University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Tomasz Lizurej

On Security of Systems Built on Blockchains

PhD Dissertation

Supervisor
prof. dr hab. Stefan Dziembowski
Institute of Informatics
University of Warsaw

Auxiliary Supervisor
dr Tomasz Michalak
Institute of Informatics
University of Warsaw

March 2024

Author's declaration:
I hereby declare that this dissertation is my own work.

March 1, 2024 .....
Tomasz Lizurej

Supervisor's declaration:
The dissertation is ready to be reviewed.

March 1, 2024 .....
prof. dr hab. Stefan Dziembowski

Auxiliary Supervisor's declaration:
The dissertation is ready to be reviewed.

March 1, 2024 .....
dr Tomasz Michalak

Abstract

While the security of blockchains hinges on their underlying architecture, this thesis emphasizes the necessity for a comprehensive analysis of various security aspects of *systems* built upon blockchain technology. In particular, the thesis studies the reliability of rating systems designed for blockchain trading platforms, conducts a formal analysis of algorithms securing against distributed cryptography attacks, and proposes a compiler for devices safeguarding against wire-tampering attacks.

In the context of the reliability of rating systems, the study provides insights into the manipulability of the Fairness-Goodness-Algorithm (FGA), a system demonstrated to be well-suited for rating graphs akin to those found in blockchain trading platforms (Kumar et al. 2016). Our research draws a clear distinction between direct and indirect manipulations of a selected victim node in graphs and demonstrates that the FGA is challenging to manipulate indirectly in real-world scenarios. A collection of formal and experimental findings is presented to substantiate and reinforce this conclusion.

In the context of security against distributed cryptography attacks, we introduce the concept of individual cryptography, a paradigm preventing sharing or dividing secrets needed to compute algorithms. We define a formal model for individual cryptography, and present an example scheme in this model - a Proof of Individual Knowledge protocol. The Proof of Individual Knowledge allows one to prove that a secret key is stored on a single machine. Along with the protocol, we show its application to the security of electronic voting systems implemented on blockchains.

Lastly, the thesis addresses the security of devices used in blockchain ecosystems, presenting a compiler for circuits that withstand wire-tampering attacks. The wire-tampering attacks allow an adversarial device producer or a holder to manipulate the specifications of the device's wires maliciously. Our solution offers provable guarantees even in the presence of modifications to every wire of the circuit.

The results presented in this thesis underscore the need to broaden the scope of blockchain security research beyond architectural designs, emphasizing the significance of analyzing diverse security facets for robust and resilient blockchain ecosystems.

Keywords: Blockchain; Security; Cryptography; Hardware security; Rating Systems Manipulations.

ACM classification: Security and privacy → Social network security and privacy. Security and privacy → Cryptography. Security and privacy → Security in Hardware.

Streszczenie

Gdy mówimy o bezpieczeństwie technologii blockchain, zazwyczaj mamy na myśli architekturę danego oprogramowania. Rozprawa ta wskazuje jednak na konieczność przeprowadzenia kompleksowej analizy różnorodnych aspektów bezpieczeństwa *systemów*, które bazują na technologii blockchain. W szczególności, praca ta bada niezawodność systemów pozycjonowania (ang. „rating systems") przystosowanych do serwisów wymiany cyfrowych aktywów (ang. „trading platforms"), przeprowadza formalną analizę algorytmów zabezpieczających przed atakami kryptografii rozproszonej i bada bezpieczeństwo urządzeń wykorzystywanych przez użytkowników technologii blockchain.

W kontekście niezawodności systemów pozycjonowania, rozprawa dokonuje szczegółowej oceny manipulowalności algorytmu Fairness-Goodness-Algorithm (FGA), systemu pozycjonowania służącego do oceny uczestników sieci analogicznych do sieci tworzonych przez użytkowników platform wymiany cyfrowych aktywów (Kumar et al. 2016). Nasze badania wskazują na różnicę pomiędzy manipulacjami bezpośrednimi i pośrednimi wybranego węzła i pokazują, że algorytm FGA jest w praktyce odporny na manipulacje pośrednie w sieciach rzeczywistych. Rozprawa przestawia zestaw formalnych i eksperymentalnych wyników, które uzasadniają tę konkluzję.

W kontekście zabezpieczeń przed atakami kryptografii rozproszonej, rozprawa wprowadza pojęcie kryptografii indywidualnej (ang. „individual cryptography") — paradygmatu, który chroni przed atakami pozwalającymi na współdzielenie sekretnej informacji potrzebnej do obliczenia zadanego algorytmu. W rozprawie definiujemy formalny model indywidualnej kryptografii i prezentujemy przykładowy schemat w tym modelu - protokół dowodu o wiedzy indywidualnej (ang. „Proof of Individual Knowledge"). Protokół ten pozwala udowodnić, że tajny klucz jest przechowywany fizycznie na jednym urządzeniu. Dodatkowo prezentujemy zastosowanie naszego protokołu w zabezpieczeniach systemów elektronicznego głosowania wdrożonych na blockchainach.

Dodatkowo rozprawa prezentuje wyniki w kontekście bezpieczeństwa urządzeń wykorzystywanych przez użytkowników technologii blockchain. Rozprawa zawiera opis i formalny dowód bezpieczeństwa kompilatora chroniącego przed atakami na specyfikację połączeń w dostarczanym urządzeniu (ang. „wire-tampering attacks"). Ataki te pozwalają nieuczciwemu producentowi lub posiadaczowi urządzenia manipulować specyfikacją połączeń urządzenia, w sposób niezgodny z zaleceniami producenta urządzenia. Nasze rozwiązanie oferuje dowodliwe gwarancje bezpieczeństwa, nawet w przypadku modyfikacji dowolnej liczby przewodów w urządzeniu.

Wyniki przedstawione w rozprawie wskazują na konieczność wyjścia poza schemat badania podstawowej architektury technologii blockchain w kontekście badania bezpieczeństwa systemów opartych na technologii blockchain. Tym samym, rozprawa podkreśla znaczenie analizy *różnorodnych* aspektów bezpieczeństwa środowiska aplikacyjnego i sprzętowego technologii blockchain.

Acknowledgments

I am thankful to my Supervisors, Stefan and Tomasz, for giving me an outstanding opportunity to go (a bit) deeper into the world of science.

I am thankful to my beloved Wife with whom we dedicated significant effort to create mutual perspective and vision for our life.

I am thankful to my Parents who love me even more every day, independently of my successes and failures.

I am thankful to my Brother and my Sister for being with me despite all the struggles of our lives.

I am thankful to my Friends who offered me enormous support during the PhD adventure. In particular, I am thankful to Piotr, Agnieszka, Joanna, Rafał, Katarzyna, Małgorzata, Jan, Mateusz, Kamil, Sławek.

I am thankful to all my Colleagues and Collaborators, who brought color and knowledge to my head. I would like to thank both the Senior Collaborators - Krzysztof Pietrzak, Sebastian Faust, Zeta Avarikioti and Paul Harrenstein, as well as all the People in da house who would see me (online) almost every day - Michelle, Gosia, Ahad, Paweł, Vincenzo, Robert, Shahriar, Parisa, Dominik.

My work was supported by the Polish National Science Centre grant *Blockchain Wallets – Cryptographic Theory and Applications*, no. 2019/35/B/ST6/04138 and the 2016/1/4 project carried out within the Team program of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund.

Contents

1 Introduction **4**

2 On Manipulating Weight Predictions in Signed Weighted Networks **8**
- 2.1 The Manipulability of the FGA . . . 8
- 2.2 Preliminaries . . . 10
- 2.3 Axiomatization . . . 12
  - 2.3.1 Goodness axiomatization . . . 13
  - 2.3.2 Fairness axiomatization . . . 16
  - 2.3.3 FGA axiomatization . . . 18
- 2.4 Complexity of attack . . . 18
- 2.5 Bounding the strength of direct and indirect attacks . . . 25
- 2.6 Simulations . . . 28
- 2.7 Better heuristic for indirect attacks . . . 30
- 2.8 Datasets' basic statistics . . . 31

3 Individual Cryptography **33**
- 3.1 Preventing Secret Holder from Collusions . . . 33
  - 3.1.1 Informal description of our model . . . 35
  - 3.1.2 Informal description of PIK . . . 37
  - 3.1.3 Related work . . . 38
- 3.2 Preliminaries . . . 39
- 3.3 The model . . . 40
- 3.4 Proofs of Individual Knowledge . . . 41
  - 3.4.1 Definition . . . 41
  - 3.4.2 Construction . . . 42
  - 3.4.3 Proof of Thm. 14 . . . 44
  - 3.4.4 Extensions . . . 50
- 3.5 Application: Voting on Blockchain . . . 52

4 Efficiently Testable Circuits without Conductivity **55**
- 4.1 Testability of Circuits . . . 56
- 4.2 Our Contribution . . . 58
  - 4.2.1 More Related Work . . . 60
- 4.3 Preliminaries . . . 60
  - 4.3.1 Notation for Circuits . . . 60
  - 4.3.2 Tampering Model . . . 61

4.4 Gate Covering Sets 62
4.5 Information Loss in Gate-Covered Circuits 63
4.5.1 Routing the Information Loss in Gate-Covered Circuits 66
4.6 Minimizing the Number of External Wires 67
4.7 The Boosted Compression Gadget 68
4.7.1 Instantiating the Building Blocks 69
4.7.2 Algebraic Notation for Circuit Computations 70
4.7.3 Information Losing Tuples 72
4.8 Propagation of the Information Loss in the Boosted Gadget 72
4.9 The Complete Construction 75

5 Final Conclusions and Outlook **78**

Chapter 1

Introduction

Blockchain is a decentralized public ledger. The term "public ledger" can be understood as a "tamperproof sequence of data that can be read and augmented by everyone" [94]. The term "decentralization" refers to the implementation of the ledger that shares information among multiple nodes, without a need to trust a single central unit. The outburst of interest in blockchains can be identified with the start of Bitcoin in January 2009 which was the very first implementation of a blockchain [46]. Since the release of Bitcoin, the technology reached a broad interest in the industry and the research communities, which are actively striving to gain a comprehensive understanding of it. The blockchain community quickly discovered that blockchains are a natural habitat for numerous applications that reach far beyond the standard use of transferring financial assets. This led to the construction of novel *information systems*. The diversity of these extensions is vast, ranging from light-hearted ones like blockchain-based game CryptoKitties [38] to the programs enhancing critical functions of blockchains like Payment Channel Networks (PCNs) [89] and web platforms built to enhance trading of blockchain assets like Bitcoin OTC [4].

The security of systems built upon blockchains inherently relies on their underlying *architecture designs*. To this day, the architectures of blockchains are very often presented in an informal form of whitepapers, with Bitcoin [97] being the primary example. Bitcoin was built upon a Proof-of-Work concept, which assumes that new blocks of data are appended to the ledger with an attached proof that a sufficient amount of work was done upon creation of the blocks. The academic community naturally initiated a more formal study of the blockchain architectures, proposing, inter alia, the Proof-of-Stake [94] and Proof-of-Space [49] blockchains. Both of these architectures aim to lower wasted resources compared to the energy-consuming Bitcoin. The architectures of blockchains usually require a cryptographically secure foundation and a financial mechanism that incentivizes its users to contribute to the platform. In this context, the users of blockchains are granted special roles and rewarded once they accomplish tasks assigned to them [97, 94]. For example, Bitcoin miners are rewarded for sharing a single block with a valid Proof-of-Work.

Despite the mere fact that blockchains are built upon *provably* or *seemingly* secure cryptographic and financial architectures, to believe that the assets held by blockchains and systems built upon blockchains are *indeed safe* because they make use of these architectures, is a rather naïve approach. A vast amount of results (i.a. [108, 45, 93, 59, 104,

36, 58]) successfully improving or attacking the blockchain technology and systems built upon blockchains from *different* angles quickly proved that the holistic study of different *aspects* of the security of systems based on blockchain technology is mandatory. We start with two straightforward examples that illustrate how important is the broader context in the discussion of the security of systems built upon blockchains. The first example involves secret storage. Usually, blockchain-based systems assume that cryptographic secrets used by the blockchain users (e.g., to authorize transactions with cryptographic signatures) are kept "safely". But in the era of countless attacks on popular operating systems [1], saving a secret (that protects high-stake assets) on a personal computer does not seem to be reasonable. Given this perspective, a line of works on so-called blockchain wallets [45] was initiated to ensure safe storage and usage of the blockchain secrets. The second example involves the order of transactions on blockchains. The architectures of systems based on Bitcoin (all other Proof-of-Work blockchains) often assume that publicly announced transactions will take (at most) a fixed amount of time to be put on the blockchain. But it turns out that, as the Bitcoin miners are incentivized to mine the most rewarding transactions first, it is easy to prevent a selected transaction from being put onchain [104, 36] simply by posting other transactions with a higher miner's fee. For this reason, the researchers make an attempt to find definitions and implementations of *fair* miner's fee mechanisms [104, 36].

In this thesis, we adopt the existing models for blockchain architectures and continue the effort to analyze various aspects of the security of systems built upon blockchains. In particular, we focus on the following research directions:

- We conduct a study of the *reliability of rating systems used by blockchain trading platforms.*
- We give a formal study of *algorithms that ensure security against distributed cryptography attacks.*
- We propose a *compiler for devices that ensures security against wire-tampering attacks.*

In the remainder of this chapter, we will summarise our contributions.

Rating systems used by blockchain trading platforms. As already mentioned, blockchains' primary application is to store and trade digital assets, whether they are financial or otherwise. These assets are specified and managed by various mechanisms, such as UTXOs on Bitcoin or Non-Fungible Tokens (NFTs) on Ethereum. The asset exchange process is rather complicated and requires advanced technical knowledge, therefore average traders may become easy targets for scam trading operations. Consequently, with the advancement of blockchains, a whole ecosystem of web platforms has emerged to facilitate the trading of digital assets. These platforms put a lot of effort into providing mechanisms that aim to increase the security of a trader. Some potential risks, like unexpected exits from transactions or artificial pumping-up of the assets prices [96] are hard to identify using automated methods, but can be mitigated with a user-ranking system maintained by users with verified identity. To this end, in our work, we attempt to understand *how reliable the rating systems are in the context of blockchain trading platforms.* In particular,

in Chapter 2, we analyze the problem of manipulating the Fairness-Goodness-Algorithm (FGA) proposed by Kumar et al. [84] which is well known in the literature. *FGA* is a measure of the mutual trust of nodes in a graph with weighted ratings. The measure is defined recursively. Each of the nodes is assigned a fairness value measuring how fair is the node with rating other nodes, and a goodness value measuring how much the node is trusted by other nodes. Given that the primary application of *FGA* is to fairly and accurately evaluate nodes in a transaction network, such as many blockchain networks, it is crucial to understand this measure's properties and susceptibility to manipulations. Unfortunately, the authors of *FGA* proved only a very limited number of its axiomatic properties. Also, little is known about the robustness of this measure against any manipulation efforts. In our research, we thus provide a full axiomatization of the *FGA*, formally define the problem of manipulability of the *FGA*, and study how the *FGA* could be manipulated in some practical scenarios.

Distributed cryptography attacks. Distributed cryptography allows parties to jointly compute some function of their inputs without revealing their inputs to other parties [11]. It can be realized using multiparty-computation protocols, fully-homomorphic encryption, or trusted execution environments. Our key observation is that distributed cryptography may be *maliciously* used to share or divide knowledge of secrets used to identify users of information systems (e.g., on the blockchain), potentially blocking the possibility of punishing a user that leaks, sells, or leases [101] its private information to other users. In Chapter 3, we introduce individual cryptography - *a formal study of algorithms that ensure security against distributed cryptography attacks*. Individual cryptography ensures that secret information needed to compute an algorithm is stored fully on a single machine. This concept holds potential for numerous applications, including incentivizing participants of protocols not to disclose their secrets to other users. For concreteness, we additionally present a Proof of Individual Knowledge protocol that forces the users to compute extensive computation on the secret in a limited time. The protocol gives the service provider provable guarantees that the secret is not shared or divided between a set of parties.

To show how the Proof of Individual Knowledge may be applied to systems built upon blockchains, we present a concept of security extension for voting systems within Decentralized Autonomous Organizations (DAOs) [102]. DAOs are programs implemented on blockchains designed to transparently and autonomously manage organisations. A fundamental component of these organizations is a voting mechanism enabling participants to conduct elections or referendums periodically. Given that each DAO user is typically associated with a secret key, the distributed cryptography can be leveraged to initially share knowledge about the secret, in order to lease [101] control over the secret for a limited time. This, in turn, facilitates processes such as vote selling. We show how individual cryptography may be used to prevent such attacks. For a more detailed description of this attack, please refer to Section 3.5.

Security of devices used by blockchain users. The last aspect of the security of the blockchain technologies that we consider is the *security of devices used to store and manage secret information needed to protect digital assets*. In the era of digital assets, the notion of possession refers to the knowledge of secret information used to bind the asset.

This approach sets a new level of risk, as stealing a digital asset can be easier than ever if we consider all the modern ways to hack devices used to store and manage secrets.

The blockchain wallet is one example of a device specifically designed to store the blockchain cryptographic information. The research community pays much attention to the security of these devices, e.g. by trying to formally understand the implications of so-called cold-hot wallets [45], or hierarchical deterministic wallets [117], as well as postquantum secure cold-hot wallets [73]. Another approach that can be taken is the study of theoretical aspects of the security of devices modeled formally as arithmetic circuits. A seminal series of papers introduces various theoretical security aspects of devices [76, 74, 52], including the problem of leaking secrets via side channels, the problem of device tampering, or the problem of secure outsourcing of device production.

Following the theoretical line of work, we try to derive security in a setting similar to the one from [74] that assumes a security model that assumes tampering with the specification of the device. In this context, in Chapter 4 we provide a *full construction of compiler that compiles circuits into circuits secure against so-called wire-tampering attacks*. The production of electronic circuits is handled via a sequence of independent steps. It usually starts with the design of the circuit and then the production of prefabricates needed to produce the circuit (e.g., the circuit board); in the main stage, the gates and the wires are applied to the circuit, and finally, the construction is tested. During the application of the wire topology to the circuit, benign (non-adversarial errors) may occur that cause computation errors [19, 27]. To counter this problem, the hardware testing community has applied numerous heuristic techniques [19, 27] to test the input/output behavior of the circuit. Our compiler is a method to test such circuits with provable guarantees efficiently. Our testing method is secure even against *adversarially* chosen errors and against an unbounded number of wire tamperings applied to the circuit. Finally, assuming that other amendments to the implementation of the circuit (e.g., the tampering of the gates) can be visually inspected, our tool can be used to protect against adversarial device tampering in practice.

Publications

The following publications were used as a foundation for this thesis.

- Chapter 2 is based on the work "On Manipulating Weight Predictions in Signed Weighted Networks" published in the proceedings of AAAI-23 [90]. All of the technical results in this chapter are my own technical contribution.

- Chapter 3 is based on the work "Individual Cryptography" published in the proceedings of CRYPTO-23 [51]. The main technical concepts presented in the chapter came from the other authors, whereas my involvement contributed to the finalization of the formalizations and formal proofs presented in the chapter.

- Chapter 4 is based on the introduction and the section 5 of the work "Efficiently Testable Circuits without Conductivity" published in the proceedings of TCC-23 [9]. All of the technical results in this chapter are my own technical contribution unless explicitly stated in the text.

Chapter 2

On Manipulating Weight Predictions in Signed Weighted Networks

Adversarial social network analysis studies how graphs can be rewired or otherwise manipulated to evade network analysis tools. While there is ample literature on manipulating fundamental network analysis tools, more sophisticated tools are much less understood in this respect. In this chapter, we focus on the problem of evading *FGA*—an edge weight prediction method for signed weighted networks by [84]. Among others, this method can be used for trust prediction in reputation systems, and was tested on real-life datasets derived from blockchain trading platforms (Bitcoin OTC, Bitcoin Alpha) [84]. We study the theoretical underpinnings of *FGA* and its computational properties in terms of manipulability. Our positive finding is that, unlike many other tools, this measure is not only difficult to manipulate optimally, but is also difficult manipulate in practice.

2.1 The Manipulability of the FGA

Many works in the body of research on adversarial social network analysis have considered how to manipulate classic tools of social network analysis such as centrality measures [37, 17, 115], community detection algorithms [112, 64, 33], and link and sign prediction algorithms [67, 66]. Also, a rapidly growing body of works studies adversarial learning on graphs using deep learning [34].

While most of the above literature focused on simple networks, in this chapter, we consider a more complex model of weighted signed networks. In this class of networks, links are labeled with real-valued weights representing positive or negative relations between the nodes [86, 87, 107]. An important application of signed weighted networks is the modelling of trust networks/reputation systems, the goal of which is to avoid transaction risk by providing feedback data about the trustworthiness of a potential business partner [103]. As an example, let us consider the cryptocurrency trading platform Bitcoin OTC [84]. In this platform, users are allowed to rate their business partners on the scale $\{-10, -9, \ldots, 10\}$, and the ratings are publicly available in the form of a who-trusts-whom network. A 6-node fragment of this network is presented in Figure 2.1.

A user who thinks of doing a transaction with another user for the first time can use the information from such a who-trust-whom network to predict the potential risk. Technically, given a trust network modeled as a weighted signed network, predicting trust

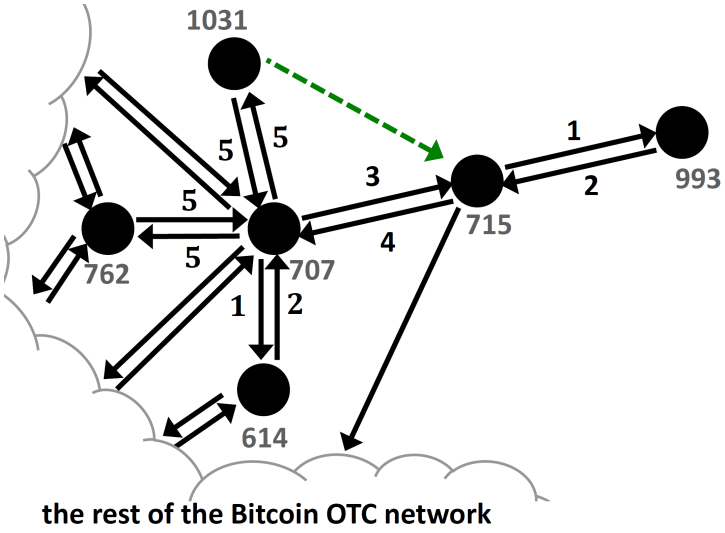


Figure 2.1: A fragment of the Bitcoin OTC network composed of nodes 993, 715, 707, 614, 1031, 762.

amounts to predicting the weights of potential new edges. A well-known edge weight prediction method, called *FGA*, was proposed by [84]. *FGA* is based on two measures of node behavior: the **goodness** that evaluates how much other nodes trust a given node, and the **fairness** that captures how fair this node is in rating other nodes. Both concepts have a mutually recursive definition that converges to a unique solution. Most importantly, Kumar et al. showed that *FGA* is effective in predicting edge weights, i.e., the level of trust between unlinked nodes. For example, in Figure 2.1, the trust of node 1031 towards node 715 is predicted by *FGA* to be 2.26.

While *FGA* seems to be an interesting tool to apply in practice, little is known about its resilience to malicious behaviour. In this chapter, we present the first study of manipulating the *FGA* function by a *rating fraud* [28, 92]. It involves fraudulent raters to strategically underrate or overrate other users for their own benefit. To magnify the strength of the manipulation, the attacker may create and act via multiple fake user identities. Such so called *Sybil attacks* are especially tempting in environments such as cryptocurrency trading platforms where creating a new identity is affordable. Rating fraud attacks may be *direct*—when targeted nodes are rated directly by the attackers—and *indirect*—when the attackers try to manipulate the neighbourhood of the target nodes rather than the target nodes themselves (see Figure 2). It is important to distinguish between direct and indirect manipulations, as in some situations, only indirect ones will be practical. This may be the case on e-commerce platforms such as e-Bay or trading platforms for crypto currencies, where nodes rate each other only after completing a transaction. When a retailer of expensive products is the target, the cost of a direct attack can be prohibitive. Hence, an indirect attack becomes an attractive alternative—it may be much cheaper to attack through the clients or business partners of such an expensive retailer (see the next section for an example).

Our contributions can be summarised as follows:

- To analyze the theoretical underpinnings of the *FGA* measure, we propose the system of basic axioms for both fairness and goodness. We prove that together they

uniquely determine the *FGA* measure;

- Next, we formulate the issue of manipulating the *FGA* measures of some target group of nodes as a set of computational problems. We then prove that all these problems are *NP*-hard and $W[2]$-hard, i.e., *FGA* is, in general, hard to manipulate.

- Given the hardness of attacking a group of nodes, we then focus our analysis on targeting a single node - directly or indirectly. As for an indirect attack, we show analytically that for some class of networks (which we call *minimum-k-neighbour graphs*, since we require that every node in this network has *indegree* and *outdegree* at least k), we can bound the strength of indirect attacks. A similar result for the direct attack gives a bound k times weaker than for the indirect case. Our positive finding is that, in this case, *FGA* measure turns out to be rather difficult to manipulate indirectly.

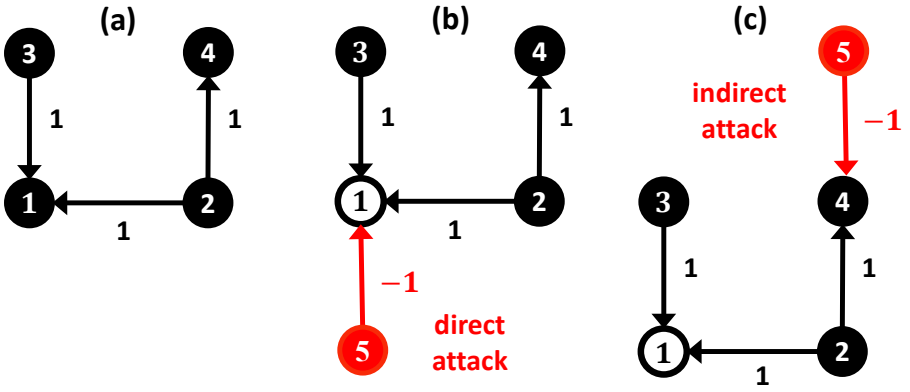
- In our experimental analysis, we first evaluate two benchmarks: (a) the strength of the aforementioned direct attack, and (b) the strength of an indirect attack based on a simple greedy approach. The latter one turns out to be very ineffective. Next, we analyse an improved greedy approach by attacking at a larger scale in every step. This approach, although costly, proves to be sometimes effective.

2.2 Preliminaries

A Weighted Signed Network (WSN) is a directed, weighted graph $G = (V, E, W)$, where V is a set of users, $E \subseteq V \times V$ is a set of (directed) edges, and $\omega : E \to [1, +1]$ is a weight function that to each $(u, v) \in E$ assigns a value between -1 and $+1$ that represents how u rates v. For any directed edge $(u, v) \in E$, let us denote by $\overline{(u, v)}$ the edge in the opposite direction, i.e., $\overline{(u, v)} = (v, u)$. For any set of directed edges E, denote by $\overline{E} = \{\overline{e} : e \in E\}$. Furthermore, let P be a set of pairs of nodes of cardinality n, i.e, $P = \{\{u_1, v_1\}, ..., \{u_n, v_n\}\}$. The domain of P is the set of nodes that make the pairs in P, i.e. $dom(P) = \{u : u \in \{u, v\} \in P\}$. Finally, we write $Pred(v)$ (resp. $Succ(v)$) to denote the set of *predecessors* (resp. *successors*) of v (resp. u) defined as follows: $Pred(v) = \{u : u \in (u, v) \in E\}$ (resp. $Succ(u) = \{v : v \in (u, v) \in E\}$).

By $[k]$, we denote a set $\{1, \ldots, k\}$. For a square matrix $M^{m \times m}$ of size m, we define $||M||_\infty = \max_{1 \leqslant i \leqslant m} \sum_{j=1}^{m} m_{ij}$, $||M||_1 = \max_{1 \leqslant j \leqslant m} \sum_{i=1}^{m} m_{ij}$. It is also known that $||M \times M||_\infty \leqslant ||M||_\infty \cdot ||M||_1$ and $||M \times M||_\infty \leqslant ||M||_1 \cdot ||M||_1$ (see `https://en.wikipedia.org/wiki/Matrix_norm`).

Kumar et al. (2016) define a recursive function, *FGA*, that assigns to each vertex of a weighted directed graph two values: *fairness* and *goodness*, $(f(v), g(v))$. The first one, $f(v)$, assigns a real value from range $[0, 1]$ to v that indicates how *fair* this node is in rating other nodes. The second one, $g(v)$, assigns a value from range $[-r, r]$ to v indicating how much *trusted* this node is by other nodes (for simplicity we assume that $r = 1$ in our work). We define an in-degree ($indeg(u)$) and out-degree ($outdeg(u)$) of a node $u \in V$. $indeg(u) = |\{(v, u) : (v, u) \in E\}|$ and $outdeg(u) = |\{(u, v) : (u, v) \in E\}|$. Kumar et al.'s



Network	(a)		(b)		(c)	
node v	$g(v)$	$f(v)$	$g(v)$	$f(v)$	$g(v)$	$f(v)$
1	1	1	**0.4**	1	**0.83**	1
2	1	1	1	0.8	1	0.75
3	1	1	1	0.7	1	0.92
4	1	1	0.8	1	0.17	1
5			1	0.3	1	0.42

Figure 2.2: Sample networks and two types of attacks.

recursive formula for $(f(v), g(v))$ is as follows:

$$g(v) = \frac{1}{indeg(v)} \sum_{u \in Pred(v)} f(u) \cdot \omega(u, v) \tag{2.1}$$

$$f(u) = 1 - \frac{1}{outdeg(u)} \sum_{v \in Succ(u)} \frac{|\omega(u, v) - g(v)|}{2}, \tag{2.2}$$

where $g(v) = 1$ for $v \in V$ with $indeg(v) = 0$, and $f(v) = 1$ for $v \in V$ with $outdeg(v) = 0$.

Kumar et al. (2016) showed that this function can be computed iteratively starting from $f^{(0)}(u) = g^{(0)}(u) = 1$. Theorem 1 from the aforementioned work states that at each step, t, the estimated values $f^{(t)}(u)$, $g^{(t)}(u)$ get closer to their limits $f^{(\infty)}(u)$, $g^{(\infty)}(u)$ and one gets $|f^{(\infty)}(u) - f^{(t)}(u)| < \frac{1}{2^t}$ and $|g^{(\infty)}(u) - g^{(t)}(u)| < \frac{1}{2^{t-1}}$. The *FGA* function can be used for predicting the weight of some not-yet existing (or unknown) edge $(u, v) \in V \times V \setminus E$ by computing the product: $\omega(u, v) = f(u) \cdot g(v)$.

As an example of the *FGA* function and how it could be attacked, let us consider Figure 2.2. Network (a) is a benchmark, where every node rates others with the highest possible value. In network (b), a new node 5 is used to perform a **direct attack** by rating node 1 with the worst possible value of -1. This decreases the goodness of node 1 to 0.4. However, as argued in the introduction such a direct attack can be prohibitively costly. Nevertheless, given the definition of the *FGA* function, node 5 can also perform an indirect attack on node 1. This can be done, for instance, by directly attacking node 4. As node 4 has already been rated positively by node 2, an opposite rating introduced by 5 will decrease the fairness of 2. In particular, comparing network (c) to (a) in Figure 2.2, the fairness of 2 decreased from 1 to 0.75. This lower fairness means that node's 2 ratings

are less meaningful in network (c) than in network (a). Hence, the goodness of node 1 decreases to 0.83.

2.3 Axiomatization

Our first result is an axiom system that completely characterizes the *FGA*. First, we present a comprehensive summary, while the details will be given afterwards.

We begin with the characterization of the goodness part of the *FGA* function. Recall that the idea behind the goodness of v is that it should reflect how this node is rated by its predecessors. Moreover, the ratings of the fairer predecessors should count more. We translate these high-level requirements into the following axioms:

- Smooth Goodness — let all predecessors of a particular node, $v \in V$, be unanimous in how they rate v and let their fairness be the same. Now, let us assume that their fairness increases equally, i.e., intuitively, the nodes that rate v become more trustworthy. Then, we require that this will result in an increase of the goodness of v, and that this increase is proportional to the increase of the fairness of v's predecessors.

- Increase Weight — let the predecessors of v be all equally fair and unanimous in how they rate v. Now, let them increase their rating of v equally. Then, we require that the goodness of v increases and that this increase is proportional to the increase in how v is rated.

- Monotonicity for Goodness — the predecessors with higher fairness should have a bigger impact on the goodness of v (similarly for the predecessors with higher weights).

- Groups for Goodness — let v be rated by k groups of the predecessors and let the nodes in each group be homogeneous and unanimous w.r.t. v. What is then the relationship between the impact these groups have on the goodness of v? In line with the previous axioms, we require that the goodness of v should be equal to the *weighted average* of the ratings achieved when these groups separately rate v.

- Maximal Trust — this basic condition requires that if all the predecessors of v have the highest possible fairness and their ratings are the highest possible, then the goodness of v should be the highest possible.

- Baseline for Goodness — a non-rated node has the goodness of 1.

Our first result is that the above axioms uniquely define the goodness part of the *FGA* function.

Let us now characterise the fairness part of the *FGA* function. Recall that the idea behind the fairness of v is that it should reflect how the ratings given by this node agree with the ratings given by other nodes, i.e. how erroneous v is. In this respect, we have the following axioms:

- Smooth Fairness — this axiom stipulates that the fairness of a node making an average error is an average of the fairness values of nodes making extreme errors.

- Monotonicity for Fairness — this axiom stipulates that the fairness of a node that rates more accurately than before should rise.
- Groups for Fairness — if the nodes rated by v can be divided into k groups such that each node in a particular group is rated by v in the same way, then the fairness of v should be equal to the *weighted average* of v's fairness in a setting where v rates these groups separately.
- Obvious Fairness Metric — Here, we stipulate that when a node makes maximal errors when rating all of its neighbors, its fairness should be 0, and when there is no error, the fairness is 1.
- Baseline for Fairness — the fairness of a node that rates no one is 1.

The above axioms uniquely define the fairness part of the *FGA* function. In summary, all the above axioms uniquely define the *FGA* function.

Theorem 1. *The Smooth Goodness, Increase Weight, Monotonicity for Goodness, Maximal Trust, Groups for Goodness, Baseline for Goodness axioms, and the Smooth Fairness, Monotonicity for Fairness, Obvious Fairness Metric, Groups for Fairness, and Baseline for Fairness axioms uniquely define the FGA function.*

2.3.1 Goodness axiomatization

Below we formally define the axioms and present the uniqueness proofs. We begin with the characterization of goodness part of the *FGA* function. Let $v \in V$ have all the predecessors $u_i \in \mathit{Pred}(v)$ homogenous and unanimous w.r.t. v, i.e. they all have the same fairness f_0 and they rate v with the same rating ω_0. Now, let us assume that f_0 of all the predecessors gets increased by the same amount, Δ. We require that the goodness of the rated node v should rise proportionally to Δ (see Figure 2.4). To formalize this axiom, let us denote the goodness of v in such a setting by $g^{\phi_\omega,\phi_f}(v)$, where ϕ_ω indicates the value of weight of the edges (u_i, v), and ϕ_f indicates the value of the fairness of all $u_i \in \mathit{Pred}(v)$.

Axiom 1 (Smooth Goodness)**.** *Let $v \in V$, such that for all $u_i \in \mathit{Pred}(v)$ it holds that $f(u_i) = f_0 \wedge\ \omega(u_i, v) = \omega_0$. Then, for all $\Delta \in \mathbb{R}$:*

$$g^{\omega_0,f_0+\Delta}(v) = g^{\omega_0,f_0}(v) + g^{\omega_0,\Delta}(v).$$

Next, let us consider an analogous situation, but now the weight ω_0 of the edges from the predecessors $\mathit{Pred}(v)$ to v increases by Δ while their fairness f_0 remains the same (see Figure 2.5). This leads to the following axiom:

Axiom 2 (Increase Weight)**.** *Let $v \in V$, such that for all $u_i \in \mathit{Pred}(v)$ it holds that $f(u_i) = f_0 \wedge \omega(u_i, v) = \omega_0$. Then, for all $\Delta \in \mathbb{R}$:*

$$g^{\omega_0+\Delta,f_0}(v) = g^{\omega_0,f_0}(v) + g^{\Delta,f_0}(v).$$

Next, we require that nodes with higher fairness have a higher impact on the goodness of the rated nodes. Similarly, higher weights should result in a better rating of the target node (see Figure 2.6).

