections. Available at `https://eprint.iacr.org/2016/1159.pdf`.

[Sompolinsky and Zohar, 2015] Sompolinsky, Y. and Zohar, A. (2015). Secure high-rate transaction processing in bitcoin. In *FC '15*.

[Sompolinsky and Zohar, 2018] Sompolinsky, Y. and Zohar, A. (2018). Phantom , ghostdag : Two scalable blockdag protocols. `https://eprint.iacr.org/2018/104.pdf`.

[Stoykov et al., 2017] Stoykov, L., Zhang, K., and Jacobsen, H. A. (2017). Vibes: fast blockchain simulations for large-scale peer-to-peer networks. ACM/IFIP/USENIX Middleware Conference: Posters and Demos 2017.

[Syverson et al., 2004] Syverson, P., Dingledine, R., and Mathewson, N. (2004). Tor: The second generation onion router. In *Usenix Sec. '04*.

[Szajda et al., 2003] Szajda, D., Lawson, B., and Owen, J. (2003). Hardening functions for large scale distributed computations. In *2003 Symposium on Security and Privacy, 2003*.

[Szajda et al., 2003] Szajda, D., Lawson, B., and Owen, J. (2003). Hardening functions for large scale distributed computations. In *IEEE SP '03*.

[Szajda et al., 2005] Szajda, D., Lawson, B., and Owen, J. (2005). Toward an optimal redundancy strategy for distributed computations. In *IEEE Cluster '05*.

[Teranishi et al., 2004] Teranishi, I., Furukawa, J., and Sako, K. (2004). k-times anonymous authentication (extended abstract). In *ASIACRYPT '04*.

[Thaler et al., 2012] Thaler, J., Roberts, M., Mitzenmacher, M., and Pfister, H. (2012). Verifiable computation with massively parallel interactive proofs. In *USENIX HotCloud '12*.

[Tian et al., 2019] Tian, Z., Gao, X., Su, S., Qiu, J., Du, X., and Guizani, M. (2019). Evaluating reputation management schemes of internet of vehicles based on evolutionary game theory. *IEEE Trans. Veh*, 68(6).

[Tramèr et al., 2020] Tramèr, F., Boneh, D., and Paterson, K. G. (2020). Remote side-channel attacks on anonymous transactions. Cryptology ePrint Archive, Report 2020/220.

[Tsabary and Eyal, 2018] Tsabary, I. and Eyal, I. (2018). The gap game. In *ACM CCS '18*.

[Tsang et al., 2007] Tsang, P. P., Au, M. H., Kapadia, A., and Smith, S. W. (2007). Blacklistable anonymous credentials: Blocking misbehaving users without ttps. In *ACM CCS '07*.

[Upadhyay, J., 2015] Upadhyay, J. (2015). Integrity and privacy of large data.

[Urquidi et al., 2016] Urquidi, M., Khader, D., Lancrenon, J., and Chen, L. (2016). Attribute-based signatures with controllable linkability. In *Trusted Systems '16*.

[v. Saberhagen, 2013] v. Saberhagen, N. (2013). Cryptonote v 2.0, https://cryptonote.org/whitepaper.pdf.

[Veronese et al., 2013] Veronese, G. S., Correia, M., Bessani, A. N., Lung, L. C., and Verissimo, P. (2013). Efficient byzantine fault-tolerance. *IEEE ToC*, 62(1).

[von Ahn et al., 2006] von Ahn, L., Bortz, A., Hopper, N. J., and O'Neill, K. (2006). Selectively traceable anonymity. In *PET '06*.

[Walfish and Blumberg, 2015] Walfish, M. and Blumberg, A. J. (2015). Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84.

[Wang et al., 2011] Wang, S., Ma, R., Zhang, Y., and Wang, X. (2011). Ring signature scheme based on multivariate public key cryptosystems. *COMPUT MATH APPL*, 62(10).

[Wang et al., 2019a] Wang, T., Liew, S. C., and Zhang, S. (2019a). When blockchain meets ai: Optimal mining strategy achieved by machine learning. *arXiv preprint arXiv:1911.12942*.

[Wang et al., 2012] Wang, Y., Wang, H., and Xu, Q. (2012). Rational secret sharing with semi-rational players. *Int. J. Grid Util. Comput.*, 3(1):59–67.

[Wang et al., 2019b] Wang, Z., Liu, J., Wu, Q., Zhang, Y., Yu, H., and Zhou, Z. (2019b). An analytic evaluation for the impact of uncle blocks by selfish and stubborn mining in an imperfect ethereum network. *Elsevier Computers & Security*, 87.

[Wee, 2011] Wee, H. (2011). Threshold and revocation cryptosystems via extractable hash proofs. In *EUROCRYPT '11*.

[Yaga et al., 2018] Yaga, D., Mell, P., Roby, N., and Scarfone, K. (2018). Blockchain Technology Overview. Technical report, NIST. https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf.

[Yakubov et al., 2018] Yakubov, A., Shbair, W. M., Wallbom, A., Sanda, D., and State, R. (2018). A blockchain-based pki management framework. IEEE/IFIP NOMS '18.

[Yang et al., 2011] Yang, G., Tang, S., and Yang, L. (2011). A novel group signature scheme based on mpkc. In *ISPEC '11*.

[Yang et al., 2020] Yang, R., Chang, X., Mišić, J., and Mišić, V. B. (2020). Assessing blockchain selfish mining in an imperfect network: Honest and selfish miner views. *Elsevier Computers & Security*, 97.

[Zhang and Preneel, 2017] Zhang, R. and Preneel, B. (2017). Publish or perish: A backward-compatible defense against selfish mining in bitcoin. In *CT-RSA '17*.

[Zheng et al., 2020] Zheng, Z., Xie, S., Dai, H.-N., Chen, W., Chen, X., Weng, J., and Imran, M. (2020). An overview on smart contracts: Challenges, advances and platforms. *Elsevier FGSC*, 105:475–491.

[Ziegeldorf et al., 2015] Ziegeldorf, J. H., Grossmann, F., Henze, M., Inden, N., and Wehrle, K. (2015). Coinparty: Secure multi-party mixing of bitcoins. In *ACM CODASPY '15*.