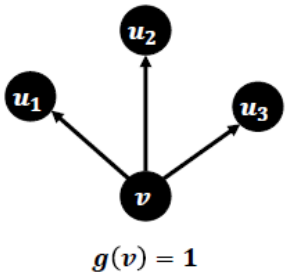


Figure 2.3: Axiom 6 sets a default value of the goodness to 1 for a node that has not been rated yet.

Axiom 3 (Monotonicity for Goodness)**.** *Let u_1 and u_2 be two nodes rated by unanimous and homogeneous sets of predecessors S_1, S_2. S_i's consist of nodes with identical fairness f_i which rate u_i with identical ω_i. If $f_1 = f_2$ and $\omega_1 > \omega_2$, then $g(u_1) \geqslant g(u_2)$. Also, if $\omega_1 = \omega_2$ and $f_1 > f_2$, then $g(u_1) \geqslant g(u_2)$.*

Monotonicity for Goodness is an analogy for the Goodness Axiom proposed by Kumar et al. (2016). While the Goodness Axiom concerns any set of predecessors, Monotonicity for Goodness focuses on unanimous and homogeneous sets of them.

Next, any node $v \in V$, that has the best possible rating given by each of its predecessors and all its predecessors have the highest possible fairness, should have maximal possible goodness (see Figure 2.6).

Axiom 4 (Maximal Trust)**.** *For any $v \in V$, such that for all $u_i \in Pred(v)$ it holds that $f(u_i) = 1 \ \wedge \ \omega(u_i, v) = 1$, for all $\Delta \in \mathbb{R}$ the goodness value of the node v is fixed to 1, i.e. $g(v) = 1$.*

The following axiom states that, when $v \in V$ is rated by k groups, where the nodes in each group are homogeneous and unanimous w.r.t. v, then the goodness of v should be equal to the *weighted average* of the ratings achieved when these groups separately rate v.

Axiom 5 (Groups for Goodness)**.** *Given $v \in V$, let $\{S_1, \ldots, S_k\}$ be a partition of $Pred(v)$, such that for all $i \in [k]$ there exists f_i, ω_i such that for all $u_j \in S_i$ it holds that $f(u_j) = f_i \wedge \omega(u_j, v) = \omega_i$. Then, it holds:*

$$g(v) = \frac{\sum_{i \in [k]} (|S_i| \cdot g_i(v))}{\sum_{i \in [k]} |S_i|},$$

where $g_i(v)$ denotes the rating of the node v rated only by the homogeneous and unanimous predecessors from group i.

Finally, we have the following baseline:

Axiom 6 (Baseline for Goodness)**.** *Any $v \in V$ with $indeg(v) = 0$ has $g(v) = 1$.*

We will now show that the above axioms uniquely define the goodness part of the *FGA* function.

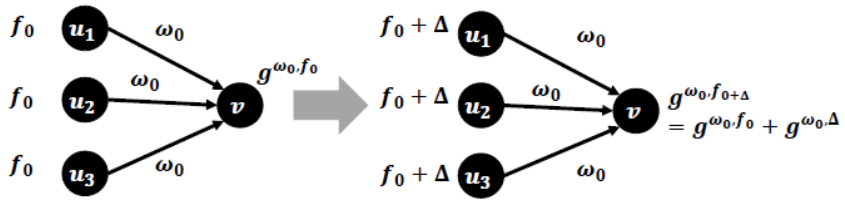


Figure 2.4: Axiom 1 says that the increase in the homogenous *fairness* of the unanimous predecessors results in the proportional increase of the *goodness* of the rated node.

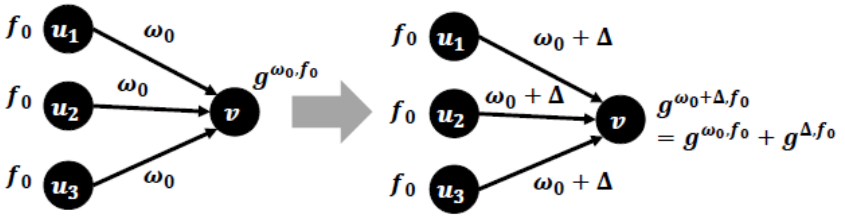


Figure 2.5: Axiom 2 says that the increase in the homogenous *weight* of the unanimous predecessors results in the proportional increase of the *goodness* of the rated node.

Theorem 2. *For any fixed fairness function $f(u)$, the Smooth Goodness, Increase Weight, Monotonicity for Goodness, Maximal Trust, Groups for Goodness, and Baseline for Goodness axioms uniquely define goodness function* (2.1).

Proof. It is easy that the goodness function (2.1) meets the conditions of the above axioms.

Now, we show that the axioms imply the goodness function (2.1). Let us define some $g_i^{w_0,f_0}(v)$ for a node v rated by homogeneous and unanimous nodes w.r.t. v. In other words, all $u \in Pred(v)$ have the same *fairness* $f(u) = f_0$ and they rate v with the same rating $\omega(u,v) = \omega_0$. From Smooth Goodness and Monotonicity for Goodness and the Cauchy's equation [105], we know that $g_i(v)$ is linearly dependant on $f(u)$ when $\omega(u,v)$ is fixed to some ω_0, i.e. for some constant $a_{\omega_0} \in \mathbb{R}$:

$$g_i^{\omega_0,f(u)}(v) = a_{\omega_0} \cdot f(u).$$

Again, from Increase Weight, Monotonicity for Goodness, and the Cauchy's equation we know that $g_i(v)$ is linearly dependant on $\omega(u,v)$ when $f(u)$ is fixed to some f_0, i.e. for some constant $b_{f_0} \in \mathbb{R}$:

$$g_i^{\omega(u,v),f_0}(v) = \omega(u,v) \cdot b_{f_0}.$$

The two equations above imply that for a set of homogeneous and unanimous predecessors, $g_i^{\omega(u,v),f(u)}(v) = g_i^{\omega,f}(v) = a_\omega \cdot f = b_f \cdot \omega$. Since the function $g_i^{\omega(u,v),f(u)}(v)$ is defined for all $\omega, f \in \mathbb{R}$, this equality implies that for any $f \neq 0$ it holds that $a_\omega = \frac{b_f}{f} \cdot \omega$. Furthermore, since a_ω is not dependant on f by definition, then $a_\omega = c \cdot \omega$ for some $c \in \mathbb{R}$. We conclude that for all $\omega, f \in \mathbb{R}$:

$$g_i^{\omega,f}(v) = c \cdot f \cdot \omega.$$

From Maximal Trust, we get that $g^i(v) = f(u) \cdot \omega(u,v)$. Now, when a node does not have unified predecessors, we can divide its predecessors into groups with fixed (f_i, ω_i).

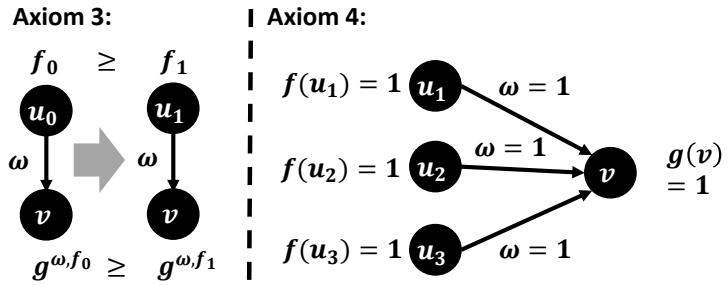


Figure 2.6: Axiom 3 says that fairer nodes should have a bigger impact on the goodness of the rated nodes. In the figure, we assume that $\omega > 0$ and node u_0 has bigger fairness than node u_1. Axiom 4 says that a fully trusted node should have the goodness value equal to 1.

From Groups for Goodness, we get:

$$g(v) = \frac{\sum_{i\in[k]}(|S_i| \cdot g_i(v))}{\sum_{i\in[k]}|S_i|} = \frac{\sum_{i\in[k]}(|S_i| \cdot f_i \cdot \omega_i)}{\sum_{i\in[k]}|S_i|} =$$

$$= \frac{1}{in(v)} \sum_{u\in Pred(v)} f(u) \cdot \omega(u, v).$$

Additionally, from Baseline for Goodness, $g(v) = 1$ for v with $indeg(v) = 0$. □

2.3.2 Fairness axiomatization

In this section, we present the axiomatization of the fairness part of the *FGA* function. The fairness axiomatization is defined with respect to the rating error of nodes. We define the error of node v rating the node u as $d = |\omega(v, u) - g(u)|$.

Our first axiom stipulates that the fairness of a node that makes an average error when rating other nodes is equal to the average of the fairness values of nodes in extreme cases.

Axiom 7 (Smooth Fairness)**.** *Assume a node v rates a set of its successors S with equal error $d = |g(u) - \omega(v, u)|$ for $u \in S$ in one setting, and with an error D in another setting, then $f^{\frac{d+D}{2}}(v) = \frac{f^d(v)+f^D(v)}{2}$.*

The following axiom states that fairness of the nodes that rate more accurately should rise (see Figure 2.7).

Axiom 8 (Monotonicity for Fairness)**.** *Let u_1 and u_2 be two nodes rating their sets of successors S_1, S_2. S_i consists of nodes v_i rated by u_i with identical error $d_i = |g(u_i) - \omega(u_i, v_i)|$. If $d_1 > d_2$, then $f(u_1) \leqslant f(u_2)$.*

The Monotonicity for Fairness axiom is an analogy for the Fairness Axiom in [84]. While the Fairness Axiom is defined for any set of predecessors, in our case, the Monotonicity for Fairness is defined only for a set of successors S_i rated with *equal* rate by the node u_i.

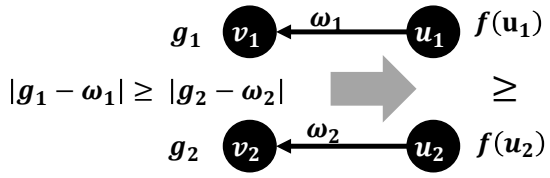


Figure 2.7: Axiom 8 says that the fairness of a node should rise when the node gives more precise ratings.

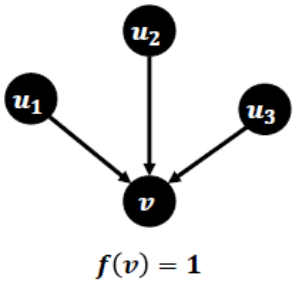


Figure 2.8: Axiom 11 says that a node that has not rated anyone yet should have fairness 1.

Next, we stipulate that when a node makes maximal errors when rating all of its neighbors, then its fairness should be 0, and when it always agrees with the actual goodness value of its rated nodes, then its fairness is 1 (see Figure 2.9).

Axiom 9 (Obvious Fairness Metric)**.** *Assume node v rates all its successor nodes S with distance $d = |g(u) - \omega(v,u)| = 0$, for $u \in S$, then $f(v) = 1$. Assume a node v rates all its successor nodes S with distance $d = |g(u) - \omega(v,u)| = 2$, for $u \in S$, then $f(v) = 0$.*

Also, when $v \in V$ rates its neighbors that can be divided to k such groups that each node in a group is rated by v with the same distance as other nodes in this group, then the fairness of v should be equal to the *weighted average* of its fairness in a setting where v rates these groups separately.

Axiom 10 (Groups for Fairness)**.** *Given $v \in V$, let $\{S_1, \ldots, S_k\}$ be a partition of $Succ(v)$ such that $\forall i \in [k]$ there exists $d_i : \forall_{u_j \in S_i} |g(u_j) - \omega(v,u_j)| = d_i$. Then:*

$$f(v) = \frac{\sum_{i \in [k]} (|S_i| \cdot f_i(v))}{\sum_{i \in [k]} |S_i|},$$

where $f^i(v)$ is the fairness of v rating group i.

Finally, a baseline for node v with $outdeg(v) = 0$ is:

Axiom 11 (Baseline for Fairness)**.** *A node v with $outdeg(v) = 0$ has $f(v) = 1$.*

Theorem 3. *For fixed goodness function $g(n)$, the Smooth Fairness, Monotonicity for Fairness , Obvious Fairness Metric, Groups for Fairness, and Baseline for Fairness axioms uniquely define fairness function* (2.2)*.*

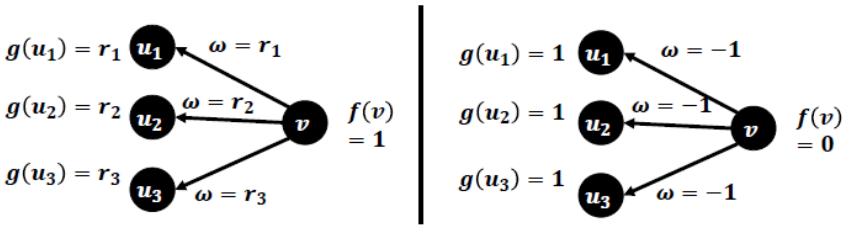


Figure 2.9: Axiom 9 says that a node that rates with perfect accuracy (its rating error $d = 0$ for all the nodes that it rates) should have maximal fairness equal to 1, and a node that makes the biggest errors (its rating error $d = 2$ for all the nodes that it rates) should have minimal fairness equal to 0.

Proof. It is easy that the fairness function (2.2) meets the conditions of the above axioms. Now let us show that the above axioms define the function (2.2). Let us define $f_i(v)$ for a node v and a group of nodes with some fixed error $d_i = |g(u) - \omega(v, u)|$. From Smooth Fairness, Monotonicity for Fairness and the Jensen's equation [105], we know that $f_i(v)$ is linearly dependant on $d_i = |g(u) - \omega(v, u)|$, i.e. for some $a, b \in \mathbb{R}$:

$$f_i(v) = b + a \cdot d_i.$$

From Obvious Fairness Metric we get that $f_i(v) = 1 - d_i/2 = 1 - |g(u) - \omega(v, u)|/2$ for a set of unified successors. Now when a node does not have unified successors, we can divide its successors to groups with fixed $d_i = |g(u_j) - \omega(v, u_j)|$. From Groups for Fairness:

$$f(v) = \frac{\sum_{i\in[k]}(|S_i| \cdot f_i(v))}{\sum_{i\in[k]} |S_i|} = \frac{\sum_{i\in[k]}(|S_i| \cdot (1 - d_i/2))}{\sum_{i\in[k]} |S_i|} =$$

$$= 1 - \frac{1}{out(v)} \sum_{u\in Succ(v)} |g(u) - \omega(v, u)|/2.$$

Finally, from Baseline for Fairness we get that $f(v) = 1$ for nodes with $outdeg(v) = 0$. □

2.3.3 FGA axiomatization

The above results imply the final axiomatization result:

Theorem 1. *The Smooth Goodness, Increase Weight, Monotonicity for Goodness, Maximal Trust, Groups for Goodness, Baseline for Goodness axioms, and the Smooth Fairness, Monotonicity for Fairness, Obvious Fairness Metric, Groups for Fairness, and Baseline for Fairness axioms uniquely define the FGA function.*

Proof. The proof follows from the proofs of Theorems 2 and 3. □

2.4 Complexity of attack

In this section we study the complexity of manipulating *FGA*.

Attack models. Given $G = (V, E, \omega(E))$, let $A \subseteq V$ be a set of attackers. We define two types of the A's objectives:

- **targeting potential links** — here, the target set TP is composed of disconnected pairs of nodes from $V \setminus A$:

$$TP \subseteq \{\{u, v\} : u, v \in V \setminus A \wedge (u, v), (v, u) \notin E\}. \tag{2.3}$$

  Intuitively, the aim is to change the predicted weight of the potential links between the pairs from TP.

- **targeting nodes** — here, the target set is $T \subseteq V \setminus A$. Intuitively, the goal is to alter the targets' reputation.

The attackers can make the following types of moves:

- **edge addition** — the attackers can add an edge (u, v) to G, where $u \in A$, $v \in T$, $(u, v) \notin E$, and with the weight $\omega(u, v) \in [-1, 1]$. This corresponds to the attacker $u \in A$ rating node $v \in T$ for the first time.

- **weight update** — an attacker $u \in A$ can update the weight of an existing edge $(u, v) \in G$ to some value $\omega(u, v) \in [-1, 1]$. This corresponds to a modification of the existing rating by the attacker.

All the attackers are allowed to make no more than k such moves in total. We will refer to k as a budget.[1]

We will now formalize our computational problems. In the first one, the attackers aim at modifying the predicted weights between the pairs of nodes in TP to decrease them below (increase above) a certain threshold. This attack corresponds to breaking potential business connections.

Problem 1 (Decrease (Increase) Mutual Trust, *DMT* (*IMT*))**.** *Given a weighted signed network $G = (V, E, \omega)$, a set of attacking nodes $A \subseteq V$, a target set of pairs of nodes TP as defined in eq. 2.3, an intermediary set $I \subseteq V$, the budget k, and a threshold $t \in [-1, 1]$, decide for all $\{u, v\} \in TP$ whether it is possible to decrease (increase) the value of either predicted weight $f(u) \cdot g(v)$ or $f(v) \cdot g(u)$ to or below (above) the threshold t by making no more than k edge additions or weight updates with the restriction that the attackers $u \in A$ are rating only the nodes from the intermediary set I.*

In the second problem, the attackers aim at altering the goodness value of the nodes from a target set T. This attack corresponds to spoiling the reputation of the target nodes.

Problem 2 (Decrease (Increase) Nodes Rating, *DNR* (*INR*))**.** *Given WSN $G = (V, E, \omega)$, a set of attackers $A \subseteq V$, a target set $T \subseteq V \setminus A$, an intermediary set $I \subseteq V$, the number of possible moves k, and threshold $t \in [-1, 1]$, decide whether it is possible, for all $v \in T$, to decrease (increase) the goodness of each vertex v to or below (above) threshold t by making no more than k edge additions or weight updates with the restriction that the attackers $u \in A$ are rating only the nodes from the intermediary set I.*

[1] We place no constraints on how the attackers distribute this budget among themselves. In an extreme case, a single attacker can do all k actions.

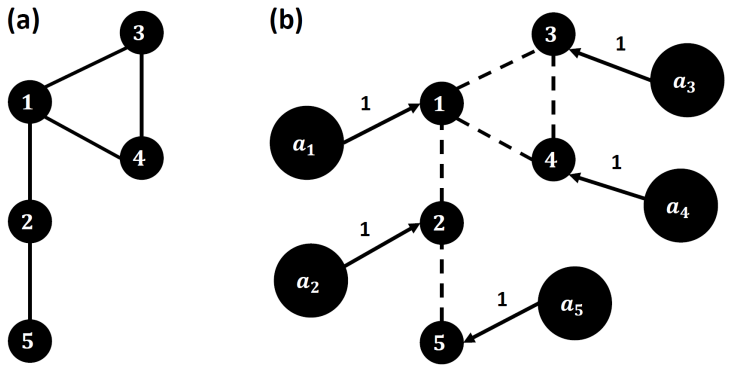


Figure 2.10: (a) The original VC problem ($k = 2$) with vertices $V = \{1, 2, 3, 4, 5\}$ and edges as in the picture. This set can be covered with 2 nodes - 1 and 5. (b) The corresponding *DMT* instance with number of moves $k = 2$, threshold $t = -1$, the set of attacked edges H, as in the original problem, and a new set of attacking nodes $\{a_1, a_2, a_3, a_4, a_5\}$ marked with big circles for which attacking edges were created $\{(a_1, 1), (a_2, 2), (a_3, 3), (a_4, 4), (a_5, 5)\}$ - each with the weight of 1. To solve this problem one needs to modify the values of the edges $\{(a_1, 1), (a_5, 5)\}$ to -1.

Hardness results. We first consider the NP-hardness of the *DMT* (*IMT*) and the *DNR* (*INR*) problems. In the proofs we will define standard polynomial time reductions [40] that show that every instance A of a problem that is known to be NP-hard can be transformed to an instance B of another problem in a way that the instance A can be solved if and only if the instance B can be solved.

We start with the hardness of the *DMT* (*IMT*).

Theorem 4. *Solving the* $DMT(IMT) = (G = (V, E, \omega),\ A, TP, I, t, k)$ *problem is NP-hard.*

Proof of Theorem 4. We reduce from the VERTEX COVER (VC) problem. In the VC problem we are given a parameter k and a graph $G = (V, E)$ and we need to decide whether there exists a set of vertices, $U \subseteq V$, $|U| \leqslant k$, that "cover" the set of the edges of this graph, i.e., every edge from E is adjacent to at least one node in U.

Given the VC problem (G, k), where $G = (V, E)$, we create an instance of our problem $DMT = (G' = (V', E', \omega), A, TP, I, t, k')$, by adding, for every node $v \in V$, an *attacking* node a_v with an edge (a_v, v) with *weight* $= 1$. We set the target threshold as follows: $t = -1$. We set $A = \{a_v : v \in V\}$, $V' = V \cup A$, $E' = \{(a_v, v) : v \in V\}$ (each of weight 1). Finally, the set of intermediary vertices is $I = V$, and the set of attacked edges TP is E, i.e., the set of the edges from G. We set $k' = k$. See Figure 2.10 for an example.

We now need to show that the reduction is correct. Firstly, given a graph $G = (V, E)$ and its vertex cover of size k, i.e., $U \subseteq V$ with $U = \{x_1, ..., x_k\}$, we show that its corresponding problem, *DMT*, as outlined above can be solved. To this end, we modify all of the k edges (a_{x_i}, x_i), where $x_i \in U$, by setting each of them to -1. Now:

- due to the fact that computing $(f(u), g(u)) = FGA(G, u)$ of a node u adjacent only to a single directed edge results in $g(u)$ being equal to the weight of this single edge, then for all $x_i \in U$ we have $g(x_i) = -1$;

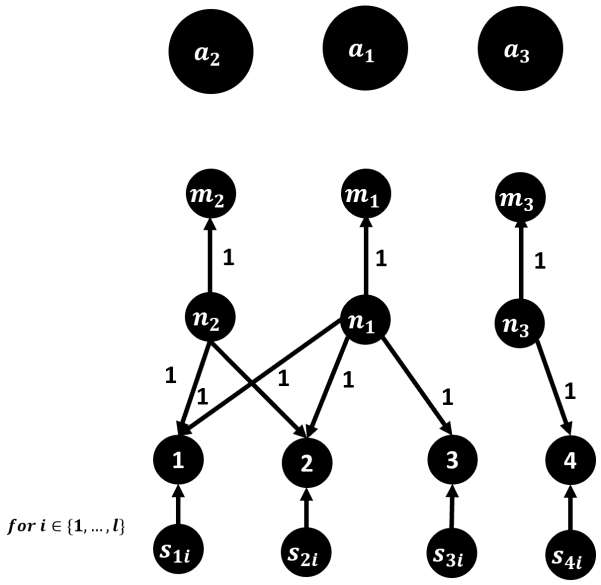


Figure 2.11: The corresponding *DNR* instance to the SC problem with target set $T = \{1, 2, 3, 4\}$ that has to be covered with at most $k = 2$ sets from $\mathcal{S} = \{\{1, 2, 3\}, \{1, 2\}, \{4\}\}$. In the *DNR* instance the set of target nodes is $\{1, 2, 3, 4\}$, the set of attacking nodes is $\{a_1, a_2, a_3\}$. The intermediary and helper nodes created for every set from $\mathcal{S}$ are $\{n_1, n_2, n_3, m_1, m_2, m_3\}$. Additionally, l stabilizing nodes s_{ji} are added to each node x_j, the number of moves is $k = 2$, and the threshold is set to $t = 1 - \epsilon$ (with l and ϵ as defined in the proof).

- from the definition of the *FGA* function, the fairness of nodes with out-degree of 0 is equal to 1. Hence, for all $u \in V$ we have that $f(u) = 1$.

From these we conclude that all of the connections in the target set TP are decreased to the threshold $t = -1$, i.e. either $f(u) \cdot g(v) = -1$ or $f(v) \cdot g(u) = -1$ for every pair $\{u, v\} \in TP$.

For the other direction, assume that we have a solution to our problem *DMT* that was created as outlined above. Recall that the set of attacking nodes is the set of newly created nodes for the *DMT* instance, i.e., $A = \{a_v : v \in V\}$, and each of them is connected with a single edge directed towards its corresponding node from V, i.e., (a_v, v) for $v \in V$ and weight of these edges equals 1. We observe that, from the definition of *FGA* and the construction of the *DMT* instance, it follows that to modify the values of the predicted connections between pairs $\{u, v\} \in TP$ (i.e. either $f(u) \cdot g(v)$ or $f(v) \cdot g(u)$) one needs to modify the goodness of the nodes in $dom(TP)$. This is because there are no outgoing edges from the nodes in TP; thus, fairness of the nodes in $dom(TP)$ is constant and equal to 1.

The goodness of the nodes in $dom(TP)$ can be modified by changing (a_v, v), where $a_v \in A$, or by adding some new edges between the attackers and the nodes from $dom(TP)$. However, to attack a single connection $\{u, v\} \in TP$, we have to obtain either $g(u) = -1$, or $g(v) = -1$. To this end, since reaching $g(v) = -1$ is only possible when all of the edges pointed at v have value -1, it is always necessary to modify the value of the existing edge (a_v, v) as well. Specifically, whenever we can reach one of the nodes in $dom(TP)$ with a

Algorithm 1: Direct attack

Input: $A, T = \{t\}, G$

for $a \in A$ **do**
  add an edge (a, t), $\omega(a, t) = -1$ to the graph G;

Algorithm 2: Indirect attack

Input: $A, T = \{t\}, G$

sort nodes in A by their fairness score
for $a \in sorted(A)$ **do**
  $N_1 \leftarrow Pred(t)$
  find a neighbor $n_2 \in Succ(n_1) \setminus \{t\}$ of a neighbor $n_1 \in N_1$ that minimizes the goodness value of t, when adding an edge (a, n_2) with weight $\omega(a, n_2) = 1$ or $\omega(a, n_2) = -1$
  add the edge to the graph G

modified edge, we are, in fact, "marking" all of the edges pointing at this node. Each of these edges corresponds to a pair in TP. If all the pairs are marked, then both the DMT and VC problems are solved. □

The proof for the IMT-hardness is analogous with the opposite signs of the weights of the created/modified edges.

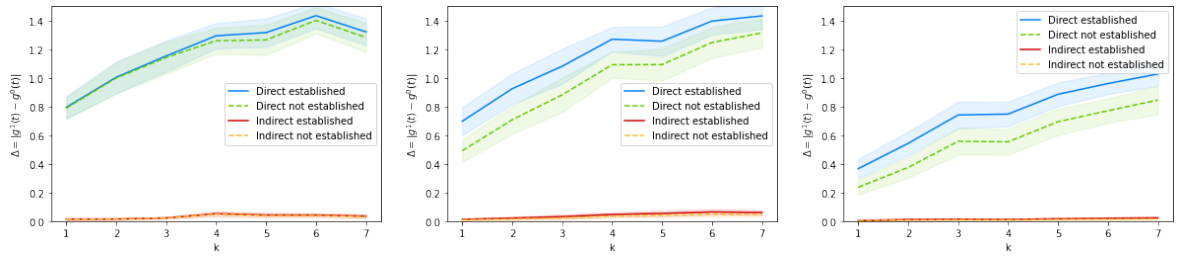


Figure 2.12: The comparison of direct/indirect, established/non-established attacks for Bitcoin OTC, Bitcoin Alpha and RFA Net networks.

Secondly, we study the hardness of $DNR(INR)$.

Theorem 5. *Solving the* $DNR(INR) = (G = (V, E, \omega),\ A, T, I, t, k)$ *problem is NP-hard.*

Proof of Theorem 5. We reduce from the SET COVER (SC) problem. In the SC problem, we are given a set of sets $\mathcal{S}$, a target set T, and a parameter k. We need to decide whether it is possible to cover the target set T with at most k sets from $\mathcal{S}$, i.e. whether there exists a subset $S \subseteq \mathcal{S}$ of size at most k, such that for all $t \in T$, there exists $S_i \in S$, such that $t \in S_i$. Given an SC problem $(\mathcal{S}, T, k)$, we create an instance of the DNR problem as follows:

- the set of target nodes in the DNR problem is the set T from the SC problem;

- For every set S_i from $\mathcal{S}$ we create an *intermediary* node m_i, one helper node n_i, and one link (n_i, m_i) and $|S_i|$ links (n_i, t) for $t \in S_i$, each of them with weight 1. We denote the set of all helper nodes n_i which point at the nodes m_i as $\mathcal{N}_{int}$;
- for each $S_i \in \mathcal{S}$ we create an attacking node a_i;
- we set the intermediary set I in the *DNR* problem to the set of m_i nodes;
- given $d_{max} = max\ \{outdeg(n_i) : n_i \in \mathcal{N}_{int}\}$, we add $l = 8d_{max}^3 - d_{max} + 1$ stabilising nodes s_i to each target node $x_j \in T$, they are required in the reduction to ensure that the change in goodness of some target nodes will not affect the goodness of the other target nodes too much;
- we set the target threshold in the *DNR* problem to be $1-\epsilon$, where $\epsilon = \frac{1}{4*d_{max}(d_{max}+l)}$,
- we set the budget to k.

In Figure 2.11, we present a sample *DNR* construction for the SC problem in which $T = \{1, 2, 3, 4\}$ has to be covered with at most $k = 2$ sets from $\mathcal{S} = \{\{1, 2, 3\}, \{1, 2\}, \{4\}\}$.

We now need to show that the reduction is correct. Firstly, let us consider an SC problem $(\mathcal{S}, T, k)$ and its set cover of size k, consisting of sets $S_i \in \mathcal{S}$ with indexes $i \in \{x_1, \ldots, x_k\} = U$. We will show that our corresponding *DNR* problem created as in the instructions above can be solved. To this end, we set new links (a_{x_i}, m_{x_i}) for $i \in U$ with weight -1.

In this case, the goodness of the intermediary node m_{x_i} decreases to a value bounded by the factor introduced by $\omega((a_{x_i}, m_{x_i})) \cdot f(a_{x_i}) = -1 \cdot f(a_{x_i}) \leqslant 0$ and the factor introduced by $\omega((n_{x_i}, m_{x_i})) \cdot f(n_{x_i}) \leqslant 1$, resulting in a value $g(m_{x_i}) \leqslant \frac{1}{2}$. This implies the decrease in the fairness value of the helper node n_{x_i}, resulting in $f(n_{x_i}) \leqslant 1 - \frac{1-\frac{1}{2}}{2*outdeg(n_{x_i})} \leqslant 1 - \frac{\frac{1}{2}}{2*d_{max}}$. Finally this decreases the goodness values of all nodes v_j rated by n_{x_i} to a value less or equal to $g(v_j) \leqslant \frac{indeg(v_j)-1}{indeg(v_j)} + \frac{1}{indeg(v_j)}(1 - \frac{\frac{1}{2}}{2*d_{max}}) \leqslant 1 - \frac{1}{4d_{max}*indeg(v_j)} \leqslant 1 - \frac{1}{4*d_{max}*(l+d_{max})} \leqslant 1 - \epsilon$. Since T is covered by the sets indexed by indices in U, then modifying the value of the links (a_{x_i}, m_{x_i}) decreases the rating of all of the target nodes in the *DNR* problem below or to the threshold.

For the other direction, assume that we have a solution to our corresponding problem *DNR*. In fact, since the only allowed actions are edge additions and weight updates to the nodes from $I = \{m_i\}_i$, the only way of modifying the goodness of the target nodes is by modifying the *fairness* of the helper nodes n_i. Either adding an edge (a_i, m_i) or adding an edge (a_j, m_i) marks a set $S_i \in \mathcal{S}$ and sets the value of the goodness of the nodes $k \in S_i$ below the threshold $1 - \epsilon$. We need to show that it is necessary to rank the node m_i to mark any node $v_j \in S_i$, otherwise their goodness value will stay above threshold (i.e. $goodness(v_j) > 1-\epsilon$ for every $v_j \in S_i$). We achieve this result by introducing l stabilising nodes for every $v_j \in T$. From the properties of the given construction one may conclude that marking nodes in the *DNR* problems implies marking sets in the set cover problem.

We will use the Theorem 6 introduced below the current proof. It shows that for a node x, when fairness of its k rating nodes is decreased by Δ, and there are l stabilising nodes rating it with 1, then the goodness value of the node x does not change too much - i.e. $g(x) \geqslant 1 - 2\frac{k}{l+k} \cdot \Delta$.

Using this result we may see that even in an edge case the nodes in the target set do not have their *goodness* value changed below the threshold if they are not marked properly as mentioned before. In the edge case a node v_i may be influenced by a set of k_1 nodes (denoted $\mathcal{K}$), which have their fairness value indirectly changed because they rate at most k_2 nodes (denoted $\mathcal{L}$) which are marked by at most k_3 helper nodes which change their fairness value by at most $\Delta = 1$. Note that $k_1, k_2, k_3 \leqslant d_{max}$.

In this case the goodness value of the nodes in $\mathcal{L}$ can be bounded by the above theorem $g(v_j) \geqslant 1-2\frac{d_{max}}{d_{max}+l}$. This implies that the fairness of the nodes in $\mathcal{K}$ falls to a value not less than $f(n_m) \geqslant 1-\frac{d_{max}}{d_{max}+l}$. This fairness modification will further influence the target nodes, but since in any scenario also for the target nodes we have $g(v_i) \geqslant 1 - 2\frac{d_{max}}{d_{max}+l}$, then the fairness value of the helper nodes will not fall below $1 - \frac{d_{max}}{d_{max}+l}$. Finally we can conclude that the nodes in the target set are influenced by at most $g(v_i) \geqslant 1 - 2\frac{d_{max}}{d_{max}+l}\frac{d_{max}}{d_{max}+l}$. We can see that when l is big enough, this value never reaches the threshold ϵ, i.e. $1 - 2\frac{d_{max}}{d_{max}+l}\frac{d_{max}}{d_{max}+l} > 1 - \frac{1}{4*d_{max}(d_{max}+l)}$ when $l > 8d_{max}^3 - d_{max}$. □

Theorem 6. *A node x that is rated by k "influencing" nodes n_i (for $i \in [k]$) and by l other "stabilising" nodes s_j (for $j \in [l]$) is given and all rates are of value 1. Suppose the fairness value of the influencing nodes decreases by at most Δ (after all modifications in the network), then the goodness value of the node x decreases by at most $2\frac{k}{l+k}\Delta$.*

Proof. By Monotonicity for Goodness we know that the goodness value of the node x will decrease maximally when we decrease the fairness value of all influencing nodes n_i by exactly Δ. We can estimate how the fairness value of the stabilising nodes ($f^{(t)}(s_i)$) and the goodness value of the rated node ($g^{(t)}(x)$) will change in the next iterations of the *FGA* function computation.

By the *FGA* definition, the stabilising nodes s_i which rate only one node x have $f^{(0)}(s_i) = 1$ and $f^{(t)}(s_i) = 1 - \frac{|1-g^{(t-1)}|}{2}$ for $t \geqslant 1$. What is more, since all ratings are of value 1, we know that $g(x) \geqslant 0$, and $f^{(t)}(s_i) = 1 - \frac{1-g^{(t-1)}}{2}$ for $t \geqslant 1$. The goodness value of the node x rated by k nodes n_i with decreased fairness and l stabilising nodes s_i, can be bounded using the above as follows - $g^{(0)}(x) = 1$ and $g^{(2t)}(x) \geqslant \frac{k}{k+l}(1 - \Delta) + \frac{l}{l+k} \cdot (1 - \frac{1-g^{(2t-2)}}{2})$ for $t \geqslant 1$. We prove by induction that for $2t \geqslant 2$ we have $g^{(2t)}(x) \geqslant 1 - \frac{k\Delta}{k+l}\sum_{2i=0}^{2t-2}[\frac{l}{2(l+k)}]^{2i}$. The statement trivially holds for $2t = 0$. Let us assume it holds for $2t$, then for $2t+2$ we have $g^{(2t+2)}(x) \geqslant \frac{k}{k+l}(1 - \Delta) + \frac{l}{l+k} \cdot (1 - \frac{1-g^{(2t)}}{2}) \geqslant \frac{k}{k+l}(1 - \Delta) + \frac{l}{l+k} \cdot (1 - \frac{1-[1-\frac{k\Delta}{k+l}\sum_{2i=0}^{2t-2}[\frac{l}{2(l+k)}]^{2i}]}{2}) = 1 - \frac{k\Delta}{k+l}\sum_{2i=0}^{2t}[\frac{l}{2(l+k)}]^{2i}$ what proves the induction. We may also further bound this sum $g^{(2t)}(x) \geqslant 1 - \frac{k\Delta}{k+l}\sum_{2i=0}^{2t-2}[\frac{l}{2(l+k)}]^{2i} \geqslant 1 - \frac{k\Delta}{k+l}\sum_{2i=0}^{2t-2}[\frac{1}{2}]^{2i} \geqslant 1 - \frac{2k\Delta}{k+l}(1 - (\frac{1}{2})^{t-1}) \geqslant 1 - 2\Delta\frac{k}{k+l}$. As stated in the preliminaries, the goodness value quickly converges to its limit and we can write $|g^{(2t)}(x) - g^{(2t-1)}(x)| < \frac{1}{2t}$, thus we can conclude that $g(x) \geqslant 1 - 2\Delta\frac{k}{k+l}$. □

The proof for the *INR*-hardness is analogous with the opposite signs of the weights of the created/modified edges.

Parametrized complexity. Finally, we study the complexity of our problems in terms of the W-hierarchy for the parameterized algorithms [39]. The parameterized reductions [39] transform every instance A with a parameter k of a problem known to be

in a given parameterized complexity class into an instance B with parameter k' of another problem in a way that (a) (A, k) can be solved if and only if (B, k') can be solved, (b) $k' \leqslant g(k)$ for some computable function g, (c) the running time is $f(k) \cdot |x|^{\mathcal{O}(1)}$ for some computable f.

The following results hold in terms of the parameterized complexity:

Theorem 7. *$DNR(INR)$ parameterized by k is $W[2]$-hard.*

Proof of Theorem 7. The Set Cover problem parameterized by the number of sets k is a $W[2]$-hard problem in the W-hierarchy. Since the reduction in the proof of Theorem 5 runs in polynomial time, the budget of the $DNR(INR)$ problem is k, this reduction is also a parameterized reduction [39]. □

Theorem 8. *$DMT(IMT)$ parameterized by k is $W[2]$-hard.*

Proof of Theorem 8. One needs to see that a slight modification of the reduction in the proof of the Theorem 5 is a parameterized reduction from the Set Cover problem parameterized by the number of sets k to $DMT(IMT)$ parameterized by the budget k. In fact, for a Set Cover problem we can create an instance $DMT(IMT) = (G = (V, E, \omega), A, TP, I, t, k)$ as in the $DNR(INR)$ reduction, but for each node x from the target set T in the corresponding DNR problem we add a vertex x', and we set $TP = \{\{x, x'\} : x \in T\}$. In this case since all of the new nodes are disconnected from the graph, the only way to break the connections between the $\{x, x'\}$ links below the given threshold t is to lower the *goodness* value of the nodes $x \in T$ below the given threshold. Again, the reduction runs in polynomial time, the budget of the $DMT(IMT)$ problem is k, this reduction is also a parameterized reduction [39]. □

2.5 Bounding the strength of direct and indirect attacks

In this section, we give bounds on the strength of the *direct* and *indirect* Sybil attacks, i.e., the attacks in which the attacker creates a new node when adding a new edge. The attacker can either rate its target *directly*, or make an attempt to manipulate the target's goodness score by rating other nodes in the network (i.e. conducting the *indirect* attack).

We denote by $g(l)/f(l)$ the goodness/fairness value of a node l before the Sybil attack, and by $g'(l)/f'(l)$ the goodness/fairness value of the node l after the attack. Let us define as well $\Delta g(l) = g'(l) - g(l)$ and $\Delta f(l) = f'(l) - f(l)$. To this end, we first define an *influence set* set $\mathcal{I}(u)$ of a node u, that contains nodes which goodness value is affected by changing the goodness value of the input node u. Figure 2.13 depicts an example of the influence set of a node u in an example network.

Definition 2.5.1 (Influence Set)**.** *Given network $G = (V, E, \omega)$ and a node $u \in V$, we define its influence set $\mathcal{I}(u)$ recursively. One begins with $\mathcal{I}(u) = \{u\}$. In each iteration, every node $v \in V$ which is indirectly connected to at least one node in $\mathcal{I}(u)$ [i.e. each node $v \in V$, for which $\exists_{v' \in \mathcal{I}(u), l \in V} : (l, v'), (l, v) \in E$)] is added to $\mathcal{I}(u)$. The procedure is repeated until the set $\mathcal{I}(u)$ stops growing.*

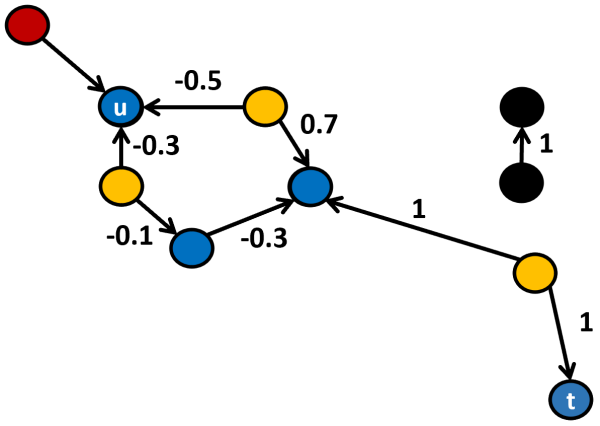


Figure 2.13: The nodes from the influence set $\mathcal{I}(u)$ of the selected node u are marked with yellow color. The node u is rated by an additional edge from the red node, affecting the selected node's goodness value. Next, the fairness value of the yellow nodes next to the node u is affected. The nodes with the affected fairness affect the goodness value of another pair of nodes. The process repeats until the node t is reached.

We start with the analysis of an *indirect* Sybil attack. Our results hold for a family of relatively dense networks, $G = (V, E, \omega)$, in which every node has a lower bound on its indegree and outdegree, i.e., $\forall_{v \in V} indeg(v) \geqslant k \;\; \& \;\; outdeg(v) \geqslant k$, and the intermediary nodes, $j \in V$, are relatively weakly rated, i.e. $\sum_{v \in Pred(j)} |\omega_{vj}| \leqslant k$. We call such networks *minimum-k-neighbour networks*.

Theorem 9 (Indirect Sybil attack)**.** *Assume a WSN $G = (V, E, \omega)$, where a new Sybil node s_i is added that rates some intermediary node $i \in V$ different than a target node $t \in V$. Whenever $\forall_{v \in V} indeg(v) \geqslant k \;\; \& \;\; outdeg(v) \geqslant k$, and $\forall_{j \in V} \sum_{v \in Pred(j)} |\omega_{vj}| \leqslant k$, then after the attack goodness value of the target node will change by no more than $\frac{2}{(indeg(i)+1) \cdot k}$.*

Proof. Let us define set V' as the influence set of the target node $\mathcal{I}(t)$. It is easy to see that the intermediary node i has to belong to V' in order to make the indirect attack successful.

For the target node, $t \in V'$, we can calculate how its $g(t)$ changes with respect to the changes introduced to the goodness of all other nodes. Here, we use the fact that the Sybil attack is indirect, i.e., the Sybil edge is not added to t.

$$g(t) = \frac{1}{indeg(t)} \sum_{u \in Pred(t)} f(u) \cdot \omega(u,t) =$$

$$\frac{1}{indeg(t)} \cdot \sum_{u \in Pred(t)} \omega(u,t) \cdot \left[1 - \frac{1}{outdeg(u)} \sum_{v \in Succ(u)} \frac{|\omega(u,v) - g(v)|}{2}\right]$$

Thus, from the triangle inequality:

$$|\Delta g(t)| \leqslant \frac{1}{2 \cdot indeg(t)} \sum_{u \in Pred(t)} \sum_{v \in Succ(u)} \frac{1}{outdeg(u)} \cdot |\omega(u,t)| \cdot |\Delta g(v)| \leqslant$$

$$\frac{1}{2 \cdot indeg(t)} \sum_{v : \exists (t,u) \in E \;\&\; (u,v) \in E} \sum_{u \in Succ(v)} \frac{|\Delta g(v)|}{outdeg(v)}$$

Finally, we obtain that:

$$|\Delta g(t)| \leqslant \frac{\sum_{v:\exists(t,u)\in E \ \& \ (u,v)\in E} |\Delta g(v)|}{2 \cdot indeg(t)}, \tag{2.4}$$

where $indeg(t) \geqslant k$.

Let us calculate $\Delta g(l)$ for all $l \in V'$. We can see that whenever we introduce a new node, s_i, that aims at node i in the network, then:

$$|\Delta g(i)| = \Big|\frac{-1 + \sum_{u\in Pred(i)} f'(u)\cdot\omega(u,i)}{indeg(i)+1} - \frac{\sum_{u\in Pred(i)} f(u)\cdot\omega(u,i)}{indeg(i)}\Big| \leqslant \tag{2.5}$$

$$\Big|\frac{\sum_{u\in Pred(i)} \Delta f(u)\cdot\omega(u,i)}{indeg(i)} - \frac{1}{indeg(i+1)} - \sum_{u\in Pred(i)} \frac{f'(u)\cdot\omega(u,i)}{indeg(i)(indeg(i)+1)}\Big| \leqslant$$

$$\Big|\frac{\sum_{u\in Pred(i)} \Delta f(u)\cdot\omega(u,i)}{indeg(i)}\Big| + \frac{2}{indeg(i)+1}$$

For the other nodes, $l \in V'$, that are not rated by s_i we have:

$$|\Delta g(l)| = \Big|\frac{\sum_{u\in Pred(l)} \Delta f(u)\cdot\omega(u,l)}{indeg(l)}\Big|$$

We can bound:

$$\Big|\sum_{u\in Pred(l)} \frac{\Delta f(u)\cdot\omega(u,l)}{indeg(l)}\Big| \leqslant \frac{1}{2\cdot indeg(l)} \sum_{v\in Pred(l), u\in Succ(v)\setminus\{l\}} \frac{|\omega_{vl}|\cdot|\Delta g(v)|}{outdeg(v)} = \tag{2.6}$$

$$\frac{1}{2\cdot indeg(l)} \sum_{i\in V} \sum_{(v,l),(v,i)\in E} \frac{|\omega_{vi}|}{outdeg(v)} |\Delta g(i)|$$

In a matrix form, we can thus write:

$$Q \leqslant M \times Q + \begin{bmatrix} 0 & \frac{2}{indeg(i)+1} & 0 & 0 & 0 \end{bmatrix}^T,$$

where Q is a vector of length $|V'|$ which on the l'th position has $\Delta g(l)$ for $l \in V'$. And M is a matrix of size $|V'|\cdot|V'|$, and its coefficients are filled according to Equation 2.6. Note that:

$$\frac{1}{2\cdot indeg(l)} \sum_{v\in Pred(l), u\in Succ(v)\setminus\{l\}} \frac{|\omega_{vl}|}{outdeg(v)} \leqslant \frac{1}{2} \tag{2.7}$$

This implies that $||M||_\infty \leqslant \frac{1}{2}$. On the other hand, for a given column j in the matrix M:

$$\sum_{l\in V} \frac{1}{2\cdot indeg(l)} \sum_{(v,l),(v,j)\in E} \frac{|\omega_{vj}|}{outdeg(v)} \leqslant$$

$$\frac{1}{2k} \sum_{l\in V} \sum_{(v,l),(v,j)\in E} \frac{|\omega_{vj}|}{outdeg(v)} \leqslant \frac{1}{2k} \sum_{v\in Pred(j)} |\omega_{vj}| \leqslant \frac{1}{2}$$

whenever $\sum_{v\in Pred(j)} |\omega_{vj}| \leqslant k$, which implies that $||M||_1 \leqslant \frac{1}{2}$. The values of $\Delta g(l)$ achieve maximum when:

$$Q = M \times Q + \begin{bmatrix} 0 & \frac{2}{indeg(i)+1} & 0 & 0 & 0 \end{bmatrix}^T$$

But in this case, we can solve the equation system with:

$$Q = \frac{1}{I - M} \times \begin{bmatrix} 0 & \frac{2}{indeg(i)+1} & 0 & 0 & 0 \end{bmatrix}^T$$

Matrix M is indeed invertible due to appropriately selected nodes $l \in V'$. What is more, since $||M||_\infty \leqslant \frac{1}{2}$, then we can write $\frac{1}{I-M} = I + M + M^2 + \ldots$ [109]. Finally, the above quality and $||M||_1 \leqslant \frac{1}{2}$ imply that:

$$|\sum_{l \in V} \Delta g(l)| \leqslant \frac{4}{indeg(i) + 1}. \tag{2.8}$$

Now, by the Equation 2.4, and the above result $|\sum_{l \in V} \Delta g(l)| \leqslant \frac{4}{indeg(i)+1}$, we get that:

$$|\Delta g(t)| \leqslant \frac{2}{(indeg(i) + 1) \cdot k}.$$

□

Additionally, the Equation 2.4 in the above proof shows that in a minimum-k-neighbour network the bound for the direct attack is approximately k times stronger than for the indirect attack. That is, when we modify the goodness value of some node i by Δ, then the value of the target node t is modified by at most $\frac{\Delta}{k}$. We summarize this observation in the theorem below.

Theorem 10 (Direct Sybil attack)**.** *Assuming in a WSN $G = (V, E, \omega)$ where one adds a new Sybil node rating directly some target node t, then the goodness value of the target node t decreases by at most $|\Delta g(t)| \leqslant \frac{3}{indeg(t)}$.*

Proof of Theorem 10. Note that the change in the goodness value of the nodes rated directly $\Delta g(t)$ can be bounded as in the Equation 2.5 in the proof of Theorem 9. We can thus set $\Delta g(t) \leqslant \left| \frac{\sum_{u \in Pred(t)} \Delta f(u) \cdot \omega(u,t)}{indeg(t)} \right| + \frac{1}{indeg(t)}$. The Equations 2.4 and 2.8 show that $\Delta g(t) \leqslant \frac{2}{indeg(t)(indeg(t)+1)} + \frac{1}{indeg(t)}$, what implies the result. □

2.6 Simulations

We conduct a series of simulations[2] on the Bitcoin OTC, Bitcoin Alpha, and RFA Net datasets studied by Kumar et al. (2016). They consist of weighted signed networks with $|V| =\geqslant 3,700$, $|E| \geqslant 24,000$ each, where the proportion of positively weighted edges is $\geqslant 84\%$. A vast majority of the nodes in each network, i.e., more than 76%, have an indegree up to 10. Furthermore, most of the users in the networks are evaluated as fair by the *FGA* function—$f(v) \geqslant 0.7$ for 100% of the nodes (with the mean $f(v)$ equal to 0.94). As for goodness, only less than 4% of users have a strongly positive score of more than 0.5, and in the Bitcoin OTC network 8% of users have negative score of less than -0.3, whereas in Bitcoin Alpha $3,8\%$ have goodness below -0.3. Section 2.8 presents more statistics of the data and algorithms used in this section.

[2]The code package that allows running simulations presented in Sections 2.6 and 2.7 is available under `https://github.com/irtomek/WeightPredictionsCode`.

	B. OTC	B. Alpha	RFA Net
max *indeg*	10	13	10
min *goodness*	0.8	0.5	0.5
num. of samples	20	30	27
num of edges	20	20	20

Table 2.1: The parameters used to search weak target nodes in the test sets. "B." stands for "Bitcoin".

	B. OTC	B. Alpha	RFA Net
average change	0.081	0.085	0.030
standard deviation	0.089	0.085	0.028
min change	0.010	0.008	0.009
max change	0.300	0.298	0.131
median	0.053	0.042	0.021
0.75-quantile	0.111	0.121	0.041

Table 2.2: The results of Algorithm 3 on different datasets.

We focus on the attacks that lower the *goodness* of the nodes, as in the *DNR* problem. In particular, each experiment was conducted on the set of attacking nodes A of size $k = \{1, \ldots, 7\}$ and the target set $T = \{t\}$ of size 1. The target, $t \in T$, was chosen randomly from those nodes that have relatively high goodness ($g(t) \geqslant 0.50$) and a low indegree ($0 < indeg(t) < 10$). We study two types of the attackers:

- **not-established** attackers — chosen from relatively newly created nodes with $0 < indeg < 10$ and $outdeg = 0$). This allows for studying Sybil-style attacks; and
- **established** attackers — chosen from the nodes with $outdeg(v) > 5$ (and iteratively choosing nodes with fairness $f(v) > 0.7$). This allows for studying attacks by the nodes whose standing in the network has been built for some time.

We simulate three types of attacks:

- **direct attacks** — a set of attackers A of size k rates directly the target node $t \in T$. The pseudocode is presented in Algorithm 1. Each attacker rates t using weight -1;
- **indirect attacks** — the attackers set A of size k rates the neighbors of the neighbors of the target node, to minimize the goodness part of the *FGA* of the target node by manipulating *fairness* of the targets' neighbors. The pseudocode of the attack is presented in the Alorithm 2. More precisely, the algorithm implements a greedy approach, where each new edge is used to minimize the *goodness* of the target node t by minimizing (or maximizing) the *fairness* of one of the targets' neighbors by directly rating the successor of the target's neighbor with an edge of weight 1 or -1. The algorithm performs calculations iteratively on the attackers sorted by the value of their fairness value.
- **mixed attack** — k_1 attacking nodes perform a direct attack and k_2 perform an indirect one, where $k_1 + k_2 = k$.

The results in Figure 2.12 are presented with a 95% confidence interval (marked with the opaque region around the solid/dashed lines). They show how a direct/indirect attack

Algorithm 3: Modified indirect attack

Input: $A, T = \{t\}, G$

sort nodes in A by their fairness score
while $i < len(sorted(A))$ **do**
  $a \leftarrow sorted(A)[i]$
  $N_1 \leftarrow Pred(t)$
  find a neighbor $n_2 \in Succ(n_1) \setminus \{t\}$ of a neighbor $n_1 \in N_1$ that minimizes the goodness value of t, when adding an edge (a, n_2) with weight $\omega(a, n_2) = 1$ or $\omega(a, n_2) = -1$
  $edges\_len \leftarrow min(SCALE * len(indegree(n_2), MAX, len(A) - i)$
  add $edges\_len$ edges to the graph G
  $i \leftarrow i + edges\_len$

by established/not established nodes influences the goodness of the target node (Δ). For Bitcoin OTC and Bitcoin Alpha, and RFA Net in both cases (direct and indirect attacks), there is no significant difference between the established and not established results (solid and dashed lines).

In Figure 2.14, we present results for a mixed setting. The individual cells of the heatmaps show: (a) Δ_1 — the absolute change of the goodness of the target node introduced by the k_1 direct edges; (b) Δ_2 — the absolute change of the goodness of the target node introduced by the k_2 indirect edges; and (c) Δ_S — the total change, i.e., $\Delta_S = \Delta_1 + \Delta_2$. The results show that the average strength of a direct attack varies between 0.2 and 1.2 for different k, and the average strength of the indirect attack is lower than 0.05, i.e., significantly smaller than the average strength of a direct attack. This result, together with the theoretical results achieved for a selected class of networks in Section 2.5, supports the claim that attacking nodes indirectly is (significantly) more difficult than attacking nodes directly.

2.7 Better heuristic for indirect attacks

We conduct additional tests to analyze the strength of the indirect attacks. We attempt to strengthen the indirect attack, allowing for a greater budget for the attack. In Algorithm 3, instead of adding only a single edge in each iteration, as in Algorithm 2, we add a series of new edges in every iteration. In more detail, we add $SCALE = 5$ times more Sybil edges than the indegree of the target node (but at most some predefined maximum $MAX = 10$). We take this approach to scale up the effect of manipulating the goodness value of the target nodes. We attack only nodes with bounded *indegree*, and the goodness value bigger than some threshold. We believe these nodes are more easily manipulable than an average node. See Table 2.1 for the details.

The analysis of the data in Table 2.2 shows that the attack using the Algorithm 3 may (but rarely does) achieve relatively strong results. To be more precise, the maximum change of the goodness value of the target node introduced by the indirect attack in the Bitcoin OTC and Bitcoin Alpha detasets reached the barrier of 0.3. This shows that in general networks (in comparison to the minimum-k-neighbour networks described in

Theorem 9) may not be strongly resistant against indirect attacks. In most cases, however, the attack gives rather weak results (75%-quantile on all datasets is at most 0.12 with low median of at most 0.05). The attack in all datasets influences the goodness value of the target node by at least 0.01.

2.8 Datasets' basic statistics

Figure 2.15 presents the histogram of the nodes sorted per indegree. Table 2.3 depicts the number of samples that were used to simulate the direct and indirect attacks for all sizes of the attacking sets considered. The nodes were chosen randomly from the set of all nodes of an example network. For mixed attacks, for each $k_1, k_2 \in \{1, \ldots, 6\}$, at least 26 samples were used for Bitcoin OTC, at least 12 samples were used for Bitcoin Alpha, and at least 17 samples were used for RFA Net.

k	B. OTC	B. Alpha	RFA Net
1	24	24	25
2	21	24	25
3	26	24	25
4	25	24	25
5	24	24	25
6	23	24	25
7	22	24	25

Table 2.3: Number of samples used to simulate direct/indirect established/not established attacks.

	B. OTC	B. Alpha	RFA Net
Size	5881	3783	9654
Edges	35592	24186	104554
Positive edges	$89,90\%$	$93,64\%$	84%
Small in-degree $< 10\%$	$87,40\%$	$85,70\%$	76%
Fair nodes $\geqslant 0.95$	$99,45\%$	$99,65\%$	$99,99\%$
Fair nodes $\geqslant 0.7$	100%	100%	100%
Goodness score $\geqslant 0$	85.85%	92.36%	93%
Goodness score $\geqslant 0.5$	2.2%	3.3%	67%
Goodness score $\leqslant -0.3$	8.2%	3.8%	0.2%

Table 2.4: Statistics of the networks used for simulations.

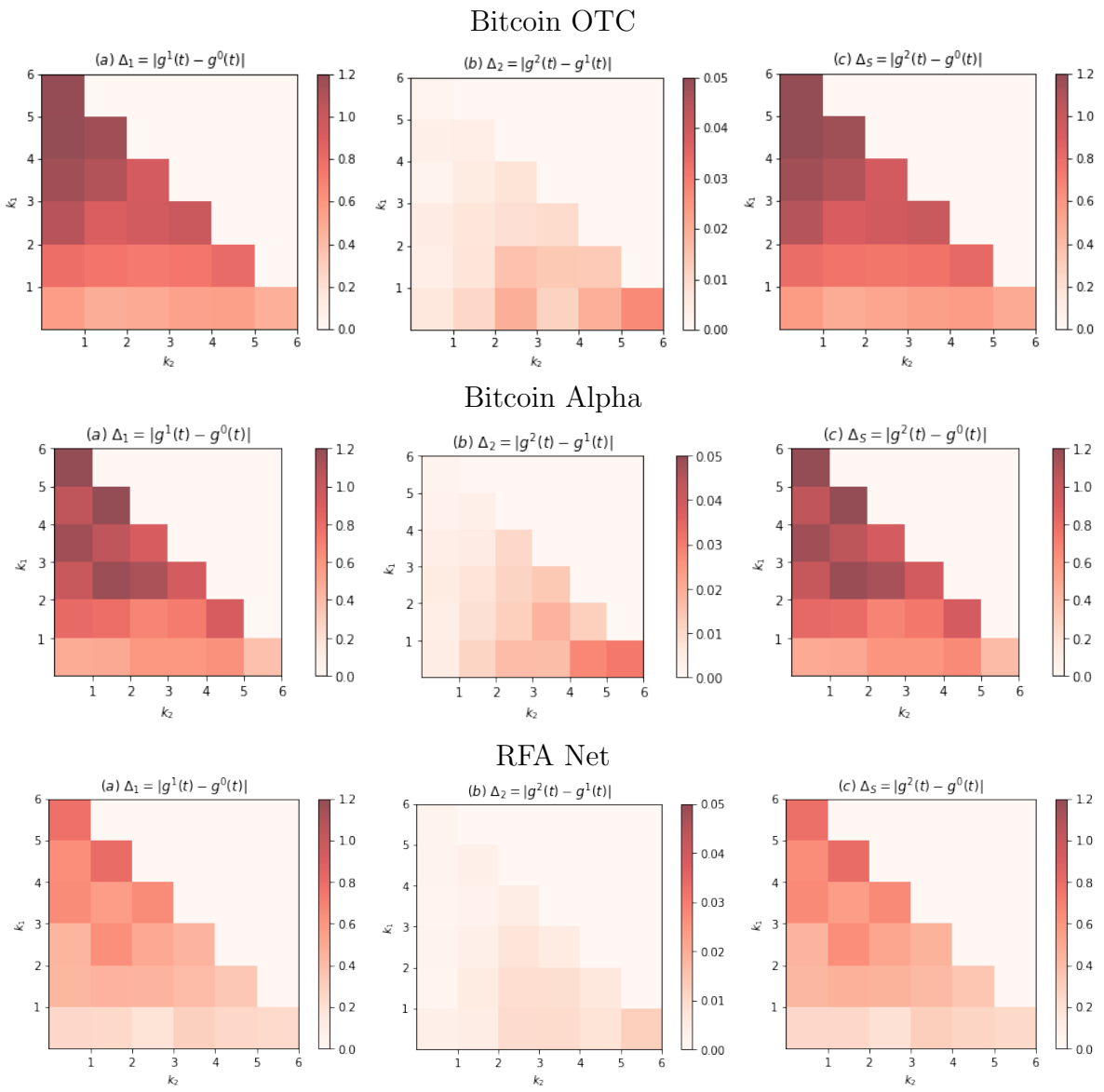


Figure 2.14: The results for the mixed settings in Bitcoin OTC, Bitcoin Alpha, RFA Net. The average strength of an indirect attack is small and significantly smaller that the average strength of a direct attack. Δ_1 shows the influence of the attack with k_1 direct edges, Δ_2 shows the influence of the attack with k_2 indirect edges, Δ_s shows the influence of the attack with k_1 direct edges and k_2 indirect edges.

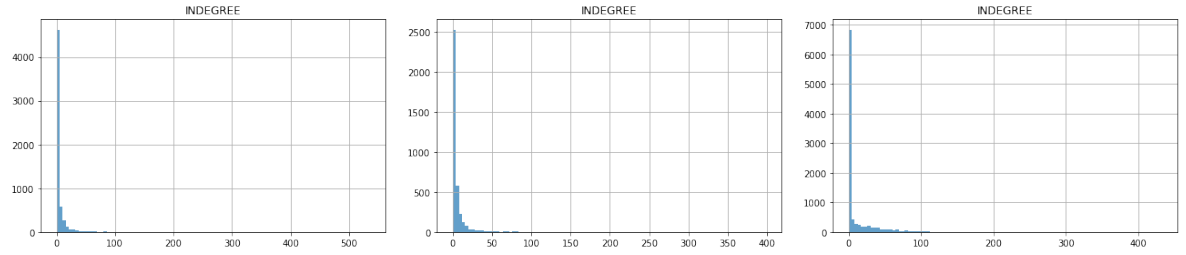


Figure 2.15: Histograms of indegree of the nodes of the Bitcoin OTC, Bitcoin Alpha, RFA Net networks.

Chapter 3

Individual Cryptography

In this chapter, we initiate a formal study of *individual cryptography*. Informally speaking, an algorithm Alg is *individual* if, in every implementation of Alg, there always exists an individual user with full knowledge of the cryptographic data S used by Alg. In particular, it should be infeasible to design implementations of this algorithm that would hide S by distributing it between a group of parties using an MPC protocol or outsourcing it to a trusted execution environment.

Additionally, we define and construct an example primitive in the model of individual cryptography, as well as picture its application to the security of systems built upon blockchains. Our new primitive - a *Proof of Individual Knowledge*, is a tool for proving that a given message is fully known to a single ("individual") machine on the Internet, i.e., it cannot be shared between a group of parties. A central technique for constructing individual cryptographic primitives is the concept of *MPC hardness*. Carefully crafting functions that are *MPC-hard* precludes an adversary from completing a cryptographic task in a distributed fashion within a specific time frame.

3.1 Preventing Secret Holder from Collusions

Multiparty computation (MPC) [68, 32, 15] is a powerful cryptographic technique that enables parties to evaluate any function securely in the presence of an adversary. It guarantees that nothing is revealed about the honest parties' inputs except what can be learned from the function's output. MPC protocols have found countless applications in cryptography and are one of the main tools for achieving privacy. In addition, MPC technology is becoming widely available in practice, e.g., for machine learning and blockchain applications. While MPCs are traditionally used for the "good" with their increasing availability in practice, there is a danger that an adversary misuses this technology to carry out malicious tasks. Let us illustrate this with the following example.

Identity sharing over the Internet. Imagine a service provider $\mathcal{S}$ that maintains a system in which individual users $\mathcal{U}$ can open accounts by paying a subscription fee. To lower this fee, the malicious users decide to open a *single* account and to share the credentials S to it between each other. In this chapter, we are interested in situations when these users are individuals $\mathcal{A}_1, \ldots, \mathcal{A}_a$ connected via the Internet that do not trust each other, i.e., we are *not* concerned about scenarios in which the credentials are shared

between different devices that belong to a single person, between members of one family, or between devices that are located physically very close to each other (so the network connection speed does not matter).

Suppose that to discourage such users from simply sharing S in plaintext, the service provider integrates ad-hoc countermeasures such that knowledge of S suffices to damage the account significantly. For example, if the "account" is a cryptocurrency address, then the knowledge of S should suffice to drain the account from all the coins. Alternatively, if it is an online storage system, then knowledge of S should permit deleting all the files. While these countermeasures should suffice to discourage the users from sharing S in plaintext, they are not sufficient to protect against a more sophisticated *identity sharing attack*, where the malicious users share S using secret sharing and jointly emulate a single "virtual" user $\mathcal{U}$ using an MPC protocol to authenticate with $\mathcal{S}$.

The identity-sharing attack is similar to the "identity-lease attack" recently introduced by Puddu et al. [101]. In an identity-lease attack, the attacker's primary goal is to temporarily outsource ("lease") its identity to a third party, who can control it for a specific time or purpose. As discussed in [101], this may, for instance, be problematic in electronic voting, where identity leasing can be used to sell votes. While Puddu et al. rely on a trusted execution environment for their attack, it is possible to carry out the same attack using an MPC committee. Here, the committee holds secret shares of the user's identity and is queried by the third party to carry out the desired task (e.g., vote for a certain party in an electronic election). For a more detailed description of the attack on voting systems, please refer to Section 3.5.

MPC hardness. By closely examining the two previous examples, we make the following crucial observation. Both attacks rely on the assumption that a distributed adversary can efficiently evaluate the cryptographic task via the MPCs. Hence, to thwart these attacks, our key idea is to make these cryptographic tasks *MPC-hard*. Informally, we say that a task/function is MPC-hard if executing it securely in a distributed way takes a significant amount of time. This implies that if a cryptographic task is MPC-hard, then in order to run it efficiently, the parties need to execute it *individually*. We formalize this new notion through a concept that we call *individual cryptography*.

Let us take a look at how MPC hardness may help us to prevent the previous two attacks. In the case of the identity sharing example, the *distributed adversary*, i.e., a tuple $\mathcal{A}_1, \ldots, \mathcal{A}_a$ of interactive machines (also called the *sub-adversaries*), uses an MPC protocol to ensure that no party individually knows the credentials to authenticate with a service provider $\mathcal{S}$. Suppose the service provider will only accept an authentication attempt if it is completed within a certain time frame. The distributed adversary now has two options. Either it runs the authentication process via the MPC. This, however, will fail due to MPC hardness. Alternatively, the adversary may ask one of the sub-adversaries, say $\mathcal{A}_j$, to execute the task *individually*, in which case $\mathcal{A}_j$ has to know the credential S entirely.[1] We formalize this concept via a new primitive that we call *Proof of Individual Knowledge (PIK)*. Informally, a PIK guarantees that the prover must know the entire secret if it wants to get accepted by a verifier. We will provide further details on PIKs in Sect. 3.1.2.

[1]Recall that in this case, the sub-adversary $\mathcal{A}_j$ can take full control over the account of the user, which was something that a malicious user tries to avoid.

Attacks via the trusted execution environment (TEE). An alternative way to perform the aforementioned attacks is to use trusted execution environments (such as Intel's SGX) to accelerate the MPC (see, e.g., [7]) or to outsource secret storage in a way similar to the one described in [101]. To make our schemes secure against such attacks, we need to make some additional assumptions about what kind of fast computation is infeasible in TEEs. One option is to assume that the honest users are equipped with hardware similar to Bitcoin mining rigs that can compute a massive amount of hashes in parallel (note that our PIK protocol is based on massive parallel computation, while the ISS uses sequential computation). Another option is to come up with new hash functions that are infeasible to compute quickly on TEEs (e.g., due to large memory requirements). We leave it as future work to design such hash functions.

3.1.1 Informal description of our model

As standard in cryptographic modeling, we have to describe the adversarial model (i.e., the adversaries' abilities) and specify what it means for a cryptographic task to be secure. Let us start with the adversarial model.

The adversarial model. As our MPC-hard functions will be based on massively evaluating a hash function, we give the distributed adversary $\mathcal{A}_1, \ldots, \mathcal{A}_a$ access to a special oracle Ω_H that allows evaluating a fixed input-length hash function $H : \{0,1\}^\lambda \to \{0,1\}^\kappa$. The oracle accepts queries of the form (x, mode), where $x \in \{0,1\}^\lambda$ and $\mathit{mode} \in \{\mathsf{fast}, \mathsf{slow}\}$. If $\mathit{mode} = \mathsf{fast}$, then a query is called *fast*. It is called *slow* otherwise. Let us give some intuition behind these two modes.

The fast queries are hash function evaluations that a sub-adversary $\mathcal{A}_b$ runs locally. We assume that these evaluations can be done very fast (orders of magnitude faster than slow queries). For example, a party may execute them using a specially designed ASIC, such as used in the context of Bitcoin mining. We assume that the number of fast queries is only bounded by the running time of the adversary, which is polynomial time. On the other hand, the slow queries model an evaluation of the hash function using an MPC protocol. In particular, this means that the sub-adversaries $\mathcal{A}_1, \ldots, \mathcal{A}_a$ can learn $H(x)$ without knowing x. Since the evaluation of a hash function using MPC technology is conceivable much slower than using an ASIC, we assume that the budget of the adversary for such queries is comparably small, i.e., bounded by some parameter σ.

In addition to bounding the number of slow queries that the sub-adversaries may ask for, in our PIK application, we put an additional restriction on the sub-adversaries. We require that they run in at most ρ communication rounds. Notice that we do not require any bounds on the communication complexity as at the end of each round, we allow the sub-adversaries to share any information that they currently possess.

Attacks exploiting the hash function structure. We need to stress that in the model above, the hash function $H : \{0,1\}^\lambda \to \{0,1\}^\kappa$ needs to be chosen carefully, and it is unrealistic to assume that λ is large. It is also important that H does not have a structure that the adversary could exploit. A natural choice for H is a compression function of a popular hash function (for example, in SHA-256, the compression function is of a type $H : \{0,1\}^{728} \to \{0,1\}^{256}$).

We illustrate this by the following example. Suppose H is a Merkle-Damgard-type hash function, and let $G : \{0,1\}^{2\kappa} \to \{0,1\}^{\kappa}$ be the compression function that H is built from, i.e., for messages of a form $(S_1 \,||\, S_2 \,||\, W)$ (with $|S_1| = |S_2| = |W| = \kappa$) we have $H(S_1 \,||\, S_2 \,||\, W) = G(G(G(IV \,||\, S_1) \,||\, S_2) \,||\, W)$ (where $IV \in \{0,1\}^{\kappa}$ is some fixed initial value, and for simplicity, we omit the last block encoding message length). Now, suppose that we want to verify if a message S consisting of two blocks (S_1, S_2) (both of length κ) is stored on one machine by forcing the prover to compute a large number of hashes of a form $H(S, W)$ (for multiple W's), which is the case for our PIK construction (see below). Taking into account the structure of H, two sub-adversaries: $\mathcal{A}_1$ and $\mathcal{A}_2$ can now "split" the computation as follows: the $\mathcal{A}_1$ first computes $h = G(IV, S_1)$, and then sends h to $\mathcal{A}_2$ who can compute $H(S_1 \,||\, S_2 \,||\, W)$ for an arbitrary number of different W's *without* knowing S_1, just by using that fact that $H(S_1 \,||\, S_2 \,||\, W) = G(G(h \,||\, S_2) \,||\, W)$. Hence, if H is a Merkle-Damgard type of hash function, the assumption that the input of every fast query is known to the adversary would be unrealistic. The same applies to sponge-based hash functions and to any other hash functions that read their input blocks in an online way and compress them to a shorter state.

Security against distributed adversaries. By using MPC-hardness, we want to enforce that the distributed adversary must run the cryptographic task individually. More concretely, if the adversaries manage to complete the cryptographic task within some specified time bound (seconds in the case of PIK and hours in the case of ISS), then one of the adversaries, say $\mathcal{A}_j$, must know some secret information S completely. Hence, we need some formal method to model "knowledge".

The question of formalizing knowledge has a long history in cryptography, and in particular, it has been studied in the context of "proofs of knowledge" [12], "knowledge of exponent" [44], or "plaintext-awareness" [13]. None of these approaches considers attacks by a distributed adversary. The most relevant to ours is the model of [13], which considers an adversary with access to a hash function (modeled as a random oracle). It is assumed that if an adversary $\mathcal{A}$ evaluated H on some input x, then $\mathcal{A}$ knows the input x and the corresponding output $H(x)$. Technically: the (input, output) pairs are later given to an algorithm $\mathcal{E}$ called "knowledge extractor". If, based on these tuples, the knowledge extractor outputs some message S, then we assume that "$\mathcal{A}$ knows S" (since $\mathcal{A}$ could have computed S herself just by observing the oracle queries and the replies to them).

In our case, we use the concept of a knowledge extractor but slightly adjust it in the following way. First, we will consider knowledge extractors $\mathcal{E}_b$ for each of the different adversaries $\mathcal{A}_b$. Second, each such knowledge extractor takes as input the transcript of queries $\mathcal{T}_b^{\mathsf{fast}}$ that $\mathcal{A}_b$ has made to the oracle Ω_H only in $mode = \mathsf{fast}$. Put differently: queries made by $\mathcal{A}_b$ in $mode = \mathsf{slow}$ (recall that these queries model MPC evaluations of H with a potentially unknown input) are not given to the knowledge extractor $\mathcal{E}_b$. Finally, we say that an adversary $\mathcal{A}_j$ individually knows a secret S if there exists an efficient knowledge extractor $\mathcal{E}_j$ such that $S \in \mathcal{E}_j(\mathcal{T}_b^{\mathsf{fast}})$.

Allowing pre-computation of the hash function. To model realistic attacks, we do not make any assumptions about how much time the distributed adversary has before the protocol starts. In particular, we allow the sub-adversaries to perform any distributed computation that involves the hash function H before the beginning of the protocol. This

will be reflected by assuming that the attack works in two phases: in the first one, called *pre-processing* phase, the adversary will not be restricted in the number of slow hash queries that she can evaluate. Such a restriction will only apply in the second, *online*, phase. We protect our protocols against such pre-computation attacks by making all hashes depend on random values that are unknown before the online phase.

3.1.2 Informal description of PIK

As outlined in the introduction, a PIK protocol should allow the prover $\mathcal{P}$ to convince the verifier $\mathcal{V}$ that a secret S (that they both know) is stored on one machine and known to the machine owner. This is done by forcing $\mathcal{P}$ to quickly reply to challenges Z from $\mathcal{V}$ in a way that proves that $\mathcal{P}$ performed intensive computation on the entire value of S. Since $\mathcal{P}$ measures the response time, it will notice any response delays that are due to the fact that S is, in fact, distributed between different "sub-adversaries", and it is not stored on one machine. This distinction is made by performing a large amount of "hash computations" on S, which in practice, for example, can be carried out via a highly optimized ASIC.

As outlined in the previous subsection, two main parameters that characterize the adversary are σ – the number of hashes that can be computed in such a way that no sub-adversary learns their inputs since they are computed using MPC (in our model, this corresponds to the "slow" queries to the oracle), and ρ – the number of communication rounds between the $\mathcal{A}_b$'s. In Sect. 3.1.1, we already explained the idea behind the slow queries. The bound on the number of communication rounds is relatively mild in practice. Recall that $\mathcal{A}_b$'s are connected over the Internet, and hence assuming that no more than 1000 rounds of communications per second (say) are executed between them is reasonable.

Our scheme is described formally in Sect. 3.4. The reader may, in particular, look at the diagram in Fig. 3.3 – we will refer to it while presenting our solution informally below. Let us start with describing a simple scheme for proving knowledge of a message consisting of one block (i.e., messages of a form $S = (S_1)$, where $S_1 \in \{0,1\}^\lambda$), and assuming that (a) the sub-adversaries cannot execute any slow queries, and (b) the sub-adversaries cannot communicate during protocol execution. In this case, the following idea works: the verifier sends a challenge Z to the prover, and the prover has to respond with $h = H(Z \,||\, S_1)$. Since no slow queries are allowed, and the sub-adversaries cannot communicate. Thus one sub-adversary, say $\mathcal{A}_b$, needs to know (Z, S_1). Let us now show how to remove our artificial assumptions in the above example.

Allowing slow queries. Recall that above, we assumed that the sub-adversaries could not perform any slow queries (i.e., no H can be computed using MPCs). We eliminate this restriction in the following way: namely, we force the prover to perform multiple computations of hashes on different nonces to find a nonce that leads to a hash starting with ζ zeros. This ensures that to convince the verifier, there must be an individual adversary $\mathcal{A}_b$ that makes a large number of fast queries that contain as input S_1. Notice that our approach is very similar to Bitcoin's puzzles, except that we do it κ times to reduce the variance in finding a solution. The nonce that is used in the ith puzzle is denoted with W^i.

Longer messages S. Let us now discuss how to eliminate the assumption that S just consists of one block. Let $S = (S_1, \ldots, S_n)$ where $n \geqslant 1$. Our idea is straightforward (see also the first column on Fig. 3.3, ignoring the values at the beginning of the hashed blocks): we apply the construction for a single block iteratively, i.e., for the ith nonce W we let $Q_1 := H(Z \,||\, W)$, and then for each $j = 2, \ldots, n+1$ we let $Q_j := H(S_j \,||\, Q_{j-1})$.

Allowing communication between the sub-adversaries. Note that the above construction is insecure if there are no restrictions on the communication between the $\mathcal{A}_b$'s. Indeed, imagine two sub-adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ and suppose S has two blocks $S = (S_1, S_2)$, with $\mathcal{A}_1$ holding S_1 and $\mathcal{A}_2$ holding S_2. Then these adversaries can break the above PIK as follows: $\mathcal{A}_1$ computes a massive number of hashes $Q_2 := H(S_1 \,||\, H(Z, W))$ (for different values of W) using fast queries to Ω^{fast} and sends the Q_2's to $\mathcal{A}_2$. Sub-adversary $\mathcal{A}_2$ processes every Q_2 by computing $Q_3 := H(S_2 \,||\, Q_2)$ in order to find Q_2 such that Q_3 starts with ζ zeros. Once such Q_2 is found, she communicates it to $\mathcal{A}_2$, who checks which W this Q_2 corresponds to and sends this Q to the verifier.

The above example shows that we need to restrict communication between the sub-adversaries. As discussed in Sect. 3.1.1 we choose to do it by putting a bound ρ on the number of rounds (an alternative approach would be to bound the communication size, but it seems more challenging to work with in practice). In our construction, we use this assumption by requiring that the prover needs to compute $d = \rho + 1$ iterations of the procedure outlined above (cf. Fig. 3.3).

Dealing with artificial fast queries. While from the above, it is clear that one of the sub-adversaries $\mathcal{A}_b$ has to know the entire S, it is not immediately clear how to build an efficient knowledge extractor. One challenge here is that the adversary may try to "confuse" us by making "useless" fast queries to Ω_H. We address this challenge by adding 2-bit flags to the inputs on which the hash function is evaluated. This enables the knowledge extractor to identify starting points for efficiently extracting the most likely values for S.

3.1.3 Related work

Using cryptography for malicious purposes has been studied before, most notably in the context of "Cryptovirology" proposed by Young and Yung [118]. The approach of [118] focuses on the malicious use of public-key encryption and not the MPCs. Hence, our concept can be viewed as a natural continuation of the approach of [118] (with MPCs being more "advanced" primitives than the public-key encryption schemes). The idea of preventing leaking secrets has been studied extensively in a context such as traitor-tracing, e.g., it [35, 82]. To our knowledge, none of these works considers a distributed adversary.

On a higher level, our concept is also related to works that look at defining the notion of "identity" in cryptography. In particular, it has some similarities to Position-Based Cryptography [30], where an identity of a user is defined by its geographic location. This approach is also based on measuring the prover's response time. Still, it is assumed that the communication is not done over the Internet but via physical signals (the whole approach is based on the fact that the speed of electromagnetic signals is fixed). Another

difference is that the users in [30] do not have secret credentials but are identified by their geographic location. As already mentioned, our proof techniques are similar to those used in the context of space-bounded cryptography, in particular in the construction of schemes that are secure based on assumptions about the restricted memory of the adversary and whose security relies on hash functions, see, e.g., [48, 50, 54, 3]. For the differences between our model and the ones used in this area, see Sect. 3.1.1.

Concurrent work. In a very recent concurrent work, Kelkar et al. [81] introduce the concept of *complete knowledge*. A proof of complete knowledge (PoCK) guarantees that a single party has complete control/knowledge of its secret. This is very similar to our notion of PIK. We emphasize that while both works aim at similar goals, there are significant differences. Their construction of a PoCK directly achieves a zero-knowledge property. In contrast, our basic PIK construction does not have this property (we outline a generic transformation of any PIK into a zkPIK in Sect. 3.4.4). In addition, they also present an implementation. On the other hand, their work is very much application-oriented. It lacks a formal model that takes into account the many subtleties that we attempt to model with the concept of individual cryptography (e.g., possible communication, fast/slow queries, and taking into account how hash functions are constructed in practice). In addition, they do not have a construction of individual secret sharing, which shows that our modeling has applications beyond proof of knowledge.

3.2 Preliminaries

A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* if for every positive integer c there exists an integer N such that for every $n > N$ we have $|f(n)| \leqslant n^{-c}$. A sequence of events has an *overwhelming probability* if the probability of their negations is negligible. We will use the following standard fact.

Lemma 11. *For every $p < 1$ we have that $(1-p)^{1/p} \leqslant e^{-1}$.*

A pair of algorithms $(\mathsf{Enc} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*, \mathsf{Dec} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*)$ is a *CPA-secure symmetric encryption scheme* (see, e.g., [80]) if for every $K, M \in \{0,1\}^*$ we have that $\mathsf{Dec}(K, \mathsf{Enc}(K, M)) = M$. Moreover, we require that for every poly-time machine $\mathcal{D}_1^{\mathsf{cpa}}$, that takes as input 1^κ, and outputs $S^0, S^1, Y \in \{0,1\}^*$ (such that $|S^0| = |S^1|$) an every every poly-time machine $\mathcal{D}_2^{\mathsf{cpa}}$ it holds that:

$$\Pr\Big[\mathcal{D}_2(Y, \mathsf{Enc}(K, S^0)) = 1\Big] - \Pr\Big[\mathcal{D}_2(Y, \mathsf{Enc}(K, S^1)) = 1\Big] \leqslant \mathsf{negl}(\kappa),$$

where $K \leftarrow\!\!\$\ \{0,1\}^\kappa$. We will also use the following lemma:

Lemma 12. *Let $\kappa, \zeta \in \mathbb{N}$ be arbitrary parameters (where ζ can be a function of κ). Let $U_1, \ldots, U_C$ be independent random variables distributed over $\{0,1\}$ and such that for each i we have that $\Pr[U_i = 1] = 2^{-\zeta}$. Let $U := U_1 + \cdots + U_C$. We have that (a) if $C = \kappa \cdot 2^{\zeta+1}$ then $\Pr[U \leqslant \kappa] \leqslant \mathsf{negl}(\kappa)$ and (b) if $C \leqslant \kappa \cdot 2^{\zeta-1}$ then $\Pr[U \geqslant \kappa] \leqslant \mathsf{negl}(\kappa)$.*

Proof. We will use the following versions of the Chernoff–Hoeffding bounds (see, e.g., Thm. 1.1 in [47]).

Lemma 13. *Let $U_1, \ldots, U_C$ be random variables independently distributed over $[0,1]$ and let $U = U_1 + \cdots + U_C$. Then for all $\varepsilon > 0$ we have*

(a) $\Pr[U < (1-\varepsilon)\,\mathbb{E}[U]] \leqslant \exp(-\varepsilon^2 \cdot \mathbb{E}[U]/2)$ *and*

(b) $\Pr[U > (1+\varepsilon)\,\mathbb{E}[U]] \leqslant \exp(-\varepsilon^2 \cdot \mathbb{E}[U]/3)$.

Clearly each $\mathbb{E}[U_i] = 2^{-\zeta}$ and hence $\mathbb{E}[U] = C \cdot 2^{-\zeta}$. Let us start with proving Point (a). Assume $C = \kappa \cdot 2^{\zeta+1}$. This implies that $\mathbb{E}[U] > \kappa \cdot 2^{\zeta+1} \cdot 2^{-\zeta} = 2\kappa$. Set $\varepsilon := 1/4$. We get

$$\begin{aligned}
\Pr[U \leqslant \kappa] &\leqslant \Pr[U \leqslant (1/2) \cdot \mathbb{E}[U]] && (3.1)\\
&= \Pr[U < (1-\varepsilon)\,\mathbb{E}[U]] && (3.2)\\
&\leqslant \exp(-\varepsilon^2 \cdot \mathbb{E}[U]/2) && (3.3)\\
&= \exp(-\kappa/16) && (3.4)\\
&\leqslant \mathsf{negl}(\kappa). && (3.5)
\end{aligned}$$

Hence Point (a) of Lemma 12 is proven. Let us now proceed to proving Point (b). Assume $C \leqslant \kappa \cdot 2^{\zeta-1}$. This implies that

$$\kappa > C \cdot 2^{-\zeta+1} \qquad (3.6)$$

Set $\varepsilon := \sqrt{9 \cdot \kappa \cdot 2^{\zeta-5}/C}$. Using the assumption that $C \leqslant \kappa \cdot 2^{\zeta-1}$ we get that

$$\begin{aligned}
\varepsilon &> \sqrt{9 \cdot \kappa \cdot 2^{\zeta-5}/(\kappa \cdot 2^{\zeta-1})} && (3.7)\\
&= 3/4 && (3.8)
\end{aligned}$$

We hence get

$$\begin{aligned}
\Pr[U \geqslant \kappa] &\leqslant \Pr\left[U \geqslant C \cdot 2^{-\zeta+1}\right] && (3.9)\\
&= \Pr[U \geqslant (1-1/2)\,\mathbb{E}[U]] && (3.10)\\
&\leqslant \Pr[U > (1-\varepsilon)\,\mathbb{E}[U]] && (3.11)\\
&\leqslant \exp(-(9 \cdot \kappa \cdot 2^{\zeta-5}/C) \cdot (C \cdot 2^{-\zeta})/2) && (3.12)\\
&= \exp(-9 \cdot \kappa \cdot 2^{-6}) && (3.13)\\
&\leqslant \mathsf{negl}(-\kappa), && (3.14)
\end{aligned}$$

where in Eq. (3.9) we used Eq. (3.6), in Eq. (3.11) — Eq. (3.8), and in Eq. (3.12) — Lemma 13 (Point (b)). Hence Point (b) of Lemma 12 is proven. □

3.3 The model

This section provides more details on the model already informally introduced in Sect. 3.1.1. All protocols are executed in an asynchronous model. Every protocol is parameterized by a security parameter 1^κ and a hash function H that is modeled differently in the honest and adversarial executions. In the honest execution, the parties access H in a black-box way (via a standard random oracle [14]), except for Sect. 3.4.4, where a "circuit access" to H is needed (in the construction of zkPIK).

In the adversarial model, the malicious parties access H via an interactive machine Ω_H, which chooses a random function $H : \{0,1\}^\lambda \to \{0,1\}^\kappa$ (where λ and κ are parameters of the model) and interacts with parties $\mathcal{A}_1, \ldots, \mathcal{A}_a$ by accepting queries of a form $(x, mode)$, where $x \in \{0,1\}^*$ and $mode \in \{\mathsf{fast}, \mathsf{slow}\}$. If $mode = \mathsf{fast}$, then a query is called *fast*. It is called *slow* otherwise. Each query coming from a party $\mathcal{A}_b$ is answered to $\mathcal{A}_b$ with $H(x)$ (we also say that $\mathcal{A}_b$ *evaluated* H on input x). The execution of a protocol is divided into the *pre-computation* phase and the *online* phase, happening one after another. We say that Ω_H *is σ-bounded* if the total number of slow queries answered in the online phase is at most σ. The queries that exceed this quota are answered with $\perp$. The total number of fast queries is only bounded by the time complexity of the adversaries (i.e., it is polynomial in κ).

The main idea is that the fast queries are "cheap" and the participants will be allowed to send much more of them than the "expensive" slow queries (which correspond to queries computed using MPC/TEE techniques). On the other hand, when analyzing what the adversarial parties learned from the execution (i.e. when defining the "knowledge extractors", see below), only the fast queries will count. Namely, only the inputs to such queries will be considered known to the querying party. The distinction between the pre-computation phase and the online phase serves to model the fact that before the protocol starts, the sub-adversaries have an unbounded (but polynomial) time and can execute any distributed protocol. Note that the *mode* flag is only used for "accounting" purposes: the actions Ω_H do not depend on the value of this flag, except when defining the available budget of queries.

At the end of the execution of a protocol, we look at the information each party received as a result of the fast oracle queries. We define the *local hash transcript of a party* $\mathcal{A}_b$ to be the sequence $\mathcal{T}_b$ of hash inputs that Ω_H received from $\mathcal{A}_b$ (in the same order in which they were received). Let $\mathcal{T}_b^{\mathsf{fast}}$ be the sub-sequence of $\mathcal{T}_b$ containing only the inputs corresponding to fast queries (call it *local fast-hash transcript of a party* $\mathcal{A}_b$). A *(knowledge) extractor* $\mathcal{E}$ is a deterministic poly-time machine that takes $\mathcal{T}_b^{\mathsf{fast}}$ as input and produces as output a finite set $\mathcal{E}(\mathcal{T}_b^{\mathsf{fast}}) \subset \{0,1\}^*$. The extractors have block-box access to H (via the same oracle Ω_H, only using the fast queries).

For future reference, also define the *global hash transcript* to be the sequence $\mathcal{T}$ of hash inputs that Ω_H received (in the same order in which they were received). Let the *global fast-hash transcript* be the sub-sequence $\mathcal{T}^{\mathsf{fast}}$ of $\mathcal{T}$ containing only the inputs corresponding to fast queries.

3.4 Proofs of Individual Knowledge

We now provide formal details of the definition and the construction that were informally presented in Sect. 3.1.

3.4.1 Definition

A *Proof of Individual Knowledge (PIK)* is a protocol $\pi_{\mathsf{PIK}}^{\rho,\sigma}$ between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ (also denoted $(\mathcal{P} \leftrightarrows \mathcal{V})$), which are probabilistic poly-time machines with access to a hash function $H : \{0,1\}^\lambda \to \{0,1\}^\kappa$. The prover and the verifier take as input

a pair $(1^\kappa, S)$, where 1^κ is a security parameter and $S \in \{0,1\}^*$. For $\eta_\mathcal{P}, \eta_\mathcal{V} \in \mathbb{N}$, we say that the prover and the Verfier are $\eta_\mathcal{P}$- and $\eta_\mathcal{V}$-*bounded* (respectively) if the prover and the Verfier make at most $\eta_\mathcal{P}$ and $\eta_\mathcal{V}$ evaluations of H (respectively). We require that the protocol is *complete*, i.e., if both parties are honest (and their inputs are as above), then with overwhelming probability, the verifier outputs yes (in which case we also say that $\mathcal{V}$ *accepts*).

The second required property is *soundness*. Define a (ρ, σ)-*distributed adversary* to be a tuple $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ of poly-time interactive *sub-adversaries*. The honest verifier $\mathcal{V}$ receives $(1^\kappa, S)$ as input and interacts with $\mathcal{A}_1$ that also receives $(1^\kappa, S)$ as input. Intuitively, $\mathcal{A}_1$ plays the role of the (malicious) prover from the point of view of the verifier $\mathcal{V}$. Initially, the sub-adversaries can run a per-computation phase, where they can send an arbitrary number of slow queries to the oracle Ω_H. Then, the protocol starts, and the sub-adversaries enter the online phase in which they can make at most σ queries to the oracle Ω_H. The adversaries run in at most ρ rounds, where each of them has the following form: (1) each sub-adversary $\mathcal{A}_b$ performs some local computation, at the end of which $\mathcal{A}_b$ outputs a string Str_b, and (2) each Str_b is delivered to every other sub-adversary $\mathcal{A}_{\widehat{b}}$. We emphasize that we do not restrict the size of the communicated messages.

Consider an execution of a distributed adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ against an honest verifier (on input $(1^\kappa, S)$). Define $\mathsf{exec}((\mathcal{A}_1, \ldots, \mathcal{A}_a) \leftrightarrows \mathcal{V}); 1^\kappa; S)$ to be equal to $(\mathcal{T}_1^{\mathsf{fast}}, \ldots, \mathcal{T}_a^{\mathsf{fast}}, Out_\mathcal{V})$, where each $\mathcal{T}_b^{\mathsf{fast}}$ is the local fast-hash transcript of $\mathcal{A}_b$ and $Out_\mathcal{V} \in \{\mathsf{yes}, \mathsf{no}\}$ is the output of $\mathcal{V}$. We now have the following.

Definition 3.4.1. *We say that $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ is a* PIK protocol sound against (ρ, σ)-distributed adversary *if there exists knowledge extractors $\mathcal{E}_1, \ldots, \mathcal{E}_a$ such that for every (ρ, σ)-distributed adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_a)$ and every $S \in \{0,1\}^*$ we have that*

$$\Pr[Out_\mathcal{V} = \mathsf{yes} \text{ and } S \notin \mathcal{E}_1(\mathcal{T}_1^{\mathsf{fast}}) \cup \cdots \cup \mathcal{E}_a(\mathcal{T}_a^{\mathsf{fast}})] \leqslant \mathsf{negl}(\kappa), \tag{3.15}$$

where $(\mathcal{T}_1^{\mathsf{fast}}, \ldots, \mathcal{T}_a^{\mathsf{fast}}, Out_\mathcal{V}) \leftarrow\!\!\$\ \mathsf{exec}((\mathcal{A}_1, \ldots, \mathcal{A}_a) \leftrightarrows \mathcal{V}); 1^\kappa; S)$. We say that $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ has extraction efficiency $(\alpha_\mathsf{O}, \alpha_\mathsf{T}, \alpha_\mathsf{S})$ *if $|\mathcal{E}_1(\mathcal{T}_1^{\mathsf{fast}}) \cup \cdots \cup \mathcal{E}_a(\mathcal{T}_a^{\mathsf{fast}})| \leqslant \alpha_\mathsf{O}$, and $\mathcal{E}$ operates in time at most α_T and uses space at most α_S. The parameters $\alpha_\mathsf{O}, \alpha_\mathsf{T}$ and α_S can be functions of some other parameters in the system.*

The idea behind α_O is that $\alpha_\mathsf{O} - 1$ is the number of "false positives", i.e., values in $\mathcal{E}_1(\mathcal{T}_1^{\mathsf{fast}}) \cup \cdots \cup \mathcal{E}_a(\mathcal{T}_a^{\mathsf{fast}})$ that are not equal to S. Obviously, the smaller α_O is, the better. In Sect. 3.4.4, we describe a method that, with high probability, allows to reduce the number of such false positives to 0 in many applications.

3.4.2 Construction

In this section, we present our construction of a PIK protocol $\pi^{\rho,\sigma}_{\mathsf{PIK}}$, which was already informally described in Sect. 3.1.2. The protocol is parameterized by a "moderate hardness parameter" $\zeta \in \mathbb{N}$ and a security parameter 1^κ. It uses a hash function $H : \{0,1\}^\lambda \to \{0,1\}^\kappa$ with $\lambda \geqslant 2\kappa$. In practice, H could be a compression function of a popular hash function. The protocol participant takes as input $S \in \{0,1\}^*$ and 1^κ. We assume that $|S|$ is a multiple of $\lambda - \kappa - 2$ (if it is not the case, then one can pad S with zeros). Let $S_1, \ldots, S_n$ be such that $S = (S_1 \,||\, \cdots \,||\, S_n)$ where $|S_1| = \cdots = |S_n| = \lambda - \kappa - 2$.

Our PIK protocol is depicted in Fig. 3.2. It uses a sub-routine $\mathsf{scratch}$ (see also Fig. 3.1 and Fig. 3.3 for a graphical representation of the computation of $\mathsf{scratch}$). The protocol starts with the verifier sending a random challenge $Z \in \{0,1\}^\kappa$. Then, the goal of the prover is to find κ nonces $W^1, \ldots, W^\kappa \in \{0,1\}^{\lambda-\kappa-2}$ that convince the verifier that the prover did a substantial amount of work for the challenge Z. This is done by requiring that the output of every $\mathsf{scratch}(S, Z, W^i)$ starts with ζ zeros. We have the following definition.

Definition 3.4.2. *We call a given* $\mathsf{scratch}(S, Z, W)$ successful *if its output starts with* ζ *zeros (cf. Step 2b on Fig. 3.2).*

This is similar to the Bitcoin mining procedure, except that we require κ nonces to be found (in Bitcoin $\kappa = 1$) in order to reduce the variance of the success probability. The main idea behind $\mathsf{scratch}$ is that it forces the prover to sequentially compute d times H on every block S_j of S. We also force the prover to compute $\mathsf{scratch}(S, Z, W)$ on a large number of W's. The search for the W^i's can be fully parallelized. The only non-parallelizable part is the $\mathsf{scratch}$ procedure, which takes as input (S, Z, W), where S is the message, Z is the challenge, and W is the nonce.

The inputs of H in $\mathsf{scratch}$ start with two bits: 00 indicates that the hash is computed on (Z, W) (where Z is a challenge and W is a nonce), 01 indicates that the first block (S_1) is hashed (together with some Q), and 10 indicates that the hashed value S_j is one of the subsequent blocks (i.e., $j > 1$). Labels 00, 01, and 10 are used to help the extractor find S (e.g. if the extractor sees that some $(01 \,||\, \widehat{S}_1 \,||\, \widehat{Q}_1)$ was hashed then the extractor guesses that $\widehat{S}_1$ is the first block of S and starts searching for a hash of a form $(10 \,||\, \widehat{S}_2 \,||\, \widehat{Q}_2)$ with $\widehat{Q}_2 = H(01 \,||\, \widehat{S}_1 \,||\, \widehat{Q}_1)$ (see Proof of Thm. 14 for the details). Note that in total $\mathsf{scratch}$ computes $nd + 1$ hashes H. Security of $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ is stated in the following theorem.

Theorem 14. *Let* κ, n, d *be as above and let* $\eta_{\mathcal{P}} := (nd+1) \cdot 2^{\zeta+1} \cdot \kappa$ *and* $\eta_{\mathcal{V}} := (nd+1) \cdot \kappa$. *Assume* $\sigma \leqslant \kappa \cdot 2^{\zeta-3}$ *and* $\rho \leqslant d - 1$. *Then* $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ *from Fig. 3.2 is a PIK protocol with* $\eta_{\mathcal{P}}$*-bounded prover and* $\eta_{\mathcal{V}}$*-bounded verifier that is sound against a* (ρ, σ)*-distributed adversary with extraction efficiency* $(\alpha_{\mathsf{O}}, \alpha_{\mathsf{T}}, \alpha_{\mathsf{S}})$, *where*

$$\mathbb{E}[\alpha_{\mathsf{O}}] = 2^{-\zeta} \cdot \ell, \; \mathbb{E}[\alpha_{\mathsf{T}}] = O(\ell_b), \text{ and } \mathbb{E}[\alpha_{\mathsf{S}}] = O(2^{-\zeta} \cdot \ell_b \cdot |S|).$$

Above, ℓ_b *is the number of hashes computed by a sub-adversary* $\mathcal{A}_b$, *and* $\ell := \ell_1 + \cdots + \ell_a$. *The unit of time is a computation of* H.

The proof of Thm. 14 is presented in Sect. 3.4.3. Before proceeding to it, let us comment on the parameters in the lemma statement. When it comes to the parameters of the honest prover and verifier, the most important ones are those denoting the "budget" for the hash computations, i.e., $\eta_{\mathcal{P}} = (nd + 1) \cdot 2^{\zeta+1} \cdot \kappa$ and $\eta_{\mathcal{V}} = (nd + 1) \cdot \kappa$ respectively. Note that each computation of the $\mathsf{scratch}$ procedure requires $(nd+1)$ hash computations. The verifier needs to do such a computation κ times; hence, she needs to perform only $(nd + 1) \cdot \kappa$ hashes. For the prover, observe that each scratch attempt succeeds with probability $2^{-\zeta}$ (by "succeeding" we mean finding a value that starts with ζ zeros). Since the prover needs to be successful κ times, she needs, on average, $(nd + 1) \cdot 2^{\zeta} \cdot \kappa$ scratch attempts. We set $\eta_{\mathcal{P}}$ to be the double of this parameter in order to make the probability that he is successful (see Def. 3.4.2) less than κ times exponentially small.

Procedure $\mathsf{scratch}(S, Z, W)$

1. Assume $S = (S_1 \,||\, \cdots \,||\, S_n)$, where each $|S_j| = \lambda - \kappa - 2$.
2. For $k = 1$ to d do:

   For $j = 1$ to n do:

$$Q_j^k := \begin{cases} H(00 \,||\, Z \,||\, W) & \text{if } k = 1 \text{ and } j = 1 \\ H(10 \,||\, S_n \,||\, Q_n^{k-1}) & \text{if } k \neq 1 \text{ and } j = 1 \\ H(01 \,||\, S_1 \,||\, Q_1^k) & \text{if } j = 2 \\ H(10 \,||\, S_{j-1} \,||\, Q_{j-1}^k) & \text{if } j > 2 \end{cases}$$

3. Output $H(10 \,||\, S_n \,||\, Q_n^d)$.

Figure 3.1: Sub-routine scratch that defines a sequence of hash computations needed to finish a single scratch attempt on input message S, challenge Z and a nonce W.

Observe also that if we substitute $2^{-\zeta}$ with $2 \cdot \kappa \cdot (nd+1)/\eta_{\mathcal{P}}$ then the formula $2^{-\zeta} \cdot \ell$ (appearing both in the bounds on $\mathbb{E}[\alpha_{\mathsf{O}}]$ and $\mathbb{E}[\alpha_{\mathsf{S}}]$) can be rewritten as: $2 \cdot \kappa \cdot (nd+1) \cdot \ell/\eta_{\mathcal{P}}$. It is interesting to look at the last factor, i.e., $\ell/\eta_{\mathcal{P}}$. Recall that ℓ is the number of hashes computed by the adversary $\mathcal{A}$. On the other hand, $\eta_{\mathcal{P}}$ is the maximal number of hashes computed by the honest prover. Hence, $\ell/\eta_{\mathcal{P}}$ corresponds to the multiplicative advantage of the adversary with respect to the honest prover, or, in other words, it answers the question "How much more computing power does the adversary have compared to the honest prover?". In our theorem, the bounds on α_{O} and α_{S} grow linearly with $\ell/\eta_{\mathcal{P}}$. Intuitively, this comes from the fact that a powerful adversary can always "mislead" the extractor by executing a large number of scratch procedures on $\widehat{S} \neq S$.

3.4.3 Proof of Thm. 14

Completeness.

Let us start with proving completeness. We first upper-bound the probability that the protocol halts in Step 2 due to the fact that the budget for H computations was exhausted. Note that this budget allows the prover to evaluate scratch $\lfloor \eta_{\mathcal{P}}/(nd+1) \rfloor = 2^{\zeta+1} \cdot \kappa$ times (since each scratch execution takes $nd+1$ hash evaluations). For $i = 1, \ldots, 2^{\zeta+1} \cdot \kappa$, let $U_i \in \{0, 1\}$ be equal to 1 if and only if the ith scratch execution was successful, i.e., it produced an output that starts with κ zeros. Set $U = U_1 + \cdots + U_{2^{\zeta+1} \cdot \kappa}$. For simplicity of the analysis assume that $\pi_{\mathsf{PIK}}^{\rho,\sigma}$ does not stop once all the W^j's are found. Clearly, the U_i's are independent. Moreover, we have that each $\Pr[U_i = 1] = 2^{-\zeta}$ (the probability that a random string starts with ζ zeros) and hence each $\mathbb{E}[U_i] = 2^{-\zeta}$. Therefore by Lemma 12 (Point (a)), we get that $\Pr[U < \kappa] \leqslant \mathsf{negl}(\kappa)$. Therefore with overwhelming probability the prover finds κ values W such that $\mathsf{scratch}(S, Z, W)$ starts with ζ zeros without exhausting her hash budget, and hence she does not output $\perp$. It is also easy to see that the Verifer always accepts in such a case (since she just repeats the same scratch

Protocol $\pi^{\rho,\sigma}_{\mathsf{PIK}}$

The protocol is parameterized with a "moderate hardness" parameter $\zeta \leqslant \kappa$. It is executed between the $\eta_{\mathcal{P}}$-bounded prover $\mathcal{P}$ and the $\eta_{\mathcal{V}}$-bounded verifier $\mathcal{V}$. Both parties take as input $(1^\kappa, S)$, where $S = (S_1, \ldots, S_n)$.

1. The verifier $\mathcal{V}$ chooses a random *challenge* $Z \in \{0,1\}^\kappa$ and sends it to the prover $\mathcal{P}$.
2. The prover $\mathcal{P}$ does the following **parallel search** across different values of $i = \{1, \ldots, \kappa\}$ and *nonces* $W^i \in \{0,1\}^{\lambda-\kappa-2}$:
   (a) Let $Q^i := \mathsf{scratch}(S, Z, W^i)$.
   (b) If Q^i starts with ζ zeros then record W^i and **stop the parallel search**.

   The above search is done as long as the prover did not exhaust her budget for computing hashes (recall that she can make at most $\eta_{\mathcal{P}}$ of them). If this happens before the search is over, then $\mathcal{P}$ outputs $\perp$ to $\mathcal{V}$, who also outputs $\perp$, and then both halt. Otherwise, we proceed to the next step.
3. The prover sends $(W^1, \ldots, W^\kappa)$ to the verifier.
4. Upon receiving $(W^1, \ldots, W^\kappa)$ the verifier outputs yes if for *all* $i \in \{1, \ldots, \kappa\}$ it holds that the output of $\mathsf{scratch}(S, Z, W^i)$ starts with ζ zeros. Otherwise, the verifier outputs no.

Figure 3.2: Proof of individual knowledge (PIK) $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ that satisfies Def. 3.4.1 (see Thm. 14). The main procedure uses a sub-routine $\mathsf{scratch}$ depicted in the Figure 3.1.

computation as the prover for the W^i values that she received in Step 4, and $\eta_{\mathcal{V}}$ hash evaluations are needed for this computation).

Soundness.

To show soundness, let $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ be a (ρ, σ)-distributed adversary. Consider an execution of $\mathcal{A}$ against an honest verifier on input $(1^\kappa, S)$ and let Z be the challenge that the verifier sends in Step 1. Recall that the adversary has access to a σ-bounded oracle Ω_H that she can use to evaluate hash function H. For an arbitrary $\widehat{S}$ (necessarily equal to S), such that $\widehat{S} = (\widehat{S}_1 \,||\, \cdots \,||\, \widehat{S}_n)$ (with each $\widehat{S}_j \in \{0,1\}^{\lambda-\kappa-2}$) and an arbitrary $\widehat{Q}_1 \in \{0,1\}^\kappa$ a *trace on $\mathcal{T}$ for $\widehat{S}$ (starting with $\widehat{Q}_1$)* is a sequence $(i_1, \ldots, i_n)$ such that

- $\mathcal{T}[i_1] = (01 \,||\, \widehat{S}_1 \,||\, \widehat{Q}_1)$ and
- for every $m = 2, \ldots t$ we have: $\mathcal{T}[i_1] = (10 \,||\, \widehat{S}_m \,||\, H(\mathcal{T}[i_{m-1}]))$.

Intuitively a trace is a sequence of indices on $\mathcal{T}$ that "look like" an attempt to evaluate a column on Fig. 3.3 for *some* $\widehat{S}$. For any nonce W, an *S-scratch on $\mathcal{T}$ for W* is a sequence

$$i_0 \,||\, (i_1^1, \ldots, i_n^1) \,||\, \cdots \,||\, (i_1^d, \ldots, i_n^d), \tag{3.16}$$

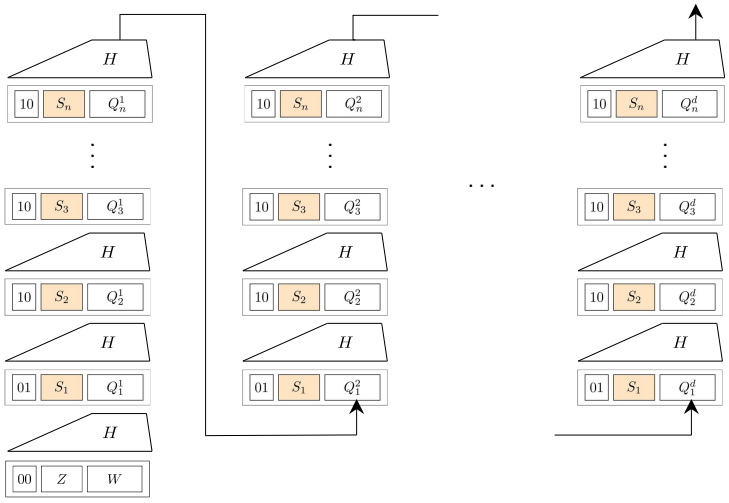


Figure 3.3: A diagram representing an execution of $\mathsf{scratch}((S_1, \ldots, S_n), Z, W)$ procedure from Fig. 3.1.

where $\mathcal{T}[i_0] = (00||Z||W)$ and each $(i_1^k, \ldots, i_n^k)$ is a trace for S staring either with $H(\mathcal{T}[i_0])$ (if $k = 1$) or with $H(10 \,||\, S_n \,||\, H(\mathcal{T}[i_n^{k-1}]))$ (otherwise). In other words: S-scratch is a sequence of indices on $\mathcal{T}$ that correspond to an attempt to evaluate the entire diagram on Fig. 3.3 for $\widehat{S} = S$.

Claim 1. *For a hash transcript $\mathcal{T}$ consider two distinct nonces W and $\widehat{W}$. Let I and $\widehat{I}$ be S-scratches on $\mathcal{T}$ for W and $\widehat{W}$, respectively. Then with overwhelming probability, we have that I and $\widehat{I}$ are disjoint, i.e., there does not exist an index i that appears in both I and $\widehat{I}$.*

sketch. Let Q_i^k's and $\widehat{Q}_i^k$'s be the values of the variables in the $\mathsf{scratch}$ procedure when run on input (S, Z, W) and $(S, Z, \widehat{W})$, respectively (see Fig. 3.2 and 3.1 or the diagram on Fig. 3.3). Suppose there exists an index i that appears in both I and $\widehat{I}$. Obviously $\mathcal{T}[i]$ cannot be of a form $(00 \,||\, Z \,||\, W)$, since $W \neq \widehat{W}$. Hence $\mathcal{T}[i]$ must have a form $(b_0 \,||\, b_1 \,||\, S_j \,||\, Q)$. This means that $\mathcal{T}[i]$ is an output of a sequence of hashes starting with $H(00 \,||\, Z \,||\, W)$ *and* simultaneously, it is an output of a sequence of hashes starting with $H(00 \,||\, Z \,||\, \widehat{W})$. Using this information, one can efficiently find a collision in H. Since the probability of finding collisions in H is negligible, the probability that such W and $\widehat{W}$ can be found has to be negligible. □

We will need the following definition.

Definition 3.4.3. *Let W be a nonce such that there exists an S-scratch on $\mathcal{T}$ for W. We call W* fast *if for every S-scratch $(i_1, \ldots, i_m)$ on $\mathcal{T}$ for W each $\mathcal{T}[i_j]$ is a fast query. Otherwise, we call it* slow*.*

We will also use the following simple fact that states that with overwhelming probability, the adversary has to compute the Q_j^k variables in the same order as the honest party executing the $\mathsf{scratch}$ procedure.

Claim 2. *Fix some $Z \in \{0,1\}^\kappa$ and $W \in \{0,1\}^{\lambda-\kappa-2}$ and let S be as above. Let Q_j^k's be the variables computed in the* $\mathsf{scratch}(S, Z, Q)$ *procedure (see Fig. 3.1). For a global hash transcript $\mathcal{T}$*

- *let i_0^1 be the first position in $\mathcal{T}$ on which $(00 \,||\, Z \,||\, W)$ appears, and*
- *for $k \in \{1, \ldots, d\}$ and $j \in \{1, \ldots, n\}$ let i_j^k be the first position in $\mathcal{T}$ such that $\mathcal{T}[i_j^k]$ ends with Q_j^k.*

Then with overwhelming probability the i_j^k's when sorted lexicographically by (k, j) are monotonically increasing, i.e.,

$$\underset{\substack{(k_0,j_0)\\(k_1,j_1)}}{\forall} k_0 < k_1 \vee ((k_0 = k_1) \wedge (j_0 < j_1)) \text{ implies that } i_{j_0}^{k_0} < i_{j_1}^{k_1}. \tag{3.17}$$

sketch. Let $\prec$ be the strict lexicographic order on (k, j)'s used in Eq. (3.17). Suppose that there exists $(k_0, j_0) \prec (k_1, j_1)$ such that $i_{j_0}^{k_0} \geqslant i_{j_1}^{k_1}$. Clearly, unless a collision in H is found (which happens with negligible probability), all the Q_j^k's are distinct, and hence we can assume that $i_{j_0}^{k_0} > i_{j_1}^{k_1}$. Without loss of generality assume that (k_0, j_0) immediately precedes (k_1, j_1) in the "$\prec$" order. Observe that $Q_{j_1}^{k_1}$ is an output of a hash function H when evaluated on an input that contains $Q_{j_0}^{k_0}$ (with a convention that $Q_0^1 := 00 \,||\, Z \,||\, W$). But $i_{j_0}^{k_0} > i_{j_1}^{k_1}$ means that $Q_{j_1}^{k_1}$ was submitted to the oracle *before* the adversary learned $Q_{j_0}^{k_0}$. Since the outputs of a random oracle are uniform over $\{0,1\}^\kappa$, this happens with negligible probability. This finishes the proof of the claim.

□

Claim 3. *Suppose the number of fast W's is at most $\kappa \cdot 2^{\zeta-3}$. Then the probability that $\mathcal{V}(1^\kappa, S)$ accepts on input S is negligible.*

Proof. Recall that we assumed that $\sigma \leqslant \kappa \cdot 2^{\zeta-3}$ and in Claim 1 we have shown that for distinct W and $\widehat{W}$ the S-scratches for W and $\widehat{W}$ are disjoint. Since $Z \in \{0,1\}^\kappa$ is sampled by the prover in the online phase, thus with overwhelming probability $(00 \,||\, Z \,||\, W)$ is sent to the oracle in the online phase. By Claim 2, this means that with overwhelming probability, all of the S-scratches on T for W contain only queries sent to the oracle in the online phase. Thus, the total number of slow W's is at most $\kappa \cdot 2^{\zeta-3}$. Adding the fast ones, we get that the total number C of W's for which an S-scratch in $\mathcal{T}$ exists is at most $\kappa \cdot 2^{\zeta-2}$. Let $W^1, \ldots, W^C$ be these nonces.

By Claim 2, with overwhelming probability, the adversary *first* needs to compute the entire S-scratch before learning whether it was successful or not. For each $i \in \{1, \ldots, C\}$ let $U_i \in \{0,1\}$ be equal to 1 if and only if $\mathsf{scratch}(S, Z, W^i)$ starts with ζ zeros. Clearly, the U_i's are independent and $\Pr[U_i = 1] = 2^{-\zeta}$. Let $U = U_1 + \cdots + U_C$. Using Lemma 12, we get that $\Pr[U \geqslant \kappa/2] \leqslant \mathsf{negl}(\kappa)$. Recall that $\mathcal{V}$ accepts only if she receives $(W^1, \ldots, W^\kappa)$ such that each $\mathsf{scratch}(S, Z, W^i)$ starts with ζ zeros. From the analysis above, we get that with overwhelming probability, there exists $\kappa - \kappa/2 = \kappa/2$ values W_i such that the corresponding value $(10 \,||\, S_n \,||\, Q_n^d)$ is not in $\mathcal{T}$, or, in other words, H was not evaluated on it. Clearly, the probability that $H(10 \,||\, S_n \,||\, Q_n^d)$ starts with ζ zeros for all such W_i's is equal to $(2^{-\zeta})^{\kappa/2} = 2^{-\zeta \cdot \kappa/2} \leqslant \mathsf{negl}(\kappa)$.

□

Before presenting our extractor $\mathcal{E}$, consider the following natural construction idea. By Claim 3, if the adversary was successful, then $\mathcal{T}^{\mathsf{fast}}$ needs to contain at least $\kappa \cdot 2^{\zeta-3}$ S-scratches of a computation of $\mathsf{scratch}(S, Z, W)$ (for some W's). Each such an S-scratch contains $d = \rho + 1$ traces for S, where ρ is the maximal number of communication rounds. Therefore every S-scratch contains a trace computed in a single round. As we show formally later (see Claim 4), all the queries corresponding to such a trace had to come from a single adversary. Hence, it should be possible to find it by scanning all $\mathcal{T}_b^{\mathsf{fast}}$'s.

More concretely, consider an extractor that scans the transcript $\mathcal{T}_b^{\mathsf{fast}}$ in order to find elements of a form $(01 \,||\, \widehat{S}_1 \,||\, \widehat{Q}_1)$ (for some $\widehat{S}_1$ and $\widehat{Q}_1$). Each such element is potentially part of an attempt by $\mathcal{A}_b$ to compute a scratch path on a message starting with $\widehat{S}_1$. The extractor stores $\widehat{S}_1$ and $\widehat{Q}_2 := H(01 \,||\, \widehat{S}_1 \,||\, \widehat{Q}_1)$. It then continues scanning the transcript to find elements of a form $(10 \,||\, \widehat{S}_2 \,||\, \widehat{Q}_2)$. Suppose $\mathcal{E}$ found such an element, and let $Q_3 := H(10 \,||\, \widehat{S}_2 \,||\, \widehat{Q}_2)$. It then looks for elements of a form $(10 \,||\, \widehat{S}_2 \,||\, \widehat{Q}_3)$, and so on, until it finds n strings: $(\widehat{S}^1, \ldots, \widehat{S}^n)$. It then outputs $(\widehat{S}^1 \,||\, \cdots \,||\, \widehat{S}^n)$.

The above approach essentially works, although it needs some modifications. The main challenge is that the adversary can make "fake" calls to Ω_H, whose goal would be just to mislead the extractor. Hence, a hash transcript can contain $\widehat{S}$-scratches for some other $\widehat{S}$'s (possibly sharing a prefix with S). We deal with these problems by exploiting the fact that in a successful execution, the adversary still needs to perform a large number $(\kappa \cdot 2^{\zeta-3})$ of $\mathsf{scratch}$ executions on real S. Our solution is presented in Fig. 3.4. The $\mathcal{E}$ procedure maintains a set $\mathcal{F}$ containing information that can lead to finding the solution S. The values are added to this set only with a certain probability $(2^{-\zeta})$. This probability is chosen so that the most likely *some* S-scratch is included in $\mathcal{F}$, yet, it reduces the size of $\mathcal{F}$ significantly, leading to better parameters. In particular, it prevents the size of $\mathcal{F}$ from becoming too large if the adversary makes lots of "fake" hash queries described above.

Claim 4. *Consider an execution of $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_a)$ against an honest verifier $\mathcal{V}$ on input $(1^\kappa, S)$. For each b, let $\mathcal{T}_b^{\mathsf{fast}}$ be the fast-hash transcript of $\mathcal{A}_b$. Suppose $\mathcal{V}(1^\kappa, S)$ accepts. Then with overwhelming probability for some b, the string S is among the values output by the knowledge extractor $\mathcal{E}(\mathcal{T}_b^{\mathsf{fast}})$.*

Proof. Consider a global hash transcript $\mathcal{T}^{\mathsf{fast}}$. Let W be a fast W (see Def. 3.4.3). Recall that, since the adversary can evaluate H on the same value multiple times, there can be multiple S-scratches on $\mathcal{T}^{\mathsf{fast}}$ for W. Let $\mathsf{first\text{-}scratch}(W, \mathcal{T})$ denote the S-scratch composed of the *first* positions on $\mathcal{T}^{\mathsf{fast}}$ for W where the given value appears on $\mathcal{T}^{\mathsf{fast}}$. More formally, let $\mathsf{first\text{-}scratch}(W)$ be the S-scratch $(i_1, \ldots, i_m)$ for W such that for every i_j is the least index i such that $\mathcal{T}^{\mathsf{fast}}[i_j] = \mathcal{T}^{\mathsf{fast}}[i]$, i.e., we have:

$$\forall_{j \in \{1,\ldots,m\}} \; \forall_{i < i_j} \; \mathcal{T}^{\mathsf{fast}}[i] \neq \mathcal{T}^{\mathsf{fast}}[i_j]. \tag{3.18}$$

Recall that ρ is the number of rounds, and we assumed that $\rho \leqslant d-1$. Therefore for every fast W and $\mathsf{first\text{-}scratch}(W)$ of form as in Eq. (3.16), there needs to exist a round and an index k such that hashes of values $\mathcal{T}[i_1^k], \ldots, \mathcal{T}[i_n^k]$ were all sent to Ω_H in a single round. Observe that the knowledge of each $\mathcal{T}[i_j^k]$ is needed to compute the next value $\mathcal{T}[i_{j+1}^k]$ and, because of Eq. (3.18) $H(\mathcal{T}[i_j^k])$ was not computed in the previous round. Since the output of a hash can be guessed with probability $2^{-\kappa}$, thus with overwhelming probability, all

Knowledge extractor $\mathcal{E}(\mathcal{T}_b^{\mathsf{fast}})$

1. Let $\ell_b := |\mathcal{T}_b^{\mathsf{fast}}|$.
2. Let $\mathcal{F}$ be a set that is initially empty. Set $\mathcal{F}$ contains pairs of a form $(\widehat{Q}, \widehat{S})$, where $\widehat{Q} \in \{0,1\}^\kappa$ is a hash output, and $\widehat{S} \in \{0,1\}^*$.
3. For each $i = 1, \ldots, \ell_b$ do
   (a) If $\mathcal{T}_b^{\mathsf{fast}}[i]$ is of a form $(01||\widehat{S}||\widehat{Q})$ (for some $\widehat{S}$ and $\widehat{Q}$) then with probability $2^{-\zeta}$ add a pair $(H(01 \,||\, \widehat{S} \,||\, \widehat{Q}), \widehat{S})$ to $\mathcal{F}$ (if it is not stored there yet).
   Note that $(01||\widehat{S}||\widehat{Q})$ can appear more than once in $\mathcal{T}_b^{\mathsf{fast}}$. Since we want the probability of being added to $\mathcal{F}$ to be the same for every string $(01||\widehat{S}||\widehat{Q})$ (no matter how often it appears in $\mathcal{T}_b^{\mathsf{fast}}$), we perform this random choice by evaluating some function $\varphi : \{0,1\}^* \to \{0,1\}$ (treated as a random oracle) such that $\Pr[\varphi(x) = 1] = 2^{-\zeta}$. We add $(H(01 \,||\, \widehat{S} \,||\, \widehat{Q}), \widehat{S})$ to $\mathcal{F}$ if and only it $\varphi(01 \,||\, \widehat{S} \,||\, \widehat{Q}) = 1$.
   (b) If $\mathcal{T}_b^{\mathsf{fast}}[i]$ is of a form $(10||\widehat{S}||\widehat{Q})$ (for some $\widehat{S}$ and $\widehat{Q}$) and that there exists a pair $(\widehat{Q}, \widehat{S}') \in \mathcal{F}$ (for some $\widehat{S}' \in \{0,1\}^*$) then add $(H(10||\widehat{S}||\widehat{Q}), (\widehat{S}'||\widehat{S}))$ to $\mathcal{F}$.
   Additionally, if $|(\widehat{S}'||\widehat{S})| = n$ then **output** $(\widehat{S}'||\widehat{S})$ (but do not terminate),

Figure 3.4: The knowledge extractor for our PIK protocol. It preoceeds in an online fashion, reading $\mathcal{T}_b^{\mathsf{fast}}$ and outputting during the execution the "candidate" values for S. The output of $\mathcal{E}(\mathcal{T}_b^{\mathsf{fast}})$ is the set of all values that were output during the execution of $\mathcal{E}$.

the values $\mathcal{T}[i_1^k], \ldots, \mathcal{T}[i_n^k]$ had to be submitted to Ω_H by single sub-adversary $\mathcal{A}_b$. Hence there must exist a trace $i_1^k, \ldots, i_n^k$ in $\mathsf{first\text{-}scratch}(W)$ that was computed by a single sub-adversary $\mathcal{A}_b$.

By Claim 3, with overwhelming probability, the total number of fast W's is greater than $\kappa \cdot 2^{\zeta-3}$. Hence, at least $\kappa \cdot 2^{\zeta-3}$ traces for S were computed by a single $\mathcal{A}_b$. The probability that each trace is added to $\mathcal{F}$ is $2^{-\zeta}$. Hence, the probability that at least one trace for S is output by *some* $\mathcal{A}_b$ is at least $1 - (1 - 2^{-\zeta})^{\kappa \cdot 2^{\zeta-3}}$, which, by Lemma 11, is at least $1 - e^{-\kappa/8} \geqslant 1 - \mathsf{negl}(\kappa)$. □

It remains to analyze the extraction efficiency of the extractor. Consider a run on $\mathcal{E}$ on input $\mathcal{T}_b$. We first show the following bound on the expected output size of $\mathcal{F}$:

$$\mathbb{E}[|\mathcal{F}|] \leqslant 2^{-\zeta} \cdot \ell_b \tag{3.19}$$

(where ℓ_b is the number of fast hash queries of $\mathcal{A}_b$). Recall that new values are added to $\mathcal{F}$ by scanning all ℓ_b positions on $\mathcal{T}_b^{\mathsf{fast}}$ in the following way:

1. if $\mathcal{T}_b^{\mathsf{fast}}[i]$ is of a form $(01 \,||\, \widehat{S} \,||\, \widehat{Q})$ then a new value is added to $\mathcal{F}$ with probability at most $2^{-\zeta}$ and

2. if $\mathcal{T}_b^{\mathsf{fast}}[i]$ is of a form $(10 \,||\, \widehat{S} \,||\, \widehat{Q})$ then a new value is added if $\widehat{Q}$ is a result of a chain of hashes that started with hashing some $(01 \,||\, \widehat{S} \,||\, \widehat{Q}')$ and $\varphi(01 \,||\, \widehat{S} \,||\, \widehat{Q}') = 1$. Since this happens with probability 2^{ζ} we get that the probability that $(10 \,||\, \widehat{S} \,||\, \widehat{Q})$ is added to $\mathcal{F}$ is $2^{-\zeta}$.

This proves (3.19). Since the values that are output by the extractor are a subset of $\mathcal{F}$, and $\ell = \ell_1 + \cdots + \ell_b$, thus we get that $\mathbb{E}[\alpha_{\mathsf{O}}] = \mathbb{E}\Big[\mathcal{E}_1(\mathcal{T}_b^{\mathsf{fast}}) \cup \cdots \cup \mathcal{E}_a(\mathcal{T}_b^{\mathsf{fast}})\Big] \leqslant 2^{-\zeta} \cdot \ell$.

To analyze the time and space complexity of $\mathcal{E}$ observe that $\mathcal{E}$ is an online algorithm that reads $\mathcal{T}_b^{\mathsf{fast}}$ once. Assume that $\mathcal{F}$ is maintained using Cuckoo Hashing [98], where in every pair $(\widehat{Q}, \widehat{S})$, the element $\widehat{Q}$ is treated as the key, and $\widehat{S}$ is the stored value. Thanks to this, looking up and inserting the values in $\mathcal{F}$ takes constant time. Thus, the expected time complexity α_{T} of each $\mathcal{E}$ is $O(\ell_b)$, and the expected space complexity α_{S} is $O(2^{-\zeta} \cdot \ell_b \cdot |S|)$. This concludes the proof.

3.4.4 Extensions

This section describes possible extensions of the basic PIK protocol from Sect. 3.4. We leave the formalization of these extensions for future work as formally modeling attacks in the network settings is often highly non-trivial. For example, it requires us to consider different attack scenarios (e.g., man-in-the-middle, replay attacks, etc.) and typically involves a detailed description of the attack model (including modeling the network and the environment).

PIK as building block.

As mentioned in Sect. 3.1, PIK is a primitive that will usually not be used in a "standalone" way but rather as a part of a more extensive system. The main reason for this is that the messages sent by the prover may provide some information about S. Indeed, the definition of PIK does not mention any privacy guarantees for the prover, neither against an external attacker nor against the verifier (who actually, in our setting from Def. 3.4.1, knows S entirely). In this section, we describe two extensions of PIK that address this problem. The first one ("Encrypted and Authenticated PIK") answers the issue of security against an external adversary, and the second one provides a version of PIK ("zero-knowledge PIK") that works against a verifier that does *not* know S. This version of PIK does not reveal any information about S to the verifier (or any other party).

Encrypted and Authenticated PIK (eaPIK).

Encrypted and Authenticated PIK (eaPIK) is a version of PIK where the communication between the prover and the verifier is secured using a symmetric key K. Let us describe this solution as a general transformation of any PIK protocol $\pi_{\mathsf{PIK}}^{\rho,\sigma}$. The main idea is quite straightforward: we let every message[2] in the protocol $\pi_{\mathsf{PIK}}^{\rho,\sigma}$ be encrypted and authenticated with a fresh key K that is shared between the prover and the verifier, sampled

[2]It is easy to see that in case of our protocol $\pi_{\mathsf{PIK}}^{\rho,\sigma}$ (see Fig. 3.2) the only message that may contain sensitive information about S is $(W^1, \ldots, W^{\kappa})$ sent by the prover to the verifier in Step 3. Hence, it is enough if only this message is encrypted.

independently from S. Note that since we have a setup phase (for establishing S), thus assuming that in this phase, we also generate K is very mild. To summarize: the client and the server share the following:

- key K for encrypting messages during the execution of the PIK protocol and
- secret S that is used as the input of both parties in PIK.

Of course, as long as the prover is honest (and hence: she stores S on a single machine), she easily convinces the verifier that she knows S. On the other hand, by the PIK properties, a dishonest prover cannot distribute S among different sub-adversaries. It is important to note that the PIK protocol only guarantees that S cannot be distributed, and K is not protected similarly. Hence, any potential application accessing the server should require the knowledge of S only, and it should not be assumed that K is stored on a single machine.

ZK-Proofs of Individual Knowledge.

Let us now describe a solution for a setting where S is known only to the prover. In this case, PIK makes sense only if some publicly-available information on S is known. We model this in the following standard way. Suppose L is an NP-language characterized by some NP-relation $\varphi : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$. The verifier holds some public information $pub \in \{0,1\}^*$, while the prover has an NP-witness $S \in \{0,1\}^*$ such that $(S, pub) \in \mathsf{L}$. A natural example of L is the language of all secret keys in some public-key encryption scheme, with pub being the public key and S being the corresponding private key. The goal of the prover is to convince the verifier that she knows S such that $\varphi(S, pub) = 1$ and that this S is stored on an individual machine. Moreover, this should be done in zero-knowledge, i.e., without revealing additional information to the verifier about S. We call a protocol that satisfies these requirements a *ZK-Proof of Individual Knowledge (zkPIK)* for relation φ.

Let us now outline how to transform a public-coin PIK protocol $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ into a zkPIK, where by "public-coin PIK" we mean protocols where the messages of the verifier are drawn at random, independently from S. It is easy to see that our $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ protocol (see Fig. 3.2) is public-coin since the only message that the verifier sends is the random challenge Z in Step 1. Our protocol works as follows. The prover and the verifier execute $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ with the following modification: the prover, instead of sending her messages in clear, commits to them using a cryptographic commitment scheme [26], and later proves in zero-knowledge [69] that she committed to messages that would make the verifier in protocol $\pi^{\rho,\sigma}_{\mathsf{PIK}}$ accept. Note that the zero-knowledge proof can be executed after the message exchange is finished; hence, the time needed to execute it does not influence the time bounds of the original PIK protocol. Clearly, this proof can also be executed in a non-interactive way [22], e.g., using one of the Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK) schemes (see, e.g., [21]). Note that this solution assumes a generic use of zero-knowledge protocols.

Reducing the number of candidate messages.

Recall that one of the parameters in PIK is the maximal size α_{O} of the set of "candidate secrets $\widehat{S}$", that besides the real value, S contains a lot of "false positives". Moreover,

α_{O} can be quite large in our construction. A simple technique for eliminating the false positives is to let the verifier publish a hash $h = H'(S)$ (where H' is some hash function different from the one used in constructing PIK). Once h is known, everyone (including the prover) can check for every candidate $\widehat{S}$ if $H'(\widehat{S}) = h$, and if it does not hold, eliminate this $\widehat{S}$ from the candidate set. It is easy to see that $\widehat{S} = S$ passes this test, while (assuming no collisions in H are found) all $\widehat{S}$'s such that $\widehat{S} \neq S$ get eliminated. This solution can be proven secure in the random oracle model as long as the entropy of S (from the point of view of an external adversary) is large enough to guarantee that the adversary cannot guess S. This is the case, e.g., if S is a uniform random key.

Let us also discuss how h can be published. One option is to let h be sent to the verifier by the prover during the execution of the protocol. In the case of the eaPIK protocol, h would be encrypted by key K. Note that we do not even need the assumption that S has high entropy since it remains hidden from the external attacker. In the case of zkPIK, h could just be a part of the public key.

3.5 Application: Voting on Blockchain

In this section, we *informally* describe how individual cryptography may be used to increase the security of voting systems on the blockchain. Voting on blockchains faces many problems [99] that diminish its applicability to large-scale elections, including issues with the privacy of a voter or coercion-freeness (i.e., the property that ensures that one is not able to "sell" a vote to any other party). However, in practice, some voting mechanisms are still implemented on blockchains, e.g., in Decentralized Autonomous Organisations (DAOs) [102]. DAOs are programs running on blockchains designed to manage organizations transparently and autonomously. Below, we define a model of a voting mechanism in DAOs – the Voting-DAO.

Let us first informally describe a cryptographic signature scheme $\Sigma = (\Sigma.\mathsf{gen}, \Sigma.\mathsf{sign}, \Sigma.\mathsf{verify})$. We assume that all algorithms of the signature system Σ are easy to compute in MPC. Given a security parameter κ, the algorithm $\Sigma.\mathsf{gen}$ produces a pair of random keys $(\mathsf{sk}, \mathsf{pk}) := \Sigma.\mathsf{gen}(\kappa)$ consisting of a private and a public key. The algorithm $\Sigma.\mathsf{sign}$ allows to create a signature $\sigma := \Sigma.\mathsf{sign}_{\mathsf{sk}}(m)$, given a secret key sk and a message m. Finally, the algorithm $\Sigma.\mathsf{verify}$ outputs a verification result $b := \Sigma.\mathsf{verify}_{\mathsf{pk}}(\sigma, m)$, given a signature σ, a public key pk and a message m. The correctness property of the system requires that given a correct pair of keys $(\mathsf{sk}, \mathsf{pk})$ and a message m, the algorithm produces a signature $\sigma := \Sigma.\mathsf{sign}_{\mathsf{sk}}(m)$ that is verified as valid, i.e. $\Sigma.\mathsf{verify}_{\mathsf{pk}}(\sigma, m)$ outputs 1. The unforgeability property of the signature requires that given a public key pk it is hard to forge a new signature σ' that verifies correctly on some message m' (i.e., it is hard to find a signature σ' and some message m', such that $\Sigma.\mathsf{verify}_{\mathsf{pk}}(\sigma', m')$ outputs 1) without access to the secret key sk that conforms to the given public key. A full definition of a signature scheme can be found in [100].

We now define the Voting-DAO – a procedure that allows periodic voting on a blockchain. We assume that the blockchain can preserve private memory[3]. Let us model Voting-DAO as a program on a private blockchain that can run procedures

[3]Implementing private memory on a public blockchain may sound tricky, but some implementations based on an assumption of a committee with honest majority were proposed e.g. in [16].

(Setup, VoteInit, VoteAdd, VoteCount).

- The procedure Setup can be run only once, during initialization of the Voting-DAO system, and allows to register a set of parties $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ by publishing their public keys $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ that conform to some secret keys $\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}$ generated by the algorithm $\Sigma.\mathsf{gen}$.

- The procedure VoteInit is used multiple times to initiate the voting process. It starts a clock that measures time until some deadline T. The clock can be deleted only when the deadline T passes (see the definition of the procedure VoteCount). The procedure VoteInit tosses and publicly announces a random nonce nonce. It can be started only when any previously started clock is deleted.

- The procedure VoteAdd allows participants identified with their secret keys sk_i to submit their vote for an option $v_i \in \{0, 1\}$, by submitting a *valid* signature $\Sigma.\mathsf{sign}_{\mathsf{sk}_i}(\mathsf{nonce}||v_i)$ conforming to a public key pk_i that was saved during initialization. The vote is saved in the private memory of the Voting-DAO, and it cannot be overwritten. The procedure can be run when some clock is still before its deadline.

- The procedure VoteCount is run when the deadline of a not-yet-deleted clock is over. It deletes the clock and publicly announces a voting result, which is a majority of the votes v_i published in the Voting-DAO. If none of the parties voted, it outputs a default value of 0.

Leveraging distributed cryptography to manipulate Voting-DAO. With distributed cryptography in hand, one can easily facilitate the process of manipulating the Voting-DAO. It is easy to see that, during a voting process modeled by the Voting-DAO, distributed cryptography may be used to allow a party $\mathcal{P}_j$ to vote on behalf of party $\mathcal{P}_i$ while still preserving the secrecy of its secret key sk_i by the party $\mathcal{P}_i$ and the secrecy of vote v_j by the party $\mathcal{P}_j$. The parties can engage in an MPC protocol that outputs $\sigma := \Sigma.\mathsf{sign}_{\mathsf{sk}_i}(\mathsf{nonce}||v_j)$ only to the holder of the vote v_j. The party $\mathcal{P}_j$ can then submit the valid vote σ to the Voting-DAO via a connection that preserves the anonymity of the transferred information.

What is more, distributed cryptography allows the creation of a correct signature $\Sigma.\mathsf{sign}_{\mathsf{sk}_i}(\mathsf{nonce}||v_i)$ without a guarantee that a single entity possesses secret information needed to compute this ballot. For example, the secret sk_i may be initially secret-shared by running the procedure $\Sigma.\mathsf{gen}$ in MPC or TEE by a set of parties. The MPC or TEE may be designed to allow only a single party to retrieve the generated secret. This way, one can organize an automated vote-selling machine that may control the voting process until its users collect back their secrets.[4]

Preventing the distributed cryptography attacks. The Voting-DAO may be enhanced using an extension similar to the zkPIK described in the previous section. The procedure VoteAdd may require that along with the voting signature $\Sigma.\mathsf{sign}_{\mathsf{sk}_i}(\mathsf{nonce}||v_i)$, each party should commit on time to $\pi = PIK(\mathsf{sk}_i||\mathsf{nonce}||v_i)$ and then later prove in ZK that given nonce, v_i and pk_i visible to the Voting-DAO, the committed π was computed

[4]This attack is similar to the automated DAO attack mentioned in [6].

using the sk_i that conforms to pk_i. Informally speaking, this procedure will ensure that a single party that computed the commitment to π had access to v_i, nonce, and sk_i needed to compute $\Sigma.\mathsf{sign}_{\mathsf{sk_i}}(\mathsf{nonce}||v_i)$. We postpone the formalization of the adjusted scheme for future investigation.

Chapter 4

Efficiently Testable Circuits without Conductivity

The final aspect of the security of systems built upon blockchains is the security of the devices used by blockchain users. The purpose of these devices is to *increase* the security of high-stake asset management. Various approaches are taken, which involve either leveraging trusted hardware units and devising new hardware configurations for managing blockchain assets [45, 71] or examining the security of the trusted hardware units themselves [76, 75, 53]. For concreteness, let us delve into the management of the secret information needed to produce signatures of transactions. Intuitively, a specialized device capable of computing signatures that contains a memory unit shielded from any external access [45] can improve the protection of secret information. However, ensuring the device generates signatures according to the intended specifications poses significant challenges. One approach is to rely on vendor-trusted modules like Trusted Platform Module [106] or Intel's Software Guard Extensions [116]. On the other hand, these technologies have been shown vulnerable multiple times [70, 110, 63]. We thus take a theoretical approach to this problem and try to understand what provable assurances can be given by the manufacturers of such cryptographic modules. Specifically, we provide a generic construction of a compiler that protects implementations of devices modelled as Boolean circuits against so-called wire-tampering attacks.

Together with Baig et al. [8], we introduced the notion of "Efficiently Testable Circuits" (ETC). It assures that any Boolean circuit C can be compiled into a pair consisting of a circuit C' and a small test set $\mathbb{T}$, such that the circuit C' is functionally equivalent to C (i.e. the input/output behavior of a subset of input/output wires of the compiled circuit C' is the same as the input/output behavior of the original circuit C). Moreover, the notion of ETCs assures that testing the input/output behaviour of the tampered circuit C' on the small set of inputs $\mathbb{T}$ allows to check whether C' preserves correct input/output behaviour on all inputs.

Although the ETC compiler above is secure against tamperings that modify even all wires of the compiled circuits, it was built upon a strong *conductivity* assumption. Conductivity assures that any duplicated wires (used to forward some value to more than one gate) can be tampered with only a *single tampering*. In another work [9], we constructed a compiler for ETCs that did not leverage the conductivity assumption. This construction is randomized and contains a layered *compression gadget*. We show that

a single test in this construction, which contains a *compression gadget* with λ layers, propagates an error with probability $1/s^{2\lambda}$. A natural question is whether it is possible to boost the probability of error detection in the compression gadget of this compiler, and provide a compiler that would result in more cost-effective constructions w.r.t. the usage of randomness.

Our contribution is a construction of a compiler, built upon the construction presented in [9], with a modified layered compression gadget. The new construction with λ layers, boosts the probability of error detection to $\left(1-(3/4)^b\right)^{\lambda-1}/2^{\lambda+1}$, assuming some integer parameter $b>1$. For sufficiently large b, the probability of error detection of our construction is approximately $2^{\lambda-1}$ larger than the probability of the error detection of the construction presented in [9].

4.1 Testability of Circuits

We start by presenting a broader context of the issue of testability of circuits.

Circuit testing. Detecting errors in circuits is of interest in various engineering and computer science areas. In circuit manufacturing, the focus is on efficiently detecting errors that randomly occur during production [27]. Querying circuits on a few carefully chosen inputs and checking the output for correctness will typically detect a large fraction of the faulty ones.

Private Circuits (PC). The cryptographic community has long focused on errors that are intentionally introduced by an adversary, as such "tampering" or "fault attacks" can be used to extract cryptographic secrets [20, 24]. Compared to testing in manufacturing, protecting circuits against fault attacks is more difficult for at least two reasons: (1) the errors are not just random but can be targeted on specific wires or gates in the circuit (2) the errors introduced by tampering must not just be detected, but the circuit must be prevented to leak any information.

For this challenging setting of *private circuits* (PC), Ishai, Prabhakaran, Sahai, and Wagner [75] construct a *circuit compiler* that given (the description of) any circuit C and some parameter k outputs (the description of) a functionally equivalent circuit C_k (i.e., $C(X)=C_k(X)$ for all X) which is secure against fault attacks that can tamper with up to k wires with each query (the faults can be persistent, so ultimately the entire circuit can be tampered with), while blowing up the circuit size by a factor of k^2. The efficacy of the compiler can be somewhat improved by allowing some small information leakage [62].

Efficiently Testable Circuits (ETC). Efficiently testable circuits (ETC), recently introduced in [8], consider a setting that "lies in between" testing for benign errors and private circuits. An ETC compiler takes any Boolean circuit $C:\mathbb{Z}_2^s\to\mathbb{Z}_2^t$ and maps it to a tuple $(C_{\mathsf{test}}:\mathbb{Z}_2^{s+s'}\to\mathbb{Z}_2^{t+t'},\mathbb{T}_{\mathsf{test}}\subset\mathbb{Z}_2^{s+s'})$ where C_{test} is functionally equivalent to C and $\mathbb{T}_{\mathsf{test}}$ is a test set that will catch any (non-trivial) tampering on C_{test}. A bit more formally, by saying C_{test} is functionally equivalent to C we mean $\forall X\in\mathbb{Z}_2^s : C_{\mathsf{test}}(X\|0^{s'})_{|t}=C(X)$ ($S_{|t}$ denotes the t bit prefix of S, $\|$ is concatenation and 0^s is the string of s zeros).

The security property states that if for a wire tampering τ on C_{test} the tampered circuit C^{τ}_{test} output errors on at least one of the (exponentially many) inputs $X\|0^{s'}$ (i.e., the t bit prefix of the output is not $C(X)$), then C^{τ}_{test} will err on at least one input in the (small) test set:

$$\forall \tau : \exists X \in \mathbb{Z}_2^s \; s.t. \; C^{\tau}_{\mathsf{test}}(X\|0^s)_{|t} \neq \overbrace{C_{\mathsf{test}}(X\|0^{s'})_{|t}}^{=C(X)} \quad \Rightarrow$$
$$\exists T \in \mathbb{T}_{\mathsf{test}} \; s.t. \; C^{\tau}_{\mathsf{test}}(T) \neq C_{\mathsf{test}}(T) \tag{4.1}$$

ETC aims at detecting *adversarial* errors like PC, but unlike PC, this detection only happens during a dedicated testing phase, not implicitly with every query. Thus ETC cannot be used to replace PCs which aim to protect secrets on a device that is under adversarial control and can be tampered with. Instead, they ensure that a circuit correctly evaluates on all inputs, even if it was under adversarial control in the past.

Using ETC can also be useful to detect benign errors, particularly in settings where one does not want to accept a non-trivial probability of missing a fault, which is the case for the heuristic techniques currently deployed in circuit manufacturing. One such setting is in space exploration where faults can be catastrophic, and to make matters worse, the high radiation in outer space is likely to cause additional faults. Here, the ability to run a cheap test repeatedly in a black-box way is useful.

While ETCs provide a weaker security guarantee than PCs in terms of how tampering is detected, the construction of the ETC from [8] achieves security under a much stronger tampering model than what is known for PCs. Furthermore, ETCs are much more efficient and rely on weaker assumptions: the ETC compiler from [8] blows the circuit up by a small constant factor while allowing for tampering *with all wires*. On the other hand, in *Private Circuits II* [75], to detect tampering with k wires already requires a blow up of k^2.

Conductivity. A major restriction of both, the PC compiler [75] and the ETC compiler from [8] is the fact that wire tamperings are assumed to be *conductive*: while a wire can be tampered (set to constant 0 or 1, or toggling) arbitrarily, if this wire has fan-out greater than 1, i.e., leads to more than one destination which can be an input to another gate or an output wire, all must carry the same value and cannot be tampered individually.[1] This is an arguably unrealistic assumption and not does not capture actual tampering attacks: Why should, say, cutting the wire at the input of one gate affect the value at another gate to which this wire is connected? While any circuit can easily be turned into a functionally equivalent one where all wires have fan-out 1 by using copy gates $\mathsf{COPY}(b) = (b, b)$, applying this to the circuit produced by the compiler from [8] will completely break its security as shown in [9]. To this end, in [9], we constructed a compiler for ETCs that gives probabilistic guarantees for the testability of the compiled circuit. Moreover, their compiler *does not* leverage the conductivity assumption.

[1]The conductivity assumption for the PC compiler from [75] is slightly stronger than ours, as they additionally assume that "faults on the output side of a NOT gate propagate to the input side".

4.2 Our Contribution

In this chapter, we show a new construction of a compiler for ETCs that does not leverage the conductivity assumption. Our construction builds upon the construction presented in [9] and similarly contains a layered compression gadget. However, for λ layers of compression and a parameter b, our compiler propagates any detected error with probability $\frac{1}{2}\left(1-(3/4)^b\right)^{\lambda-1}/2^{\lambda}$ in comparison to $1/2^{2\lambda}$ achieved by the old construction. In other words, for any circuit C compiled into a pair $(C_{\mathsf{test}}, \mathbb{T}_{\mathsf{test}})$, we get a probabilistic guarantee that:

$$\exists T \in \mathbb{T}_{\mathsf{test}} \; s.t. \; \Pr_{R}[C^{\tau}_{\mathsf{test}}(T\|R) \neq C_{\mathsf{test}}(T\|R)] \geqslant \left(1-(3/4)^b\right)^{\lambda-1}/2^{\lambda+1} \tag{4.2}$$

where $\lambda \in \mathbb{N}$ is a parameter specifying the number of layers in the compression sub-gadget, and b is a boosting parameter of the testing sub-gadget. A larger λ will decrease the extra input/output wires but will increase the required number of test queries, a larger b will increase the number of extra input/output wires but will decrease the required number of test queries. To achieve the above guarantee of boosted probability, we formally study how the error is propagated through each layer of the compression gadget depending on a *single* bit of randomness crossed with the wire containing the error[2].

Implications - size, and randomness efficiency. Below, we make a discussion of size and randomness efficiency for a circuit of size n, λ layers of compression, and two parameters $d, b \in \mathbb{N}, b < d$ (where d is the compression parameter, and b is the boosting parameter). Our compiler propagates the error with probability $\left(1-(3/4)^b\right)^{\lambda-1}/2^{\lambda+1}$ after a single query, it grows the circuit to the size of roughly $15n + (12n + b \cdot n) \cdot \left[1-(b/d)^{\lambda}\right]/(1-b/d)$, it uses $\lambda \cdot d$ bits of randomness for a single test, and adds approximately $4n \cdot b^{\lambda-1}/d^{\lambda} + d \cdot b/(d-b)$ output wires to the circuit. In comparison, the construction from [9] propagates the error with probability $1/2^{2\lambda}$ after a single query, it grows the circuit to the size of roughly $27n$, it uses $\lambda \cdot d$ bits of randomness for a single test, and adds approximately $4n/d^{\lambda}$ output wires to the circuit. For concreteness, for a circuit with approximately $15 \cdot 10^9$ gates, $\lambda = 6$ layers of compression, the compression parameter $d = 60$, the boosting parameter $b = 5$, we can estimate, that our compiler will propagate the error with probability approximately 0.002 after a single test, it will grow the circuit approximately 35 times, it will use 360 bits of randomness, but it will add around 4000 new output bits to the construction. On the other hand, the construction from [9] will use the same amount of 360 bits of randomness; it will propagate the error with probability approximately 0.00025 (around 8 times lower than our construction), but it will grow the circuit 20 times and it will add only 2 output wires to the construction. We believe that the optimal parameters of a practical implementation of an ETC compiler should be obtained through the engineering process combining the layers of our construction and the construction from [9].

[2]In contrast, the construction from [9] studies how the error is propagated through each layer of their construction depending on the whole vector of randomness attached to the layer. Their guarantee is insufficient to use a single randomness bit several times to boost the probability of catching the error on a single wire.

Our solution in a nutshell. To obtain our construction, similarly to [9], we start with precompiling a circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ to a pair $(C_{\mathsf{gate}}, \mathbb{T}_{\mathsf{gate}})$, such that C_{gate} is functionally equivalent to C. What is more, after the precompilation, only a few input wires and a *large* number of output wires (a few times larger than the number of gates of the original circuit C) are added to the precompiled C_{gate}. The precompiled circuit C_{gate} has a property that it is testable[3] on the test set $\mathbb{T}_{\mathsf{gate}}$ (cf. Equation 4.1). As in [9], we apply a layered construction to lower the number of output wires of the precompiled construction. We fix a compression parameter d and a boosting parameter b. To construct the layered construction of our gadget we do the following.

- In the first layer, we group the output wires of the precompiled circuit to groups of d wires. Then (given a randomness vector $R_1 = (r_{1,1}, \ldots, r_{1,d})$ attached to the first layer) we compute product $\sum_{j=1,\ldots,d} z_j r_j$ for each group of wires $(z_1, \ldots, z_d)$. The output of the computation of each group must be then copied b times, using the COPY gates.

- In a layer $i \in \{2, \ldots, \lambda - 1\}$, we *reuse* each randomness bit of a vector $R_i = (r_{i,1}, \ldots, r_{i,d})$ attached to the layer. We do this by rewiring the output from a single group in the previous layer which was already copied b times in the previous layer. In other words, we rewire a single output from the previous layer to b *distinct* randomness bits from $R_i = (r_{i,1}, \ldots, r_{i,d})$ in the layer i. This allows us to compute the product several times on wires carrying the same information (cf. Figure 4.4 and Section 4.7). Again, the output of the computation of each product in the layer i must be copied b times, using the COPY gates.

- The last layer of our construction is constructed similarly to the previous layers, but the output of the product computations in the last layer is carried only by a single wire and it is not copied b times.

To achieve the boosted security guarantee, in the Theorem 22, we study how the error is propagated with respect to a single randomness bit $r_{i,j}$ in a layer i, not the whole randomness vector R_i as in [9]. As we show in the final Theorem 23, our construction $(C_{\mathsf{test}}, \mathbb{T}_{\mathsf{test}})$ assures that (given a fixed number of additional input wires s_λ):

- **Either the output is wrong:**

$$\exists X \in \mathbb{T}_{\mathsf{test}} : C^{\tau}_{\mathsf{test},\lambda}(X||0^{s_\lambda}) \neq C_{\mathsf{test},\lambda}(X||0^{s_\lambda}),$$

- **or the testing gadget detects an inconsistency:**

$$\exists X \in \mathbb{T}_{\mathsf{test}} : \Pr_{R \leftarrow \mathbb{Z}_2^{s_\lambda}} \left[C^{\tau}_{\mathsf{test},\lambda}(X||R) \neq C_{\mathsf{test},\lambda}(X||R) \right] \geqslant \left(1 - (3/4)^b\right)^{\lambda - 1} / 2^{\lambda+1}.$$

[3]The actual security guarantee provided by the precompiled circuit is slightly stronger than the "testability" (cf. Equation 4.1), as we achieve a guarantee of observability of so-called "information loss", cf. Section 4.5.

4.2.1 More Related Work

Testing circuits is a significant topic in hardware manufacturing, the books [19, 27] discuss heuristics for testing and practical issues of the problem. Circuit-compilers (as used in this chapter) which harden a circuit against some "physical attacks" in a provably secure way were first introduced for leakage attacks (concretely, leaking values of a small number of wires) by Ishai et al. in [77]. Based on this compiler they later also gave a compiler to protect against tampering [75]. This line of research was continued in a sequence of papers on tampering wires [41, 42, 65, 61] or gates [78, 57, 83]. As discussed in the introduction, these compilers aim at protecting secrets in the circuit, while efficiently testable circuits [8] only aim at detecting tampering in a special test phase.

Apart from compilers, a line of research was pioneered by Micali and Reyzin in [95] on reductions or composition of cryptographic building blocks to prevent "general" leakage. The first cryptographic primitive achieving security against a general notion of leakage (bounded leakage) from standard cryptographic building blocks is the leakage-resilient cipher from [55], by now we have leakage-resilient variants of most basic cryptographic primitives including signatures [60, 25] or MACS [18], an excellent overview on the area is [79]. Unfortunately for tampering no construction secure against general tampering – or even a notion of what "general tampering" means – exists, where notions like the non-malleable codes [56] are applicable. The non-malleable codes can protect data in memory (rather than during computation) from very general classes of tampering attacks [88, 31, 2, 10, 43].

Trojans is the most powerful physical attack model, where an attacker can not just tamper the circuit but completely replace it. Some limited provable-security results against this class of attacks are [53, 29]. There are few attempts to use general verifiable computation to certify the output of circuits [5, 111].

4.3 Preliminaries

We borrow the notations and the tampering model from [9].

4.3.1 Notation for Circuits

A circuit is modeled as a Directed Acyclic Graph $C_\gamma = (V, E)$ and a labeling function $\gamma : V \to \mathfrak{G}$. The vertices in V refer to gates and the directed edges in E refer to wires. The labeling function γ assigns specific gates to the vertices, where $\mathfrak{G}$ is the set of gates allowed[4]. Each wire carries a bit from $\mathbb{Z}_2$, and each gate is taken from the set of allowed gates $\mathfrak{G}$ (including $\{\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}, \mathsf{COPY}, \mathsf{NOT}\}$ and two special $\{\mathbf{in}, \mathbf{out}\}$ gates).

For $v \in V$, let $E^-(v) = \{(u, v) \in E\}$ and $E^+(v) = \{(v, u) \in E\}$ be the sets of v's incoming and outgoing edges, respectively. For $e = (u, v) \in E$ we define $V^-(e) = u$ and $V^+(e) = v$. We split the vertices into three sets $V = \mathcal{I} \cup \mathcal{G} \cup \mathcal{O}$, where $\mathcal{I} = \{I_1, I_2, ..., I_s\}$ are vertices which are assigned to **in**, and $\mathcal{O} = \{O_1, O_2, ..., O_t\}$ are these assigned to **out**.

[4]The parameter γ will be often defined only implicitly in the text, as it will be obvious from the context.

Given $C_\gamma = (V, E)$ and an input $X = (x_1, \ldots, x_s) \in \mathbb{Z}_2^s$, we define a valuation function

$$\mathrm{val}_{C_\gamma, X} : V \cup E \to \mathbb{Z}_2 \tag{4.3}$$

which assigns each gate the value it outputs and each wire the value it holds when the circuit is evaluated on X. More formally the valuation function for vertices $v \in V$ and edges $e \in E$ is defined as

$$\begin{aligned} \mathrm{val}_{C_\gamma, X=(x_1,x_2,\ldots,x_s)}(v) &= \begin{cases} x_i, & \text{if } v = I_i. \\ \gamma(v)(\mathrm{val}_{C_\gamma,X}(E^-(v))), & \text{otherwise.} \end{cases} \\ \mathrm{val}_{C_\gamma, X}(e) &= \mathrm{val}_{C_\gamma,X}(V^-(e)). \end{aligned}$$

We will sometimes just write val_X if the circuit considered is clear from the context. The behaviour of the circuit C can be associated with the function that it evaluates, i.e. $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$. We can define this function as follows: $C(X) = (\mathrm{val}_{C,X}(O_1), \mathrm{val}_{C,X}(O_2), \ldots, \mathrm{val}_{C,X}(O_t))$.

Additionally, we add the notation of the sequences of the input values to gates. By $(\mathrm{val}_{C,X}(e))_{e \in E^-(v)}$ we understand the sequence of the values given to v, when the circuit is evaluated on the input X. E.g. for a gate v in a circuit C that is evaluated on 0 and 1 given input X to C, we write $(\mathrm{val}_{C,X}(e))_{e \in E^-(v)} = 01$.

4.3.2 Tampering Model

In our work, and [9], we consider an adversary who can arbitrarily tamper with every wire of the circuit, i.e., flip its value, set it to 0, set it to 1, or leave it untampered. Unlike [8] or [75], in [9] and this chapter, we *do not* take advantage of the *conductivity* assumption. This means, we operate on circuits with conductivity 1, where all nodes $n \in V$ of a circuit C have fan-in and fan-out equal to an inherent fan in, fan-out of $\gamma(n)$, and all output wires of all nodes can be tampered independently. We assume that the input circuit is not conductive. In [8], we assumed k-conductivity, i.e., a value of some wire in the circuit could be copied to at most k distinct destinations with a restriction that all of them must be tampered equally. Without loss of generality, every k-conductive circuit can be transformed into a 1-conductive circuit, as by using COPY gates, every k-conductive circuit can be turned into a non-conductive one while at most doubling the circuit size and increasing the depth by a factor $\lceil \log(k) \rceil$.

The tampering of a wire is described by a function $\mathbb{Z}_2 \to \mathbb{Z}_2$ from the set of possible bit tamper functions $\mathcal{T} = \{\mathrm{id}, \mathrm{neg}, \mathrm{one}, \mathrm{zero}\}$. The tampering of an entire circuit $C = (V, E)$ is defined by a function $\tau : E \to \mathcal{T}$ mapping each wire to a tampering function. We sometimes write τ_e to denote $\tau(e)$ for convenience. The valuation function can now also take the tampering τ into account:

$$\mathrm{val}_X^\tau : V \cup E \to \mathbb{Z}_2.$$

The only difference to the (non-tampered) valuation function from Eq(4.3) is that we

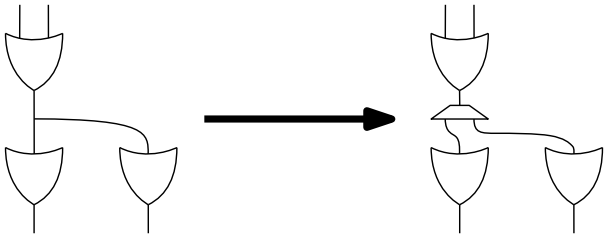


Figure 4.1: Examples of $2-$*conductive* (left circuit) and $1-$*conductive* (right circuit) circuits. A single wire on the left circuit is copied to two destinations. The adversary can apply only a single tampering to this wire. On the right circuit, this wire is divided into three parts with a COPY gate. The adversary can apply separate tampering to each of these parts.

apply the tampering to each value of an edge after it is being computed, formally:

$$\begin{aligned} \mathrm{val}^{\tau}_{C_\gamma, X=(x_{1,2},\ldots,x_s)}(v) &= \begin{cases} x_i, & \text{if } v = I_i. \\ \gamma(v)(\mathrm{val}^{\tau}_{C_\gamma,X}(E^-(v))), & \text{otherwise.} \end{cases} \\ \mathrm{val}^{\tau}_{C_\gamma,X}(e) &= \tau_e(\mathrm{val}^{\tau}_{C_\gamma,X}(V^-(e))). \end{aligned}$$

By C^τ we can again understand a function that describes the input-output behavior of the *tampered* circuit: $C^\tau(X) = (\mathrm{val}^{\tau}_{C,X}(O_1), \mathrm{val}^{\tau}_{C,X}(O_2), ..., \mathrm{val}^{\tau}_{C,X}(O_t))$.

4.4 Gate Covering Sets

In previous works [8, 9], we developed a notion of a *gate covering set* $\mathbb{T}_{\mathsf{gate}}$, that extended the notion of a *wire covering set* $\mathbb{T}_{\mathsf{wire}}$ presented in [8]. A wire covering set $\mathbb{T}_{\mathsf{wire}}$ is a set of inputs to a specific circuit, such that every wire is evaluated to both 0 and 1, given some inputs from the test set $\mathbb{T}_{\mathsf{wire}}$. Moreover, [8] provides a procedure that compiles every circuit into its wire-covered version.

Definition 4.4.1 (Definition 4 from [8])**.** *The set $\mathbb{T}_{\mathsf{wire}}$ is a wire covering set for a circuit C if $\forall\ e \in E(C),\ b \in \{0,1\}\ \exists\ X \in \mathbb{T}_{\mathsf{wire}}\ :\ \mathrm{val}_{C,X}(e) = b$.*

In our construction, we take advantage of the stronger notion of the gate covering set $\mathbb{T}_{\mathsf{gate}}$ introduced in [9]. It ensures the covering of all wires of the circuit and additionally the covering of the pairs of wires, which are input wires to multi-input gates (in our model, the AND, OR, XOR gates).

Definition 4.4.2 (Definition 2 from [9])**.** *The set $\mathbb{T}_{\mathsf{gate}}$ is a gate covering set for a circuit C with gate assignment γ if it satisfies:*

- $\forall_{v \in V(C)\ :\ \gamma(v) \in \{\mathsf{COPY},\mathsf{NOT},\mathsf{OUTPUT}\}} : |\{(\mathrm{val}_{C,X}(e))_{e \in E^-(v)} : X \in \mathbb{T}_{\mathsf{gate}}\}| \geqslant 2$,
- $\forall_{v \in V(C)\ :\ \gamma(v) \in \{\mathsf{AND},\mathsf{OR}\}} : |\{(\mathrm{val}_{C,X}(e))_{e \in E^-(v)} : X \in \mathbb{T}_{\mathsf{gate}}\}| = 4$,
- $\forall_{v \in V(C)\ :\ \gamma(v) \in \{\mathsf{XOR}\}} : |\{(\mathrm{val}_{C,X}(e))_{e \in E^-(v)} : X \in \mathbb{T}_{\mathsf{gate}}\}| \geqslant 3$.

We call a circuit with a gate-covering set a *gate-covered* circuit. Any node in a circuit that has enough evaluation sequences as in the Definition 4.4.2, given any test set, we call gate-covered.

The previous work [9] shows how to transform any circuit C into its functionally equivalent gate-covered circuit C_{gate} using their Algorithm 1 (we will refer to it as Algorithm A in this text). For any circuit C with max fan-in 2 and n gates, the Algorithm A adds only 5 input wires, creates a circuit of size $6n$ and a gate covering set $\mathbb{T}_{\mathsf{gate}}$ of size 6.

4.5 Information Loss in Gate-Covered Circuits

Information loss. In the following, we shall discuss the propagation of the *information loss*. We say that we have the information loss on a wire (or node) w in a circuit C, if one has access to two distinct inputs X_0, X_1 on which the wire (node) in the circuit should be evaluated to different bits, but is evaluated to the same bit because of the tampering τ applied to the circuit. In other words, for some X_0, X_1, we have:

$$\left(\mathrm{val}_{X_0,C}(w) = 0 \wedge \mathrm{val}_{X_1,C}(w) = 1\right) \wedge \left(\mathrm{val}^{\tau}_{X_0,C}(w) = \mathrm{val}^{\tau}_{X_1,C}(w)\right).$$

Information loss is a slightly more robust notion than the evaluation error, as the tamperings in a circuit may be adversarially chosen in a way that the evaluation error vanishes during the evaluation of the circuit. E.g., imagine a wire in the tampered circuit that is *almost* always evaluated to 0 in an untampered evaluation, and the adversarial tampering sets its value to *constant* 0. In general, it is easy for an adversary to manipulate the (almost) always correct or (almost) always incorrect evaluations on internal wires of the circuit. We will thus make use of the information loss - a pair of evaluations on a single wire that ensures that this wire evaluates to both 0 and 1, and an error occurs on one of these evaluations. We will be able to propagate the information loss through layers of the circuit without the risk of vanishing error.

How information loss propagates in gate-covered circuits. We now show that information loss is easily trackable in any gate-covered circuit C_{gate}.

For any such circuit with a gate-covering set $\mathbb{T}_{\mathsf{gate}}$, we show the following property: for any tampering applied to the wires of the C_{gate}, either we observe an information loss on one of the output wires of the multi-input gates $\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}$ (given only the inputs from the gate-covering set $\mathbb{T}_{\mathsf{gate}}$), or the output wires of the circuit are always set to a constant value or always toggled or always correctly evaluated.

Theorem 15. *For any circuit $C_{\mathsf{gate}} : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ with gate-covering set $\mathbb{T}_{\mathsf{gate}}$, for any tampering function τ applied to the circuit then at least one of the following holds:*

- ***Information loss on multi-input gates***

$$\exists_{X_0,X_1\in\mathbb{T}_{\mathsf{gate}},\ n\in V(C_{\mathsf{gate}})\ :\ \gamma(n)\in\{\mathsf{AND},\mathsf{OR},\mathsf{XOR}\}}\ :$$

$$\Big(\mathrm{val}_{X_0,C_{\mathsf{gate}}}(n)=0\wedge\mathrm{val}_{X_1,C_{\mathsf{gate}}}(n)=1\Big)\wedge\Big(\mathrm{val}^{\tau}_{X_0,C_{\mathsf{gate}}}(n)=\mathrm{val}^{\tau}_{X_1,C_{\mathsf{gate}}}(n)\Big)$$

- ***Constant output***

$$\exists_{i\in[t],c\in\{0,1\}}\forall X\in\mathbb{Z}_2^s\ :\ C^{\tau}_{\mathsf{gate}}(X)[i]=c$$

- ***At most toggled output***

$$\exists_{T\in\{0,1\}^t}\forall X\in\mathbb{Z}_2^s:\ C^{\tau}_{\mathsf{gate}}(X)=C_{\mathsf{gate}}(X)+T$$

Proof. The proof follows a modular argument. For this, we need a definition of Topological Layers of Computation on any circuit C_γ. In the definition below, we say that a wire e is connected to a gate g in a circuit C_γ described with a DAG (denoted by predicate $connected_{C_\gamma}(g,e)$ holds) if and only if there exists a direct connection or connection going through a path of COPY or NOT gates between g and the predecessor of e in the circuit.

Definition 4.5.1 (Topological Layers of Computation)**.** *For any circuit C, we recursively define its Topological Layers of Computation:*

- 0^{th}*-layer of Computation* $\mathcal{L}_0=\mathcal{I}(C)$
- i^{th}*-layer of Computation* $\mathcal{L}_i=\{g\in V(C_\gamma):\forall_{e\in E^-(g)}: connected_{C_\gamma}(g',e)$ *for some* $g'\in\mathcal{L}_0\cup\ldots\cup\mathcal{L}_{i-1}$ *and* $\gamma(g)\in\{\mathsf{XOR},\mathsf{AND},\mathsf{OR}\}\}$.

By $G_i(C)$, we denote a subgraph induced by the layers $\mathcal{L}_0,\ldots,\mathcal{L}_i$ of the circuit. Below we consider $C=C_{\mathsf{gate}}$. We run an experiment that evaluates layer by layer the tampered C (assuming C has $L+1$ layers). In the i'th layer either there is an information loss and we stop the experiment or the layer's output is at most toggled [see event $\mathcal{E}_2$ below] and the experiment proceeds to the next layer. We define the following predicates for a gate g in layer i:

- $\mathcal{E}_1(g,i)$ holds if $g\in\mathcal{L}_i\ \wedge\ \exists_{X_0,X_1\in\mathbb{T}'}\mathrm{val}_{X_0,C}(g)=0\ \wedge\ \mathrm{val}_{X_1,C}(g)=1\ \wedge\ \mathrm{val}^{\tau}_{X_0,C}(g)=\mathrm{val}^{\tau}_{X_1,C}(g)$,
- $\mathcal{E}_2(g,i)$ holds if $g\in\mathcal{L}_i\ \wedge\ \forall_{X\in\mathbb{Z}_2^s}:\mathrm{val}^{\tau}_{X,C}(g)=\mathrm{val}_{X,C}(g)+f\Big[\tau(e):e\in E(G_i(C))\Big]$.

In the 0^{th}-layer of the circuit, by definition of the tampering function, for any node $g\in\mathcal{L}_0(C)$: $X\in\mathbb{Z}_2^s:\mathrm{val}^{\tau}_{X,C}(g)=\mathrm{val}_{X,C}(g)$. This implies event $\mathcal{E}_2(g,0)$ on any gate from this layer. We prove the following for the tampered circuit C:

$$\forall_{\tau(C),i\in\{1,\ldots,L\}}:\forall_{j\in\{0,\ldots,i-1\},g'\in\mathcal{L}_j}:\mathcal{E}_2(g',j)\implies\forall_{g\in\mathcal{L}_i(C)}\ :\ \mathcal{E}_1(g,i)\vee\mathcal{E}_2(g,i)$$

We first study the gates of the first layer:

- The AND gate in the 1^{st}-layer is connected to the input gates only via a sequence of COPY and NOT gates. The computation on this gate can be described as $P_g(a,b) = a \cdot b$. The tampered output of the gate is $\tilde{P}_g(a,b) = \tilde{a} \cdot \tilde{b}$, where $\tilde{a} \in \{a, a+1, 0, 1\}$, $\tilde{b} \in \{b, b+1, 0, 1\}$. The tampering of a wire a is set to 1 or 0 whenever there is constant tampering on its path from the 0^{th}layer, $a+1$ or $b+1$ whenever on the path there is an odd number of toggle tamperings, and a or b whenever there is an even number of toggle tamperings on the path. Whenever $\tilde{a} = 0 \vee \tilde{b} = 0$, then $\tilde{P}_g(a,1) = 0$ and $P(a,1) = a$, we get an information loss. Now, since by the construction of the Algorithm 4, the wire P is connected via a COPY to the output, the event $\mathcal{E}_1(g,1)$ occurs.

  In other cases:

  - if $\tilde{a} = 1$ (or $\tilde{b} = 1$), $P(a,1) = a$ and $\tilde{P}(a,1) = const.$ (resp. $P(b,1) = b$ and $\tilde{P}(b,1) = const.$) [$\mathcal{E}_1(g,1)$ occurs],
  - when $\tilde{a} = a+1$ (or $\tilde{b} = b+1$), then $P(1,b) = b$ and $\tilde{P}(1,b) = 0$ (resp. $P(1,a) = 1$ and $\tilde{P}(a,1) = 0$) [$\mathcal{E}_1(g,1)$ occurs],
  - otherwise $\tilde{P}(a,b) = ab$ [$\mathcal{E}_2(g,1)$ occurs].

- A detailed analysis similar to the above applies for the OR gate. Whenever $\tilde{a} = 1 \vee \tilde{b} = 1$, or both of the input wires are tampered to a constant, then the output of the gate is constant and we get the $\mathcal{E}_1(g,1)$ event. Whenever at least one of the input wires (say the wire a) is tampered to constant 0, then (for a fixed value of the other wire) the output of the gate should switch, but does not switch. One again achieves the $\mathcal{E}_1(g,1)$ event. Finally, whenever at least one of the input wires is toggled (say the wire a), on the input $a = 0$, and different values of the input wire b, the output should switch, but does not.

- The input wires of the XOR gate are also connected only via a sequence of COPY and NOT gates to the input. We observe that $P_g(a,b) = a + b$, and the tampered output $\tilde{P}_g(a,b) = \tilde{a} + \tilde{b}$, where $\tilde{a} \in \{a, a+1, 0, 1\}$, $\tilde{b} \in \{b, b+1, 0, 1\}$.

  - if $\tilde{a} = const.$ (or $\tilde{b} = const.$), $P(a,0) = a$ and $\tilde{P}(a,0) = const.$ (resp. $P(0,b) = b$ and $\tilde{P}(0,b) = const.$) [$\mathcal{E}_1(g,1)$ occurs],
  - when $\tilde{a} = a + c_a, \tilde{b} = b + c_b$, then $P(a,b) = a + b$, $\tilde{P}(a,b) = a + b + c_a + c_b$ [$\mathcal{E}_2(g,1)$ occurs].

In the i'th layer, the inputs to all of the gates are, again, connected to the gates of the previous layers only via a sequence of COPY, NOT gates. Now, once the induction assumption holds in the layers $\{1, \ldots, i-1\}$, the event $\mathcal{E}_2$ on all gates assures that the case analysis from the first layer may be repeated, but the tampered wires $\tilde{a}, \tilde{b}$ will now get a constant tampering 0 or 1, or a toggle bit depending on the tamperings chosen on the edges of the graph induced by layers from the set $\{0, \ldots, i\}$.

This implies that on multi-input gates of the circuit, we either get event $\mathcal{E}_1$ or $\mathcal{E}_2$. Whenever the event $\mathcal{E}_1$ occurs, the information loss on one of the multi-input gates of the circuit occurs. Otherwise only the event $\mathcal{E}_2$ on these gates may occur. The OUTPUT gates of the circuit are connected via a sequence of COPY and NOT gates to the gates of

the topological layers of computation of the circuit. If on their paths one finds a constant tampering, then some output bit is set constant; if only toggles are found there, the output bits are at most toggled, which concludes the last case of the Theorem. □

4.5.1 Routing the Information Loss in Gate-Covered Circuits

In this section, we show that any gate-covered circuit can be converted to another gate-covered circuit for which any information loss that appears on its multi-input gates is routed to the output of the circuit. We present Algorithm 4 that adds a COPY gate to the output wires of the multi-input gates in the gate-covered circuit. The added COPY gates forward one copy of the original wires to their previous destinations and another copy directly to the output (Fig 4.2).

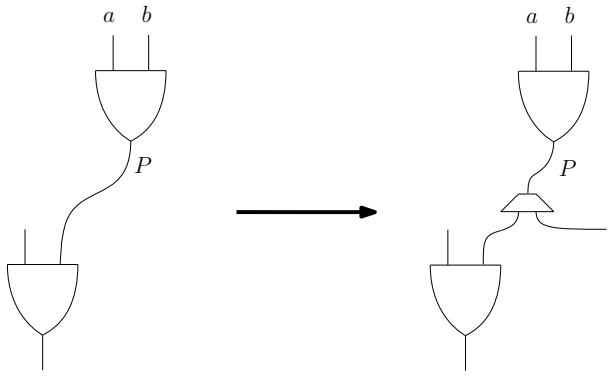


Figure 4.2: Adding a COPY gate to the wire P in the Algorithm 4. This creates two wires; the left one is connected to the previous successor of the wire P, and the right one is sent to the output of the circuit. Algorithm 4 takes into account only the wires P which originate at $\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}$ gates in the original circuit.

Algorithm 4: Routing the Information Loss in a Gate-Covered C

Input: $C_{\mathsf{gate}} : \mathbb{Z}_2^s \to \mathbb{Z}_2^t, \mathbb{T}_{\mathsf{gate}}$
Output: $C_{\mathsf{test},0}, \mathbb{T}_0$
Initialize $C_{\mathsf{test},0} = C_{\mathsf{gate}}$, $\mathbb{T}_0 = \mathbb{T}_{\mathsf{gate}}$
for $g \in V(C_{\mathsf{gate}})$ **do**
  if $g \in \{\mathsf{AND}, \mathsf{OR}, \mathsf{XOR}\} \wedge E^+(g)$ *is not an output wire of* $C_{\mathsf{test},0}$ **then**
    Insert to $C_{\mathsf{test},0}$ a COPY gate between g and $V^+(w)$.
    One of the output wires of the new gate should go to $V^+(w)$, the other one should be left as an additional output wire of the modified circuit.
return $C_{\mathsf{test},0}, \mathbb{T}_0$

Proposition 16. *The Algorithm 4 transforms a gate-covered circuit* $C_{\mathsf{gate}} : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ *with gate-covering set* $\mathbb{T}_{\mathsf{gate}}$ *into another gate-covered circuit* $C_{\mathsf{test},0} : \mathbb{Z}_2^s \to \mathbb{Z}_2^{t+t_0}$ *with additional output bits and the same gate-covering set,* $\mathbb{T}_0 = \mathbb{T}_{\mathsf{gate}}$*, where one observes for any tampering* τ *of the circuit* $C_{\mathsf{test},0}$ *at least one of the following holds:*

- ***Information loss on output:*** $\exists b \in \{0,1\}, X_0, X_1 \in \mathbb{T}_0, i \in \{1,\ldots,t+t_0\}$ *such that*

$$\mathrm{val}_{X_0}(C_{\mathsf{test},0})[i] = 0, \mathrm{val}_{X_1}(C_{\mathsf{test},0})[i] = 1, \mathrm{val}^{\tau}_{X_0}(C_{\mathsf{test},0})[i] = \mathrm{val}^{\tau}_{X_1}(C_{\mathsf{test},0})[i] = b$$

- ***At most toggled output***

$$\exists_{B\in\{0,1\}^t} \forall X \in \mathbb{Z}_2^s \; \exists Y \in \mathbb{Z}_2^{t_0} : \; C^{\tau}_{\mathsf{test},0}(X) = C_{\mathsf{gate}}(X) \| Y + B \| 0^{t_0}$$

Proof. It is easy to see that the same test set $\mathbb{T}_0 = \mathbb{T}_{\mathsf{gate}}$ is a gate covering set for the transformed $C_{\mathsf{test},0}$. Now, according to Theorem 15 in the transformed circuit:

1. Either we get information loss on one of the multi-input gates of C_{gate}: in this case, one of the output wires of $C_{\mathsf{test},0}$ is connected via a COPY gate to the output of the multi-input gate. The information loss is propagated to the output of the COPY gate.

2. Or one of the output wires of C_{gate} always evaluates to a constant value. We observe an information loss on this wire, because it is wire-covered according to the definition of the gate-covering set $\mathbb{T}_0$. If the output wire of a gate in C_{gate} is an output of one of the multi-input gates, then the information loss is propagated to the output of the COPY gate in $C_{\mathsf{test},0}$. Otherwise, the output wire in C_{gate} is already an output wire in $C_{\mathsf{test},0}$. The same follows when an input or an output wire of one of the COPY gates in $C_{\mathsf{test},0}$ are set to a constant.

3. Or every output wire of the gate-covered circuit C_{gate} is at most toggled. In this case, if none of the output wires of the circuit $C_{\mathsf{test},0}$ evaluate to a constant value and constant tamperings were not used on the input and output wires of the COPY gates, then all of the inputs of the transformed are at most toggled. Otherwise, if a constant tampering was used, then we observe an information loss on the output, what concludes the proof.

□

4.6 Minimizing the Number of External Wires

In the previous section, we showed that any circuit transformed into its functionally equivalent gate-covered circuit allows to detect *efficiently* whether it behaves correctly on *all* possible inputs. The testing should follow via input/output testing of a tampered version of the transformed circuit on a limited number of inputs. The construction presented in the Algorithm 4 adds, however, as many output wires to the transformed circuit as the number of gates of the original circuit. This approach is impractical, as implementations

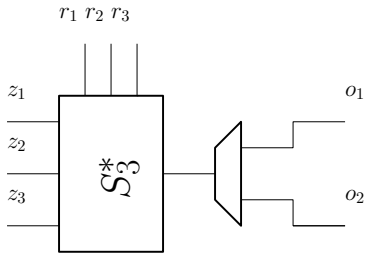


Figure 4.3: Example gadget $S^*_{d,b}$, with $d = 3, b = 2$.

of the circuits usually have a few billions of internal gates, but they can have only a few thousands of input/output wires.

In [9], we proposed a construction of a layered compression gadget that allows to lower the number of output wires while still preserving the guarantees that the errors of computation are detectable. Each layer of their gadget compresses the number of the wires by some factor. For a fixed parameter λ, their construction adds λ layers, each propagating the error to the next layer with probability $\frac{1}{2}$. In summary, for λ layers, a single test detects that an error of computation was propagated to the output of the compressing gadget with probability[5] $\frac{1}{2^\lambda}/2^\lambda$. The layered gadget adds a fixed number of input wires for each construction layer. These new input wires are used as randomness input in the testing process.

In our work, we propose a layered construction that *boosts* the probability of the error propagation by attempting to catch the error multiple times in each layer. At the same time, we are able to preserve the number of the input wires with randomness used in each layer of the construction from [9].

4.7 The Boosted Compression Gadget

To build our construction, we first define a gadget S^*_d that computes a single product tuple, i.e. the output of this gadget is defined as follows: $S^*_d\left((z_i)_{i=1,\ldots,d}, (r_i)_{i=1,\ldots,d}\right) = \sum_{j=1,\ldots,d} z_j r_j$. We use this gadget to create another gadget $S^*_{d,b}$ that computes the product using the S^*_d gadget and then copies its output to b output wires. The Figure 4.3 pictures the construction of the $S^*_{d,b}$ gadget.

Before we proceed, let us introduce an additional notation. Given a vector $R = (r_1, \ldots, r_d) \in \mathbb{Z}_2^d$ and a parameter $b \in \{0\} \cup [d-1]$, we define a shifted vector as $R^{\gg b} = (r_{d-b+1}, \ldots, r_d, r_1, \ldots, r_{d-b})$. For example, for $R = (r_1, r_2, r_3, r_4)$, we have $R^{\gg 1} = (r_4, r_1, r_2, r_3)$, or $R^{\gg 2} = (r_3, r_4, r_1, r_2)$. In the subsequent paragraphs, we will also use $R = R^{\gg 0}$.

We will now show how the $S^*_{d,b}$ gadgets are used to build the boosted version of the compression gadget – the $G^*_{n,\lambda,d,b}$ gadget. Each layer $i \in [\lambda]$ of the $G^*_{n,\lambda,d,b}$ gadget takes as input n_{i-1} wires and outputs n_i wires. The sub-gadgets $S^*_{d,b}$ in each layer have assigned ordering numbers from the set $\{1, 2, \ldots\}$.

[5]The probability that the error is propagated to the output is 2^λ smaller than $\frac{1}{2^\lambda}$ because of the special method of input/output testing introduced in [9] that is required to ensure that not only the error but the information loss is propagated through layers.

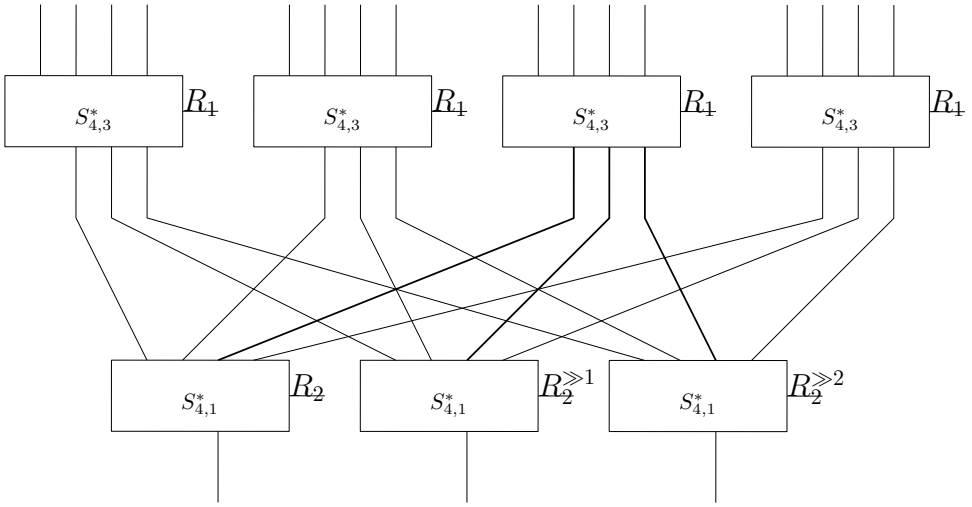


Figure 4.4: The boosted product gadget $G^*_{n,\lambda,d,b}$ with $n = 16$ input wires and $\lambda = 2$ layers. The gadget was built upon $S^*_{d,b}$ sub-gadgets, that take as input $d = 4$ wires and additional $d = 4$ wires with randomness, and copies the output to $b = 3$ distinct locations. In the second layer, the boosting is applied. For example the output of the third gadget in the first layer is copied to $b = 3$ distinct sub-gadgets in the second layer. In the second layer, a single randomness vector is reused $b = 3$ times, as it is shifted by one position in each gadget.

- In the first layer, all of the $n_0 = n$ input wires are inputs of n_0/d gadgets $S^*_{d,b}$, producing $n_1 = n_0 \cdot \frac{b}{d}$ output wires. We can assume that n_0 is divisible by d. Otherwise, the remainder inputs can be input to some $S^*_{d',b}$ with $d' < d$. Each gate $S^*_{d,b}$ in the first layer is given the same randomness input vector R_1.

- In a layer $i \in \{2, \ldots, \lambda - 1\}$, each n_i is divisible by b, as all gadgets in the $i-1$'th layer output b wires. We now require that b output wires of a k'th gadget $S^*_{d,b}$ of the layer $i-1$, go to $S^*_{d,b}$ gadgets with ordering numbers $b\lfloor \frac{k-1}{d} \rfloor + \{1, \ldots, b\}$ in the i'th layer as the $[k-1 \mod d]+1$'th input wire of all b gadgets. In other words, in the i'th layer, n_{i-1} wires output by n_{i-1}/b gadgets $S^*_{d,b}$ from the layer $i-1$ are sent to at most $b\lfloor \frac{n_{i-1}}{bd} \rfloor + b$ gadgets $S^*_{d,b}$ in the layer i. In case only $d' < d$ of the input wires of the last b gadgets $S^*_{d,b}$ are used, they are replaced with gadgets $S^*_{d',b}$.

- The final λ'th layer of the construction is constructed similarly as the previous layers $\{2, \ldots, \lambda - 1\}$, but the parameter b of the $S^*_{d,b}$ gadget is set to 1.

- Each gate $S^*_{d,b}$ in the layers $i \in \{2, \ldots, \lambda\}$ at the k'th position is given $R_i^{\gg k \mod b}$ as the randomness input vector.

An example illustration of the $G^*_{n,\lambda,d,b}$ gadget can be found in the Figure 4.4.

4.7.1 Instantiating the Building Blocks

We are able to instantiate all parts of the $G^*_{n,\lambda,d,b}$ gadget, by reusing the sub-gadgets defined in [9].

We first recall the definitions of the *copying tree* $\triangle_m$ and the *xoring tree* $\triangleright_d$ from [9]. The *copying tree* $\triangle_m$ is a complete binary tree with one input wire and m output wires, where the root is the input wire, the leaves are the output wires and all internal nodes are the COPY gates. The *xoring tree* $\triangleright_d$ is a complete binary tree with d input wires and one output wire, where leaves are the input wires, the root is the output wire and all internal nodes are XOR gates. The Figure 4.5 from [9] pictures the $\triangle_m$, $\triangleright_d$ sub-gadgets.

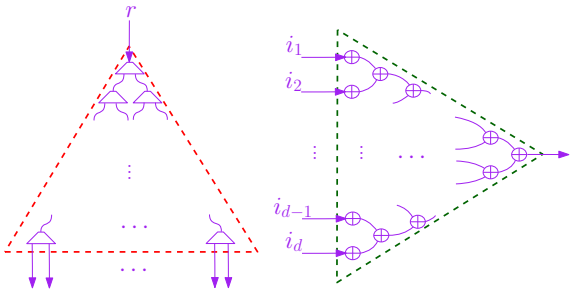


Figure 4.5: The instantiation of the $\triangle, \triangleright$ gadgets with complete trees of the COPY, XOR gates.

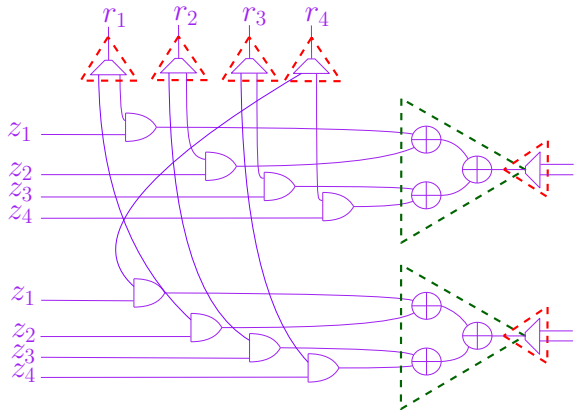


Figure 4.6: The picture shows part of a single layer of the gadget $G^*_{n,\lambda,d,b}$, with $d = 4$, $b = 2$, that consist of two sub-gadgets $S^*_{d,b}$. Each of the sub-gadgets consists of $d = 4$ gates of type AND, a single xoring-tree $\triangleright_d$, and a single copying-tree $\triangle_b$. The randomness input to the layer is reused by first using the copying trees $\triangle$, and then connecting the randomness wires as the appropriate inputs of the AND gates of the $S^*_{d,b}$ sub-gadgets.

It is easy to see that each $S^*_{d,b}$ can be implemented with d gates of type AND, a single xoring-tree $\triangleright_d$, and a single copying-tree $\triangle_b$. A vector of input wires with randomness in each layer of the $G^*_{n,\lambda,d,b}$ gadget can be reused by applying a copying-tree $\triangle$ to each randomness input wire, and then by connecting the output wires of the copying-trees to appropriate inputs of the $S^*_{d,b}$ gadgets. Figure 4.6 illustrates how the $S^*_{d,b}$ sub-gadgets and a single layer of the $G^*_{n,\lambda,d,b}$ gadget are implemented.

4.7.2 Algebraic Notation for Circuit Computations

To formally analyze the security guarantees of the $G^*_{n,\lambda,d,b}$ gadget, we first recall the algebraic notation introduced in [9].

In the algebraic notation, the wires of a circuit C_γ carry elements of a ring of multivariate polynomials in $\mathbb{Z}_2$. The variables of the polynomials are held by the input gates of the circuit. The polynomials held by the wires are recursively determined, similarly as in the definition of the valuation function val, by extrapolating the functions assigned

to the nodes by the labelling function γ. Each function from the set of allowed gates $\mathfrak{G}$ can be naturally extrapolated. For example, output of a gate AND with two input wires carrying polynomials p_1, p_2 can be extrapolated as $p_1 \cdot p_2$.

We recall that *applying a tampering* τ to the circuit C_γ can be easily defined in the algebraic notation - toggling a wire is adding 1 to the polynomial of the wire, and setting a wire to a fixed value 0(1) is defined as setting the polynomial to a constant 0(1).

In the previous work [9], we made a few observations that allowed us to concisely describe possible outputs of their tampered gadgets. For our analysis, we will need to analyze only the output of the $\triangleright, \triangle, S_d^*$ gadgets. To this end, we repeat the analysis presented in [9].

Proposition 17 (Proposition 4 from [9] – Output of the copying trees)**.** *Let $\triangle^\tau$ be given r as input, and r' be its output. Then $r' \in \{0, 1, r, r+1\}$.*

In the Proposition 17, we say that output of the copying tree is either constant, toggled, or the original value of its single input wire, depending on the number of toggling or constant tamperings on the path from the root of the copying tree to the output wire.

Proposition 18 (Proposition 5 from [9] – Output of the xoring trees)**.** *Let $a_1, ..., a_d$ be the input values to $\triangleright^\tau$ and p be its output. Then $p = \beta + \sum_{i=1,...,d} \alpha_i a_i$, where $\alpha_i, \beta \in \{0, 1\}$.*

In the Proposition 18, we say that single output of the xoring tree is a linear combination of its input. If there is constant tampering on a path from some input wire to the output wire, the coefficient α_i of the input value a_i is set to 0, the coefficient β depends on the number of toggling tamperings and values of the constant tamperings.

Proposition 19 (Proposition 6 from [9] – Output of the multiplication gates)**.** *Let (z_i, r_i) be a pair of input wires to some multiplication gate in S^τ and let $mult_i$ denote the output value of this multiplication gate. Then $mult_i = \alpha_i(z_i)r_i + \beta_i(z_i)$, where α_i, β_i are linear functions over $\mathbb{Z}_2$ for all i's.*

The above Proposition states that for a fixed tampering τ, the output value of the multiplication gate m_i can be described as a linear function of r_i. It is a direct consequence of the fact that, given any fixed τ on S^τ, $\tau(z_i) \in \{0, 1, z_i, z_i + 1\}$, $\tau(r_i) \in \{0, 1, r_i, r_i + 1\}$, the output of the multiplication gate is equal to $mult_i = \tau(z_i) \cdot \tau(r_i)$.

Proposition 20 (Output of the S_d^* gadget with applied tampering)**.** *Let p be the output value of the gadget $\triangleright^\tau$ from the construction of $S_d^{*\tau}$ which takes as input values $z_1, z_2, ..., z_d, r_1, r_2, ..., r_d$. Then $p = \beta(z_1, ..., z_d) + \sum_{i=1,...,d} \alpha_i(z_i)r_i$, where α_i (β_i) are linear functions over $\mathbb{Z}_2$.*

In Proposition 20, we conclude that the output value of S_d^* can be understood as a multi-variate polynomial with the variables r_i being dependant only on the $\alpha_i(z_i)$ coefficients. The Proposition 20 gives a result similar to the Proposition 7 from [9].

4.7.3 Information Losing Tuples

Proposition 16 shows that all meaningful errors of computation will result in an information loss on one of the output wires of the precompiled circuit. In other words, if the tampered version of the precompiled circuit triggers an error on some internal wire on any of the inputs, then one is able to find two inputs from the test set, such that one of the output wires of the circuit is evaluated to the same bit, although the evaluation on this wire should differ for these test inputs. In the following sections, we will consider how the information loss is propagated through the boosted compression gadget $G^*_{n,\lambda,d,b}$. For this reason, we first recall the definition of the *information-loosing tuples* from [9]. The notion of information-loosing tuples allows us to abstract the idea of the propagation of the information loss from the computations on the circuit. The non-tampered valuations on selected wires are denoted with the vectors X_i, and the tampered valuations on the same wires are denoted with the vectors Y_i.

Definition 4.7.1 (Definition 4 from [9] – Information-loosing tuples)**.** *We say that $(X_1, ..., X_m; Y_1, ..., Y_m)$ - a tuple of n-ary vectors over $\mathbb{Z}_2$ - is an information-losing tuple if there exist i, j, k, such that $(X_i[k] \neq X_j[k]) \wedge (Y_i[k] = Y_j[k])$. The triple (i, j, k) is called an information-losing witness for $(X_1, ..., X_m; Y_1, ..., Y_m)$.*

4.8 Propagation of the Information Loss in the Boosted Gadget

In this section, we analyze how the information loss on an input wire of the boosted gadget $G^*_{n,\lambda,d,b}$ is propagated through its layers. We start with analyzing the first layer of the $G^*_{n,\lambda,d,b}$ gadget. The first layer of our gadget is composed of the $S^*_{d,b}$ sub-gadgets, each of them connected to the same randomness vector R_1. Let us start with the observation that the sub-gadgets S^*_d, which make the main part of the sub-gadgets $S^*_{d,b}$, define *a single layer of computation* sub-gadget S_d from [9]. For this reason, we first recall the result from [9] that the information loss at one of the input wires of the S_d sub-gadget is propagated with probability at least $1/2$ to one of the output wires of the sub-gadget. This property is assured when the randomness is tossed twice for the same layer.

Theorem 21 (Theorem 2 from [9] – Information loss propagation)**.** *Let $(X_1, ..., X_m; X^\tau_1, ..., X^\tau_m)$ be an information-losing tuple. For $z \in \{1, ..., m\}$, let R_z, Q_z be vectors in $\mathbb{Z}^d_2$ chosen independently and uniformly at random. Let*

$$Y_z = S_d(X_z|R_z),\ Y_{z+m} = S_d(X_z|Q_z),\ Y^\tau_z = S^\tau_d(X^\tau_z|R_z),\ Y^\tau_{z+m} = S^\tau_d(X^\tau_z|Q_z),$$

for $i = 1, ..., z$. Then $(Y_1, ..., Y_{2m}; Y^\tau_1, ..., Y^\tau_{2m})$ is an information-losing tuple with probability at least $\frac{1}{2}$ over $(R_z, Q_z)_{z\in\{1,...,m\}}$.

In the boosted compression gadget, once the information loss passes through one of the sub-gadgets S^*_d to its output wire in the first layer, it is copied to b input wires of the second layer, say wires $z_1, \ldots, z_b$. By the construction of the $G^*_{n,\lambda,d,b}$ gadget, in the second layer, the wires $z_j \in \{z_1, \ldots, z_b\}$ are connected to distinct S^*_d sub-gadgets in a way that different randomness inputs $r_{2,j}$ of the same randomness vector R_2 are used

to compute product $z_j \cdot r_{2,j}$. This effect is achieved by shifting the randomness vector R_2 appropriately for every S_d^* sub-gadget. The same reasoning applies to every pair of subsequent layers $(i-1, i)$, where the layer i is connected to a randomness vector R_i.

Let us denote with $R_{i,j}$ a random variable describing a single randomness input bit at the j'th position of a random variable R_i. In contrast to the result recalled in the Theorem 21, in the next step we study the probability of the information loss propagation through the S_d^* sub-gadget over a single *bit* $R_{i,j}$ of the randomness, not the whole randomness input R_i.

Theorem 22 (Pointed information loss propagation)**.** *Let $(X_1, ..., X_m; X_1^\tau, ..., X_m^\tau)$ be an information-loosing tuple with a witness (i, j, k), and the vectors of the tuple in $\mathbb{Z}_2^d$. For $z \in \{1, \ldots, m\}$, let R_z, Q_z be vectors over $\mathbb{Z}_2^d$ with a randomly and uniformly chosen bits at position k, i.e. $R_z = \{r_{z,1}, \ldots, R_{z,k}, \ldots, r_{z,d}\}, Q_z = \{q_{z,1}, \ldots, Q_{z,k}, \ldots, r_{z,d}\}$ where $R_{z,k}, Q_{z,k}$ are random variables in $\mathbb{Z}_2$, and all other bits of R_z, Q_z – denoted as the $R_{z,\overline{k}}, Q_{z,\overline{k}}$ vectors – can be selected before randomly sampling $\{R_{z,k}, Q_{z,k}\}_{z\in\{1,\ldots,m\}}$. Let*

$$Y_z = S_d^*(X_z|R_z),\ Y_{z+m} = S_d^*(X_z|Q_z)$$

$$Y_i^\tau = S_d^{*\tau}(X_z^\tau|R_z^\tau),\ Y_{z+m}^\tau = S_d^{*\tau}(X_z^\tau|Q_z^\tau),$$

for $z = 1, ..., m$. Then $(Y_1, ..., Y_{2m}; Y_1^\tau, ..., Y_{2m}^\tau)$ is an information-loosing tuple with probability at least $\frac{1}{4}$ over $(R_{z,k}, Q_{z,k})_{z\in\{1,\ldots,m\}}$.

Proof. Let us denote with o the output wire of the S_d^* sub-gadget. Let (i, j, k) be a witness of information loss for $(X_1, ..., X_m; X_1^\tau, ..., X_m^\tau)$. Then

$$(X_i[k] \neq X_j[k]) \wedge X_i^\tau[k] = X_j^\tau[k]. \tag{4.4}$$

Let R_i, R_j be vectors with a uniformly random bit at position k. Observe, that for either i or j (we choose i below), for any choice of randomness bits other than the randomness bit at the k'th position - $R_{i,\overline{k}} \in \{0,1\}^{d-1}$, the difference between the correct and the wrong evaluation $\mathrm{DIFF_i}(R_{i,k}, R_{i,\overline{k}})$ on this output wire can be described as:

$$\mathrm{DIFF_i}(R_{i,k}, R_{i,\overline{k}}) = \mathrm{val}_{X_i|R_{i,k},R_{i,\overline{k}}}(o) - \mathrm{val}^\tau_{X_i'|R_{i,k},R_{i,\overline{k}}}(o) = R_{i,k} + \epsilon_i + \sum_{t=1,\ldots,d;t\neq k} \delta_t r_{i,t}, \tag{4.5}$$

for some $\epsilon_i, \delta_t \in \{0,1\}$. The above implies that for any choice of $R_{i,\overline{k}} \in \{0,1\}^{d-1}$, the probability over $R_{i,k}$ that the wrong evaluation occurs is:

$$\Pr\Big[\mathrm{DIFF_i}(R_{i,k}, R_{i,\overline{k}}) = 1\Big] = \frac{1}{2}.$$

In other words, for the index i and exactly half of the choices of $R_{i,k}$, there will occur an error on the output wire o. If the above holds for the index i, then for the index j, for any choice of randomness bits other than the randomness bit at the k'th position - $R_{j,\overline{k}} \in \{0,1\}^{d-1}$:

$$\mathrm{DIFF_j}(R_{j,k}, R_{j,\overline{k}}) = \mathrm{val}_{X_j|R_{j,k},R_{j,\overline{k}}}(o) - \mathrm{val}^\tau_{X_j|R_{j,k},R_{j,\overline{k}}}(o) = \kappa_k R_{j,k} + \epsilon_j + \sum_{t=1,\ldots,d,t\neq k} \kappa_t r_{j,t},$$

where $\kappa_t, \epsilon_j \in \{0,1\}$. It means that for any choice of $R_{j,\overline{k}} \in \{0,1\}^{d-1}$:

$$\Pr\Big[\mathrm{DIFF}_{\mathrm{j}}(R_{j,k}, R_{j,\overline{k}}) = 1\Big] \in \{0, \frac{1}{2}, 1\}.$$

Finally we know, that for any choice of $R_{i,\overline{k}}, R_{j,\overline{k}} \in \mathbb{Z}_2^{d-1}$:

$$\{\frac{1}{2}\} \subseteq \{\Pr\Big[\mathrm{val}_{X_i|R_{i,k},R_{i,\overline{k}}}(o) = 0\Big], \Pr\Big[\mathrm{val}_{X_j|R_{j,k},R_{j,\overline{k}}}(o) = 0\Big]\} \subseteq \{0, \frac{1}{2}, 1\}. \tag{4.6}$$

We finally observe that for a fixed X_i (including $X_i = X_j$), and a fixed tampering τ applied to S_d^*, for any $R_{i,\overline{k}} \in \mathbb{Z}_2^{d-1}$, whenever $\Pr\Big[\mathrm{DIFF}_{\mathrm{i}}(R_{i,k}, R_{i,\overline{k}}) = 1\Big] = \frac{1}{2}$, then either:

1. for some $b \in \{0,1\}$ it holds that $\mathrm{val}^{\tau}_{X_i|R_{i,k}=b,R_{i,\overline{k}}}(o) \neq \mathrm{val}_{X_i|R_{i,k}=b,R_{i,\overline{k}}}(o)$ and $\mathrm{val}_{X_i|R_{i,k}=b,R_{i,\overline{k}}}(o) = 0$. In other words, the evaluation error happens when $R_{i,k} = b$, and the correct evaluation should output 0. In this case, we say that flag $\mathrm{FLAG}(X_i, R_{i,\overline{k}}) = \mathbb{0}$ is set; or

2. for some $b \in \{0,1\}$ it holds that $\mathrm{val}^{\tau}_{X_i|R_{i,k}=b,R_{i,\overline{k}}}(o) \neq \mathrm{val}_{X_i|R_{i,k}=b,R_{i,\overline{k}}}(o)$ and $\mathrm{val}_{X_i|R_{i,k}=b,R_{i,\overline{k}}}(o) = 1$. In other words, the evaluation error happens when $R_{i,k} = b$, and the correct evaluation should output 1. In this case, we say that flag $\mathrm{FLAG}(X_i, R_{i,\overline{k}}) = \mathbb{1}$ is set.

Given the above, we can estimate the probability that the output tuple $(Y_1, ..., Y_{2m}; Y_1^{\tau}, ..., Y_{2m}^{\tau})$ contains an information error, i.e. an event $\mathcal{E}_{\text{INF-LOSS}}$ occurs. For any τ applied to the sub-gadget S_d^*, and random variables $R_i, Q_i R_j, Q_j$ defined as in the theorem statement, we have $\Pr\Big[\mathrm{val}_{X_i|R_{i,k},R_{i,\overline{k}}}(o) = 0\Big] = \frac{1}{2}$, $\Pr\Big[\mathrm{val}_{X_i|Q_{i,k},Q_{i,\overline{k}}}(o) = 0\Big] = \frac{1}{2}$, and $\Pr\Big[\mathrm{DIFF}_{\mathrm{j}}(R_{j,k}, R_{j,\overline{k}}) = 1\Big] \in \{0, \frac{1}{2}, 1\}$, $\Pr\Big[\mathrm{DIFF}_{\mathrm{j}}(Q_{j,k}, Q_{j,\overline{k}}) = 1\Big] \in \{0, \frac{1}{2}, 1\}$. In the analysis, we show that for any choice of the order of sampling $R_{i,k}, Q_{i,k}, R_{j,k}, Q_{j,k}$, as well as any $R_{i,\overline{k}}, Q_{i,\overline{k}}, R_{j,\overline{k}}, Q_{j,\overline{k}}$ chosen before sampling $R_{i,k}, Q_{i,k}, R_{j,k}, Q_{j,k}$, the event $\mathcal{E}_{\text{INF-LOSS}}$ will occur with high probability.

Now, (A) EITHER the input wire at $R_{i,k}$ (and $Q_{i,k}$) evaluates to 1, i.e. $X_i[k]$ equals 1 (and $X_j[k]$ equals 0). In this case $\Pr\Big[\mathrm{val}_{X_i|R_{i,k},R_{i,\overline{k}}}(o) = 0\Big] = \frac{1}{2}$; and $\Pr\Big[\mathrm{val}_{X_j|R_{j,k},R_{j,\overline{k}}}(o) = 0\Big] \in \{0,1\}$ (the same follows for $Q_{i,k}, Q_{i,\overline{k}}$, and $Q_{j,k}, Q_{j,\overline{k}}$).

1. Whenever $\mathrm{FLAG}(X_i, R_{i,\overline{k}}) = \mathrm{FLAG}(X_i, Q_{i,\overline{k}}) = b$, then with high probability in one execution we obtain a correct evaluation to $b-1$, and in the other execution we obtain an incorrect evaluation to $b-1$ (since $\Pr\Big[\mathrm{val}_{X_i|R_{i,k},R_{i,\overline{k}}}(o) = 0\Big] = \frac{1}{2}$, and the error occurs on b). We can conclude that $\Pr[\mathcal{E}_{\text{INF-LOSS}}] \geqslant \frac{1}{2}$.

2. Whenever $\mathrm{FLAG}(X_i, R_{i,\overline{k}}) \neq \mathrm{FLAG}(X_i, Q_{i,\overline{k}})$, and $\Pr\Big[\mathrm{DIFF}_{\mathrm{j}}(R_{j,k}, R_{j,\overline{k}}) = 1\Big] = \frac{1}{2}$ (or $\Pr\Big[\mathrm{DIFF}_{\mathrm{j}}(Q_{j,k}, Q_{j,\overline{k}}) = 1\Big] = \frac{1}{2}$), then $\mathrm{FLAG}(X_j, R_{j,\overline{k}})$ (or $\mathrm{FLAG}(X_j, Q_{j,\overline{k}})$) equals to $\mathrm{FLAG}(X_i, R_{i,\overline{k}})$ or $\mathrm{FLAG}(X_i, Q_{i,\overline{k}})$. By a similar argument as in the previous point, we get $\Pr[\mathcal{E}_{\text{INF-LOSS}}] \geqslant \frac{1}{2}$.

3. Whenever $\mathrm{FLAG}(X_i, R_{i,\overline{k}}) \neq \mathrm{FLAG}(X_i, Q_{i,\overline{k}})$ and both $\Pr\Big[\mathrm{DIFF_j}(R_{j,k}, R_{j,\overline{k}}) = 1\Big]$, $\Pr\Big[\mathrm{DIFF_j}(Q_{j,k}, Q_{j,\overline{k}}) = 1\Big] \in \{0,1\}$. In this case, from evaluations on X_j, we get an evaluation to some bit (either correct or incorrect). As $\mathrm{FLAG}(X_i, R_{i,\overline{k}}) \neq \mathrm{FLAG}(X_i, Q_{i,\overline{k}})$, we get an evaluation to every bit (both correct and incorrect) with probability 1/2. One of the evaluations will thus match with probability $\frac{1}{2}$ the given evaluation and we get $\Pr[\mathcal{E}_{\text{INF-LOSS}}] \geqslant \frac{1}{2}$.

(B) OR the input wire at $R_{i,k}$ (and $Q_{i,k}$) evaluates to 0, i.e. $X_i[k]$ equals 0 (and $X_j[k]$ equals 1). In this case $\mathrm{val}_{X_i|R_{i,k},R_{i,\overline{k}}}(o) \in \{0,1\}$ and $\Pr\Big[\mathrm{val}_{X_j|R_{j,k},R_{j,\overline{k}}}(o) = 0\Big] = \frac{1}{2}$ (the same follows for $Q_{i,k}, Q_{i,\overline{k}}$, and $Q_{j,k}, Q_{j,\overline{k}}$).

Whenever both $\Pr\Big[\mathrm{DIFF_j}(R_{j,k}, R_{j,\overline{k}}) = 1\Big] = \frac{1}{2}$ and $\Pr\Big[\mathrm{DIFF_j}(Q_{j,k}, Q_{j,\overline{k}}) = 1\Big] = \frac{1}{2}$, we have $\Pr[\mathcal{E}_{\text{INF-LOSS}}] \geqslant \frac{1}{2}$ by the case analysis as in the point (A).

Next, we observe that two samplings of $R_{i,k}, Q_{i,k}$ will give a pair of correct evaluation and an error evaluation with probability $\frac{1}{2}$. When both $\Pr\Big[\mathrm{DIFF_j}(R_{j,k}, R_{j,\overline{k}}) = 1\Big]$, $\Pr\Big[\mathrm{DIFF_j}(Q_{j,k}, Q_{j,\overline{k}}) = 1\Big] \in \{0,1\}$, then sampling of $Q_{j,k}, Q_{j,\overline{k}}$ will match either the correct evaluation or the wrong evaluation from sampling $R_{i,k}, Q_{i,k}$ with probability $\frac{1}{2} + (1 - \frac{1}{2})\frac{1}{2} = 3/4$. In total, we have $\Pr[\mathcal{E}_{\text{INF-LOSS}}] \geqslant \frac{3}{8}$.

Finally, we consider a case when for only one of R_j and Q_j (say R_j) the probability that one gets an error on the output is $\Pr\Big[\mathrm{DIFF_j}(R_{j,k}, R_{j,\overline{k}}) = 1\Big] = \frac{1}{2}$, and for the other one (respectively, Q_j) the probability that one gets an error on the output $\Pr\Big[\mathrm{DIFF_j}(Q_{j,k}, Q_{j,\overline{k}}) = 1\Big]$ is either 0 or 1. Again, the first two samplings of $R_{i,k}$ will give a pair of correct evaluations and an error evaluation with probability $\frac{1}{2}$, and then a sampling of $Q_{j,k}$ that is always correct or wrong will match either the correct evaluation or the wrong evaluation from sampling $R_{i,k}, Q_{i,k}$ with probability $\frac{1}{2}$. In total, we have $\Pr[\mathcal{E}_{\text{INF-LOSS}}] \geqslant \frac{1}{4}$. □

4.9 The Complete Construction

The building blocks defined and the results given in the previous sections allow us to build a boosted version of the compiler that compiles any circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ into another functionally equivalent circuit $C_{\mathsf{test},\lambda} : \mathbb{Z}_2^{s+s'} \to \mathbb{Z}_2^{t+t'}$ such that for any non-trivial tampering of the circuit $C_{\mathsf{test},\lambda}$, running the testing procedure on the tampered $C_{\mathsf{test},\lambda}$, one always detects an error with high probability.

Theorem 23 (Testing Probability of the Boosted Circuit)**.** *On input circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ along with parameter λ, Algorithm 5 outputs a circuit $C_{\mathsf{test},\lambda} : \mathbb{Z}_2^{s+s_g+s_\lambda} \to \mathbb{Z}_2^{t+t_\lambda}$ such that for any tampering τ of $C_{\mathsf{test},\lambda}$ if*

$$\exists X \in \mathbb{Z}_2^s : C^{\tau}_{\mathsf{test},\lambda}(X||0^{s_g+s_\lambda}) \neq C_{\mathsf{test},\lambda}(X||0^{s_g+s_\lambda})$$

then when observing behaviour of the circuit $C_{\mathsf{test},\lambda}$ on its test set $\mathbb{T}_{\mathsf{test}}$,

- ***Either the output is wrong:***

$$\exists X \in \mathbb{T}_{\mathsf{test}} : C^{\tau}_{\mathsf{test},\lambda}(X||0^{s_\lambda}) \neq C_{\mathsf{test},\lambda}(X||0^{s_\lambda}),$$

Algorithm 5: The Compiler

Input: $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t, \lambda, d, b$

Output: $C_{\mathsf{test},\lambda}$

1 Compile circuit C into a gate-covered $C_{\mathsf{gate}} : \mathbb{Z}_2^{s+s_g} \to \mathbb{Z}_2^t$ with a test-set $\mathbb{T}_{\mathsf{gate}}$, by running Algorithm A on it.

2 Add the COPY gates that route the information loss in the gate-covered circuit to the testing gadget, by running Algorithm 4 on the pair $C_{\mathsf{gate}}, \mathbb{T}_{\mathsf{gate}}$. This procedure provides a circuit with additional t_0 output bits $(C_{\mathsf{test},0} : \mathbb{Z}_2^{s+s_g} \to \mathbb{Z}_2^{t+t_0})$ and a test set $\mathbb{T}_0$.

3 Append the $G^*_{n,\lambda,d,b}$ gadget to the t_0 wires added in the previous step, where $n = t_0$. This step adds s_λ wires to the input of the circuit, but replaces the t_0 output bits created in the previous step with t_λ new output bits, producing a circuit $C_{\mathsf{test},\lambda} : \mathbb{Z}_2^{s+s_g+s_\lambda} \to \mathbb{Z}_2^{t+t_\lambda}$

4 **return** $C_{\mathsf{test},\lambda}$

- ***or the testing gadget detects an inconsistency:***

$$\exists X \in \mathbb{T}_{\mathsf{test}} : \Pr_{R \leftarrow \mathbb{Z}_2^{s_\lambda}} \left[C^\tau_{\mathsf{test},\lambda}(X||R) \neq C_{\mathsf{test},\lambda}(X||R) \right] \geqslant \frac{1}{2} \left(1 - (3/4)^b\right)^{\lambda-1} / 2^\lambda.$$

Proof. By the Proposition 16 we know that either we observe an information loss on the first t bits of the intermediary circuit $C_{\mathsf{test},0}$ or its output is toggled, or we observe information loss on t_0 wires added during Step 2 of the Algorithm 5, or the output is always correct. Any error on the first t bits of the circuit is detected on at least one query from $\mathbb{T}_0 \subseteq \mathbb{T}_{\mathsf{test}}$ (by the properties of the Algorithm A and the Proposition 16). Next, we append the gadget $G^*_{n,\lambda,d,b}$ to the remaining $n = t_0$ wires of the construction. By Theorem 21, we know that the information loss survives with probability $1/2$ through the *first* layer of the gadget $G^*_{n,\lambda,d,b}$ when queried with fresh randomness twice. Now, whenever the information loss passes through one of the S^*_d gadgets in a layer $i-1$, it is copied to b locations in a layer i. By the construction of the $G^*_{n,\lambda,d,b}$ gadget, in each of these locations in the layer i, the wire with the information loss is connected to an AND gate with a *distinct* randomness input. Thus, by the Theorem 23, the information loss is propagated with probability $1-(3/4)^b$ through this layer (when queried with fresh randomness twice). We can conclude that the information loss survives with probability $\frac{1}{2}\left(1-(3/4)^b\right)^{\lambda-1}$ if we query with two fresh randomness vectors in each layer. If we query with only one random string in each layer, the error showing up on the output is $\frac{1}{2}\left(1-(3/4)^b\right)^{\lambda-1}/2^\lambda$. □

Testing procedure. Given any circuit $C^\tau_{\mathsf{test},\lambda}$ with any tampering τ on its wires we test it by querying it on all the test inputs in $\mathbb{T}_{\mathsf{test}}$ along with uniformly random $R \in \mathbb{Z}_2^{s_\lambda}$. We can repeat the testing procedure κ times with fresh randomness to get the probability of catching an error $1 - \left(1 - \frac{1}{2^{\lambda+1}}\left(1-(3/4)^b\right)^{\lambda-1}\right)^\kappa$.

Circuit parameters. For any circuit $C : \mathbb{Z}_2^s \to \mathbb{Z}_2^t$ with n gates, using the Algorithm 5 with parameter λ. The first step of the Algorithm produces a gate-covered circuit C_{gate}

with 5 new input bits and a test set of size 6 and creates a circuit of size $\approx 7n$ gates (see Section 4.4). The second step of the algorithm adds XOR and COPY gate to every nonlinear gate of the circuit, adding $\approx 2 \cdot 4n$ gates and roughly $\approx 4n$ output wires (in the previous estimation at least $3n$ out of $7n$ gates are the COPY gates). The third step of the algorithm replaces the $4n$ intermediary wires with $\lambda \cdot d$ input bits. After the first layer $n_0 = 4n$ input wires are transformed to $n_1 = b \cdot \frac{n_0}{d}$ output wires. In the next layers $i \in \{2, \ldots, \lambda-1\}$, n_{i-1} input wires are transformed to at most $n_i \leqslant b \cdot \left(\frac{n_{i-1}}{d} + b\right) = \frac{n_{i-1}}{\frac{d}{b}} + b^2$ output wires (see Section 4.7). In the last layer $n_\lambda \leqslant \frac{1}{b}\left(\frac{n_{\lambda-1}}{\frac{d}{b}} + b^2\right)$. In total, after applying $\lambda > 1$ layers of the construction, the number of additional output wires can be bounded with $t_\lambda = n_\lambda \leqslant \frac{1}{b} \cdot \left(\frac{4n}{\left(\frac{d}{b}\right)^\lambda} + b^2 \cdot \frac{\frac{d}{b} - \left(\frac{b}{d}\right)^{\lambda-1}}{\frac{d}{b}-1}\right) \leqslant \frac{4n}{d\left(\frac{d}{b}\right)^{\lambda-1}} + \frac{d \cdot b}{d-b}$. The layered compression gadget with λ layers adds roughly $(3 \cdot 4n + b \cdot n)\frac{1-\left(\frac{b}{d}\right)^\lambda}{1-\frac{b}{d}}$ new gates to the circuit, whereas at most $3 \cdot 4n \cdot 2$ new gates[6] are added by the layered construction in [9].

[6] [9] claimed that at most $4n \cdot 2$ gates are added to the layered construction, but this estimation counted only AND gates, and in this chapter we take into account the AND, XOR, and the COPY gates.

Chapter 5

Final Conclusions and Outlook

In this dissertation, we studied the security of systems built on blockchains from a few novel viewpoints.

In the context of *reliability of rating systems used by blockchain trading platforms*, we conducted in Chapter 2 a formal study of the manipulability of the Fairness-Goodnes Algorithm (*FGA*). We proposed the first complete axiomatization of the *FGA* measure. Our axiomatization is built upon, among others, the properties of homogeneously and unanimously rated nodes and the properties of the rating nodes that achieve constant rating errors. We proved that our axioms uniquely determine the *FGA* measure. Furthermore, we studied direct and indirect manipulation attacks on the *FGA* measure. In the direct attack, the attacker directly manipulates the rating of the target node. In the indirect one, the attacker aims to manipulate a rating of the target node without directly adding an edge to this node. We derived analytical results concerning the strength of the direct attacks and weakness of the indirect attacks in the networks where each node has minimum k neighbors (in and out). Finally, we experimentally analyzed the strength of direct attacks and analyzed two different greedy algorithms for indirect attacks. The experiments showed that *FGA* may be manipulated indirectly in real-life networks, but only at a high cost.

Overall, a higher-level insight from the theoretical and experimental analysis is that *FGA* is rather difficult to manipulate indirectly in real-life networks. More generally, while worst-case hardness results are common in the literature and various other tools turned out to be easily manipulable by well-crafted heuristics for direct attacks [17, 113, 114, e.g.], to the best of our knowledge, our study of indirect attacks is a novel approach. The immunity of the *FGA* measure against indirect attacks not only provides a good argument for using the *FGA* measure in practice, but also fosters a more careful study of other centrality measures.

In the future, it would be interesting to undertake a comparison of various candidate measures found in existing literature with regard to their manipulability. This work could lead to developing a more comprehensive approach to understanding the manipulability of all weighted ranking functions. It may further examine novel ranking functions tailored for blockchain trading platforms, considering factors beyond user opinions. These additional factors include analyzing user transaction history, which may provide insights into potential money laundering or artificial asset price inflation. Moreover, exploring the manipulability of ranking functions based on machine learning techniques would be an

exciting avenue of research. Lastly, we encourage investigations into the axiomatization of ranking functions, as this could enhance our comprehension of their inherent characteristics or even pave the way for creating a new ranking function with robust axiomatic properties.

In the context of *algorithms that ensure security against distributed cryptography*, we initiated the formal study of individual cryptography in Chapter 3. We achieved individually secure constructions based on the assumption of the existence of MPC- and TEE-hard functions. In this work, we assumed MPC- and TEE-hardness is assured by the existence of a device that allows for massive parallel computations of a hash function on selected inputs. Given this assumption, to visualize how to implement functionalities that take into account the concept of individual cryptography, we formally defined the notions of Proof of Individual Knowledge and Secret Sharing with Snitching. Informally speaking, the Proof of Individual Knowledge ensures that some secrets are stored individually on a single machine. It may be further used to incentivize the users not to disclose the secret to other users, which may be particularly useful in settings like subscription fee-based systems or online voting systems. Secret Sharing with Snitching is a primitive that incentivizes the users not to disclose the secrets to third parties before a selected deadline and may provide a cheaper alternative to constructions like Verifiable Delay Functions [23].

It would be exciting to search for particular computational problems that are MPC- or TEE-hard in practice and do not require the usage of specialized hardware (one good candidate would be functions that require many memory lookups, as this assumption may allow to abandon the use specialized hardware), and to improve the practical parameters that we achieve. The new MPC-hard assumptions may lead to more practical constructions, as the current ones could not be used at a larger scale, e.g., by video streaming services that would like to prevent account sharing, as the requirement of specialized and energy-consuming hardware may cause too high costs for individual users. The other task is to leverage the current construction of SSS to a construction that has a formal model for incentives, possibly achieving a construction fully integrated with the blockchain. The mpc hardness and the concept of individual cryptography can have broad applications. We encourage the reader to study how the MPC-hard functions may affect the security of existing protocols (in particular, the ones that assume that secrets are secretly generated by independent parties that are not supposed to share their knowledge) and to make an attempt to find novel individually secure primitives. This research direction is, in a sense, a complementary effort to the search for "MPC-friendly primitives" (see, e.g., [72]). Finally, we see that it would be interesting to see how the concept of individual cryptography can be applied to other related concepts like colusion-freeness [85] or collusion-detterence [91].

In the context of the security of *devices used by blockchain users*, we have studied the construction of a compiler that ensures security against so-called wire-tampering attacks in Chapter 4. We were able to construct a compiler that transforms circuits into efficiently testable circuits with amplified (boosted) security guarantees (compared to the guarantees achieved by [9]) at a moderate cost of an additional number of gates in the transformed circuits. Our construction leverages the concept of testing, which ensures security guarantees against a stronger adversary than the self-correcting construction from [75] at a lower cost. On the other hand, the construction from [75] provides security

through self-correction during the execution of the circuit without the need for periodical testing. We want to stress again that constructing a highly efficient testing construction would require exploiting the trade-off between the amplified security guarantees assured by the layers of our construction and the size-efficiency of the layers of the construction presented in [9].

This and the cited work [9] give a compiler for ETCs and solve one of the two open problems given in [8], as they *do not* leverage the error conductivity assumption. On the other hand, the second problem given in [8] is still open. It focuses on dealing with ETCs that should be secure against an adversary that is able to tamper with the specification of the gates of a tested circuit. The idea of circuit testing itself is a broad field to explore. We particularly encourage the reader to explore a range of opportunities between the weak testing guarantees achieved in a very strong adversarial model presented in works like [29] and strong security guarantees in a visibly weaker security model. The whole spectrum of possible functionalities and tampering models that lie between [29] and our work may pave a new way for the idea of circuit testing. In [29], the authors focused on the security of a selected functionality (PRG generator), and they allowed a very general adversary that is able to replace the functionality of a single device with arbitrary functionality. On the other hand, the security guarantees achieved by [29] are relatively weak (for T tests, the adversary is allowed to output approximately $1/T$ wrong outputs during the usage of the device).

Bibliography

[1] URL: https://www.cisa.gov/known-exploited-vulnerabilities-catalog.

[2] D. Aggarwal, Y. Dodis, and S. Lovett. "Non-malleable codes from additive combinatorics". In: *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*. Ed. by D. B. Shmoys. ACM, 2014, pp. 774–783. DOI: 10.1145/2591796.2591804.

[3] J. Alwen, B. Chen, K. Pietrzak, L. Reyzin, and S. Tessaro. "Scrypt Is Maximally Memory-Hard". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*. Ed. by J. Coron and J. B. Nielsen. Vol. 10212. Lecture Notes in Computer Science. 2017, pp. 33–62. DOI: 10.1007/978-3-319-56617-7_2. URL: https://doi.org/10.1007/978-3-319-56617-7_2.

[4] wiki article. *Bitcoin-OTC*. [Online; accessed June-2021]. 2021. URL: https://en.bitcoin.it/wiki/Bitcoin-OTC.

[5] G. Ateniese, A. Kiayias, B. Magri, Y. Tselekounis, and D. Venturi. "Secure Outsourcing of Cryptographic Circuits Manufacturing". In: *ProvSec*. Ed. by J. Baek, W. Susilo, and J. Kim. Vol. 11192. Lecture Notes in Computer Science. Springer, 2018, pp. 75–93. DOI: 10.1007/978-3-030-01446-9_5.

[6] J. Austgen, A. Fábrega, S. Allen, K. Babel, M. Kelkar, and A. Juels. *DAO Decentralization: Voting-Bloc Entropy, Bribery, and Dark DAOs*. 2023. arXiv: 2311.03530 [cs.CR].

[7] R. Bahmani et al. "Secure Multiparty Computation from SGX". In: *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*. Ed. by A. Kiayias. Vol. 10322. Lecture Notes in Computer Science. Springer, 2017, pp. 477–497. DOI: 10.1007/978-3-319-70972-7_27. URL: https://doi.org/10.1007/978-3-319-70972-7_27.

[8] M. A. Baig, S. Chakraborty, S. Dziembowski, M. Gałązka, T. Lizurej, and K. Pietrzak. "Efficiently Testable Circuits". In: *ITCS - Innovations in Theoretical Computer Science*. 2023.

[9] M. A. Baig, S. Chakraborty, S. Dziembowski, M. Gałązka, T. Lizurej, and K. Pietrzak. *Efficiently Testable Circuits without Conductivity*. 2023. URL: https://dl.acm.org/doi/proceedings/10.1007/978-3-031-48621-0.

[10] M. Ball, D. Dachman-Soled, S. Guo, T. Malkin, and L. Tan. "Non-Malleable Codes for Small-Depth Circuits". In: *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*. Ed. by M. Thorup. IEEE Computer Society, 2018, pp. 826–837. DOI: 10.1109/FOCS.2018.00083.

[11] C. Baum, B. David, and R. Dowsley. "Insured MPC: Efficient Secure Computation with Financial Penalties". In: *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*. Ed. by J. Bonneau and N. Heninger. Vol. 12059. Lecture Notes in Computer Science. Springer, 2020, pp. 404–420. DOI: 10.1007/978-3-030-51280-4_22. URL: https://doi.org/10.1007/978-3-030-51280-4_22.

[12] M. Bellare and O. Goldreich. "On Defining Proofs of Knowledge". In: *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Ed. by E. F. Brickell. Vol. 740. Lecture Notes in Computer Science. Springer, 1992, pp. 390–420. DOI: 10.1007/3-540-48071-4_28. URL: https://doi.org/10.1007/3-540-48071-4_28.

[13] M. Bellare and P. Rogaway. "Optimal Asymmetric Encryption". In: *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*. Ed. by A. D. Santis. Vol. 950. Lecture Notes in Computer Science. Springer, 1994, pp. 92–111. DOI: 10.1007/BFb0053428.

[14] M. Bellare and P. Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*. Ed. by D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby. ACM, 1993, pp. 62–73. DOI: 10.1145/168588.168596.

[15] M. Ben-Or, S. Goldwasser, and A. Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. Ed. by J. Simon. ACM, 1988, pp. 1–10. DOI: 10.1145/62212.62213.

[16] F. Benhamouda et al. *Can a Public Blockchain Keep a Secret?* Cryptology ePrint Archive, Paper 2020/464. https://eprint.iacr.org/2020/464. 2020. URL: https://eprint.iacr.org/2020/464.

[17] E. Bergamini, P. Crescenzi, G. D'angelo, H. Meyerhenke, L. Severini, and Y. Velaj. "Improving the betweenness centrality of a node by adding links". In: *Journal of Experimental Algorithmics (JEA)* 23 (2018), pp. 1–5.

[18] F. Berti, C. Guo, T. Peters, and F. Standaert. "Efficient Leakage-Resilient MACs Without Idealized Assumptions". In: *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*. Ed. by M. Tibouchi and H. Wang. Vol. 13091. Lecture Notes in Computer Science. Springer, 2021, pp. 95–123. DOI: 10.1007/978-3-030-92075-3_4.

[19] S. Bhunia and M Tehranipoor. “The Hardware Trojan War”. In: *Cham„ Switzerland: Springer* (2018).

[20] E. Biham and A. Shamir. “Differential Fault Analysis of Secret Key Cryptosystems”. In: *CRYPTO*. Vol. 1294. Lecture Notes in Computer Science. Springer, 1997, pp. 513–525.

[21] N. Bitansky et al. “The Hunting of the SNARK”. In: *J. Cryptol.* 30.4 (2017), pp. 989–1066. DOI: 10.1007/s00145-016-9241-9.

[22] M. Blum, P. Feldman, and S. Micali. “Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)”. In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. Ed. by J. Simon. ACM, 1988, pp. 103–112. DOI: 10.1145/62212.62222.

[23] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. *Verifiable Delay Functions*. Cryptology ePrint Archive, Paper 2018/601. https://eprint.iacr.org/2018/601. 2018. URL: https://eprint.iacr.org/2018/601.

[24] D. Boneh, R. A. DeMillo, and R. J. Lipton. “On the Importance of Eliminating Errors in Cryptographic Computations”. In: *J. Cryptol.* 14.2 (2001), pp. 101–119.

[25] E. Boyle, G. Segev, and D. Wichs. “Fully Leakage-Resilient Signatures”. In: *J. Cryptol.* 26.3 (2013), pp. 513–558. DOI: 10.1007/s00145-012-9136-3.

[26] G. Brassard, D. Chaum, and C. Crépeau. “Minimum Disclosure Proofs of Knowledge”. In: *J. Comput. Syst. Sci.* 37.2 (1988), pp. 156–189. DOI: 10.1016/0022-0000(88)90005-0.

[27] M. Bushnell and V. Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Vol. 17. Springer Science & Business Media, 2004.

[28] Y. Cai and D. Zhu. “Fraud detections for online businesses: a perspective from blockchain technology”. In: *Financial Innovation* 2.1 (2016), pp. 1–10.

[29] S. Chakraborty, S. Dziembowski, M. Gałązka, T. Lizurej, K. Pietrzak, and M. Yeo. “Trojan-Resilience Without Cryptography”. In: *Theory of Cryptography*. Ed. by K. Nissim and B. Waters. Cham: Springer International Publishing, 2021, pp. 397–428. ISBN: 978-3-030-90453-1.

[30] N. Chandran, V. Goyal, R. Moriarty, and R. Ostrovsky. “Position-Based Cryptography”. In: *SIAM J. Comput.* 43.4 (2014), pp. 1291–1341. DOI: 10.1137/100805005.

[31] E. Chattopadhyay and X. Li. “Non-malleable codes and extractors for small-depth circuits, and affine functions”. In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. Ed. by H. Hatami, P. McKenzie, and V. King. ACM, 2017, pp. 1171–1184. DOI: 10.1145/3055399.3055483.

[32] D. Chaum, C. Crépeau, and I. Damgård. “Multiparty Unconditionally Secure Protocols (Extended Abstract)”. In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. Ed. by J. Simon. ACM, 1988, pp. 11–19. DOI: 10.1145/62212.62214.

[33] J. Chen et al. “GA-based Q-attack on community detection”. In: *IEEE Transactions on Computational Social Systems* 6.3 (2019), pp. 491–503.

[34] L. Chen et al. “A Survey of Adversarial Learning on Graphs”. In: *arXiv* (2020), arXiv–2003.

[35] B. Chor, A. Fiat, M. Naor, and B. Pinkas. “Tracing traitors”. In: *IEEE Trans. Inf. Theory* 46.3 (2000), pp. 893–910. DOI: 10.1109/18.841169.

[36] H. Chung and E. Shi. “Foundations of Transaction Fee Mechanism Design”. In: *CoRR* abs/2111.03151 (2021). arXiv: 2111.03151. URL: https://arxiv.org/abs/2111.03151.

[37] P. Crescenzi, G. D’angelo, L. Severini, and Y. Velaj. “Greedily improving our own closeness centrality in a network”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 11.1 (2016), p. 9.

[38] CryptoKitties. *Collect and breed digital cats!* URL: https://www.cryptokitties.co/.

[39] M. Cygan et al. *Parameterized Algorithms*. 1st. Springer Publishing Company, Incorporated, 2015. ISBN: 3319212745.

[40] M. Cygan et al. *Parametrized Algorithms*. Springer, 2015, 2015.

[41] D. Dachman-Soled and Y. T. Kalai. “Securing Circuits against Constant-Rate Tampering”. In: *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Ed. by R. Safavi-Naini and R. Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 533–551. DOI: 10.1007/978-3-642-32009-5_31.

[42] D. Dachman-Soled and Y. T. Kalai. “Securing Circuits and Protocols against 1/poly(k) Tampering Rate”. In: *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*. Ed. by Y. Lindell. Vol. 8349. Lecture Notes in Computer Science. Springer, 2014, pp. 540–565. DOI: 10.1007/978-3-642-54242-8_23.

[43] D. Dachman-Soled, F. Liu, E. Shi, and H. Zhou. “Locally Decodable and Updatable Non-malleable Codes and Their Applications”. In: *J. Cryptol.* 33.1 (2020), pp. 319–355. DOI: 10.1007/s00145-018-9306-z.

[44] I. Damgård. “Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks”. In: *Advances in Cryptology - CRYPTO ’91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*. Ed. by J. Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Springer, 1991, pp. 445–456. DOI: 10.1007/3-540-46766-1_36. URL: https://doi.org/10.1007/3-540-46766-1_36.

[45] P. Das, S. Faust, and J. Loss. “A Formal Treatment of Deterministic Wallets”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’19. London, United Kingdom: Association for Computing Machinery, 2019, 651–668. ISBN: 9781450367479. DOI: 10.1145/3319535.3354236. URL: https://doi.org/10.1145/3319535.3354236.

[46] J. Davis. *The crypto-currency*. 2011. URL: https://www.newyorker.com/magazine/2011/10/10/the-crypto-currency.

[47] D. P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. ISBN: 978-0-521-88427-3. URL: http://www.cambridge.org/gb/knowledge/isbn/item2327542/.

[48] C. Dwork, M. Naor, and H. Wee. "Pebbling and Proofs of Work". In: *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*. Ed. by V. Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, 2005, pp. 37–54. DOI: 10.1007/11535218_3. URL: https://doi.org/10.1007/11535218_3.

[49] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. *Proofs of Space*. Cryptology ePrint Archive, Paper 2013/796. https://eprint.iacr.org/2013/796. 2013. URL: https://eprint.iacr.org/2013/796.

[50] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. "Proofs of Space". In: *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*. Ed. by R. Gennaro and M. Robshaw. Vol. 9216. Lecture Notes in Computer Science. Springer, 2015, pp. 585–605. DOI: 10.1007/978-3-662-48000-7_29. URL: https://doi.org/10.1007/978-3-662-48000-7_29.

[51] S. Dziembowski, S. Faust, and T. Lizurej. "Individual Cryptography". In: *Advances in Cryptology – CRYPTO 2023* (2023). URL: https://link.springer.com/book/10.1007/978-3-031-38554-4.

[52] S. Dziembowski, S. Faust, and F.-X. Standaert. "Private Circuits III: Hardware Trojan-Resilience via Testing Amplification". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria: Association for Computing Machinery, 2016, 142–153. ISBN: 9781450341394. DOI: 10.1145/2976749.2978419. URL: https://doi.org/10.1145/2976749.2978419.

[53] S. Dziembowski, S. Faust, and F. Standaert. "Private Circuits III: Hardware Trojan-Resilience via Testing Amplification". In: *ACM CCS*. Ed. by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi. ACM, 2016, pp. 142–153. DOI: 10.1145/2976749.2978419.

[54] S. Dziembowski, T. Kazana, and D. Wichs. "One-Time Computable Self-erasing Functions". In: *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*. Ed. by Y. Ishai. Vol. 6597. Lecture Notes in Computer Science. Springer, 2011, pp. 125–143. DOI: 10.1007/978-3-642-19571-6_9. URL: https://doi.org/10.1007/978-3-642-19571-6_9.

[55] S. Dziembowski and K. Pietrzak. "Leakage-Resilient Cryptography". In: *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*. IEEE Computer Society, 2008, pp. 293–302. DOI: 10.1109/FOCS.2008.56.

[56] S. Dziembowski, K. Pietrzak, and D. Wichs. "Non-Malleable Codes". In: *J. ACM* 65.4 (2018), 20:1–20:32. DOI: 10.1145/3178432.

[57] K. Efremenko et al. "Circuits Resilient to Short-Circuit Errors". In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2022. Rome, Italy: Association for Computing Machinery, 2022, 582–594. ISBN: 9781450392648. DOI: 10.1145/3519935.3520007.

[58] S. Eskandari, S. Moosavi, and J. Clark. *SoK: Transparent Dishonesty: front-running attacks on Blockchain*. 2019. arXiv: 1902.05164 [cs.CR].

[59] I. Eyal and E. G. Sirer. *Majority is not Enough: Bitcoin Mining is Vulnerable*. 2013. arXiv: 1311.0243 [cs.CR].

[60] S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum. "Leakage-Resilient Signatures". In: *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*. Ed. by D. Micciancio. Vol. 5978. Lecture Notes in Computer Science. Springer, 2010, pp. 343–360. DOI: 10.1007/978-3-642-11799-2_21.

[61] S. Faust, P. Mukherjee, J. B. Nielsen, and D. Venturi. "A Tamper and Leakage Resilient von Neumann Architecture". In: *Public-Key Cryptography – PKC 2015*. Ed. by J. Katz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 579–603. ISBN: 978-3-662-46447-2.

[62] S. Faust, K. Pietrzak, and D. Venturi. "Tamper-Proof Circuits: How to Trade Leakage for Tamper-Resilience". In: 2011, pp. 391–402. DOI: 10.1007/978-3-642-22006-7_33.

[63] S. Fei, Z. Yan, W. Ding, and H. Xie. "Security vulnerabilities of SGX and countermeasures: A survey". In: *ACM Computing Surveys (CSUR)* 54.6 (2021), pp. 1–36.

[64] V. Fionda and G. Pirro. "Community deception or: How to stop fearing community detection algorithms". In: *IEEE Transactions on Knowledge and Data Engineering* 30.4 (2017), pp. 660–673.

[65] D. Genkin, Y. Ishai, M. M. Prabhakaran, A. Sahai, and E. Tromer. "Circuits Resilient to Additive Attacks with Applications to Secure Computation". In: *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*. STOC '14. New York, New York: Association for Computing Machinery, 2014, 495–504. ISBN: 9781450327107. DOI: 10.1145/2591796.2591861.

[66] M. T. Godziszewski, T. P. Michalak, M. Waniek, T. Rahwan, K. Zhou, and Y. Zhu. "Attacking similarity-based sign prediction". In: *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2021, pp. 1072–1077.

[67] M. T. Godziszewski, Y. Vorobeychik, and T. Michalak. "Adversarial Link Prediction in Spatial Networks". In: *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. 2023, pp. 1817–1825.

[68] O. Goldreich, S. Micali, and A. Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. Ed. by A. V. Aho. ACM, 1987, pp. 218–229. DOI: 10.1145/28395.28420.

[69] S. Goldwasser, S. Micali, and C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM J. Comput.* 18.1 (1989), pp. 186–208. DOI: 10.1137/0218012.

[70] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller. "Cache attacks on Intel SGX". In: *Proceedings of the 10th European Workshop on Systems Security*. 2017, pp. 1–6.

[71] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. *Founding Cryptography on Tamper-Proof Hardware Tokens*. Cryptology ePrint Archive, Paper 2010/153. https://eprint.iacr.org/2010/153. 2010. URL: https://eprint.iacr.org/2010/153.

[72] L. Grassi, C. Rechberger, D. Rotaru, P. Scholl, and N. P. Smart. "MPC-Friendly Symmetric Key Primitives". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi. ACM, 2016, pp. 430–443. DOI: 10.1145/2976749.2978332.

[73] M. Hu. *Post-Quantum Secure Deterministic Wallet: Stateless, Hot/Cold Setting, and More Secure*. Cryptology ePrint Archive, Paper 2023/062. https://eprint.iacr.org/2023/062. 2023. URL: https://eprint.iacr.org/2023/062.

[74] Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. "Private Circuits II: Keeping Secrets in Tamperable Circuits". In: *Advances in Cryptology - EUROCRYPT 2006*. Ed. by S. Vaudenay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 308–327. ISBN: 978-3-540-34547-3.

[75] Y. Ishai, M. Prabhakaran, A. Sahai, and D. A. Wagner. "Private Circuits II: Keeping Secrets in Tamperable Circuits". In: *EUROCRYPT*. Ed. by S. Vaudenay. Vol. 4004. Lecture Notes in Computer Science. Springer, 2006, pp. 308–327. DOI: 10.1007/11761679_19.

[76] Y. Ishai, A. Sahai, and D. Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: *Advances in Cryptology - CRYPTO 2003*. Ed. by D. Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 463–481. ISBN: 978-3-540-45146-4.

[77] Y. Ishai, A. Sahai, and D. Wagner. "Private circuits: Securing hardware against probing attacks". In: *Annual International Cryptology Conference*. Springer. 2003, pp. 463–481.

[78] Y. T. Kalai, A. B. Lewko, and A. Rao. "Formulas Resilient to Short-Circuit Errors". In: *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*. IEEE Computer Society, 2012, pp. 490–499. DOI: 10.1109/FOCS.2012.69.

[79] Y. T. Kalai and L. Reyzin. "A survey of leakage-resilient cryptography". In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. Ed. by O. Goldreich. ACM, 2019, pp. 727–794. DOI: 10.1145/3335741.3335768.

[80] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. ISBN: 9781466570269. URL: https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269.

[81] M. Kelkar, K. Babel, P. Daian, J. Austgen, V. Buterin, and A. Juels. *Complete Knowledge: Preventing Encumbrance of Cryptographic Secrets*. Cryptology ePrint Archive, Paper 2023/044. https://eprint.iacr.org/2023/044. 2023. URL: https://eprint.iacr.org/2023/044.

[82] A. Kiayias and Q. Tang. "How to keep a secret: leakage deterring public-key cryptosystems". In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. Ed. by A. Sadeghi, V. D. Gligor, and M. Yung. ACM, 2013, pp. 943–954. DOI: 10.1145/2508859.2516691.

[83] A. Kiayias and Y. Tselekounis. "Tamper Resilient Circuits: The Adversary at the Gates". In: *Advances in Cryptology - ASIACRYPT 2013*. Ed. by K. Sako and P. Sarkar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 161–180. ISBN: 978-3-642-42045-0.

[84] S. Kumar, F. Spezzano, V. Subrahmanian, and C. Faloutsos. "Edge weight prediction in weighted signed networks". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE. 2016, pp. 221–230.

[85] M. Lepinksi, S. Micali, and a. shelat. "Collusion-Free Protocols". In: *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*. STOC '05. Baltimore, MD, USA: Association for Computing Machinery, 2005, 543–552. ISBN: 1581139608. DOI: 10.1145/1060590.1060671. URL: https://doi.org/10.1145/1060590.1060671.

[86] J. Leskovec, D. Huttenlocher, and J. Kleinberg. "Predicting positive and negative links in online social networks". In: *Proceedings of the 19th international conference on World wide web*. 2010, pp. 641–650.

[87] J. Leskovec, D. Huttenlocher, and J. Kleinberg. "Signed networks in social media". In: *Proceedings of the SIGCHI conference on human factors in computing systems*. 2010, pp. 1361–1370.

[88] X. Li. "Improved non-malleable extractors, non-malleable codes and independent source extractors". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*. Ed. by H. Hatami, P. McKenzie, and V. King. ACM, 2017, pp. 1144–1156. DOI: 10.1145/3055399.3055486.

[89] Lightningnetwork. *Lightning Network*. URL: https://github.com/lightningnetwork/lnd.

[90] T. Lizurej, T. Michalak, and S. Dziembowski. "On Manipulating Weight Predictions in Signed Weighted Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.4 (2023), pp. 5222–5229. DOI: 10.1609/aaai.v37i4.25652. URL: https://ojs.aaai.org/index.php/AAAI/article/view/25652.

[91] E. V. Mangipudi, D. Lu, and A. Kate. "Collusion-Deterrent Threshold Information Escrow". In: *IACR Cryptol. ePrint Arch.* (2021), p. 95. URL: https://eprint.iacr.org/2021/095.

[92] D. Mayzlin, Y. Dover, and J. Chevalier. "Promotional reviews: An empirical investigation of online review manipulation". In: *American Economic Review* 104.8 (2014), pp. 2421–55.

[93] S. Mazumdar, P. Banerjee, A. Sinha, S. Ruj, and B. K. Roy. "Strategic Analysis of Griefing Attack in Lightning Network". In: *IEEE Transactions on Network and Service Management* 20.2 (2023), pp. 1790–1803. DOI: 10.1109/tnsm.2022.3230768. URL: https://doi.org/10.1109%2Ftnsm.2022.3230768.

[94] S. Micali. "ALGORAND: The Efficient and Democratic Ledger". In: *CoRR* abs/1607.01341 (2016). arXiv: 1607.01341. URL: http://arxiv.org/abs/1607.01341.

[95] S. Micali and L. Reyzin. "Physically Observable Cryptography". In: *Theory of Cryptography*. Ed. by M. Naor. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 278–296. ISBN: 978-3-540-24638-1.

[96] M. L. Morgia, A. Mei, A. M. Mongardini, and E. N. Nemmi. *A Game of NFTs: Characterizing NFT Wash Trading in the Ethereum Blockchain*. 2023. arXiv: 2212.01225 [cs.CY].

[97] S. Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: *Cryptography Mailing list at https://metzdowd.com* (Mar. 2009).

[98] R. Pagh and F. F. Rodler. "Cuckoo hashing". In: *J. Algorithms* 51.2 (2004), pp. 122–144. DOI: 10.1016/j.jalgor.2003.12.002.

[99] S. Park, M. Specter, N. Narula, and R. L. Rivest. "Going from bad to worse: from Internet voting to blockchain voting". In: *Journal of Cybersecurity* 7.1 (Feb. 2021), tyaa025. ISSN: 2057-2085. DOI: 10.1093/cybsec/tyaa025. eprint: https://academic.oup.com/cybersecurity/article-pdf/7/1/tyaa025/42533672/tyaa025.pdf. URL: https://doi.org/10.1093/cybsec/tyaa025.

[100] D. Pointcheval and J. Stern. "Security Proofs for Signature Schemes". In: vol. 1070. May 1996, pp. 387–398. ISBN: 978-3-540-61186-8. DOI: 10.1007/3-540-68339-9_33.

[101] I. Puddu, D. Lain, M. Schneider, E. Tretiakova, S. Matetic, and S. Capkun. "TEEvil: Identity Lease via Trusted Execution Environments". In: *CoRR* abs/1903.00449 (2019). arXiv: 1903.00449. URL: http://arxiv.org/abs/1903.00449.

[102] N. Reiff. *Decentralized Autonomous Organization (DAO): Definition, purpose, and example*. URL: https://www.investopedia.com/tech/what-dao/.

[103] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. "Reputation systems". In: *Communications of the ACM* 43.12 (2000), pp. 45–48.

[104] T. Roughgarden. *Transaction Fee Mechanism Design*. 2021. arXiv: 2106.01340 [cs.CR].

[105] C. G. Small. *Functional Equations and How to Solve Them*. New York, USA: Springer, 2007.

[106] H. Tan, W. Hu, and S. Jha. "A remote attestation protocol with Trusted Platform Modules (TPMs) in wireless sensor networks." In: *Security and Communication Networks* 8.13 (2015), pp. 2171–2188.

[107] J. Tang, Y. Chang, C. Aggarwal, and H. Liu. "A survey of signed network mining in social media". In: *ACM Computing Surveys (CSUR)* 49.3 (2016), pp. 1–37.

[108] I. Tsabary, M. Yechieli, A. Manuskin, and I. Eyal. *MAD-HTLC: Because HTLC is Crazy-Cheap to Attack*. 2021. arXiv: 2006.12031 [cs.CR].

[109] H. W. Turnbull. "A Matrix Form of Taylor's Theorem". In: *Proceedings of the Edinburgh Mathematical Society* 2.1 (1930), 33–54. DOI: 10.1017/S0013091500007537.

[110] J. Van Bulck, F. Piessens, and R. Strackx. "SGX-Step: A practical attack framework for precise enclave execution control". In: *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. 2017, pp. 1–6.

[111] R. S. Wahby, M. Howald, S. Garg, A. Shelat, and M. Walfish. "Verifiable ASICs". In: *IEEE SP*. IEEE Computer Society, 2016, pp. 759–778. DOI: 10.1109/SP.2016.51.

[112] M. Waniek, T. P. Michalak, M. J. Wooldridge, and T. Rahwan. "Hiding individuals and communities in a social network". In: *Nature Human Behaviour* 2.2 (2018), p. 139.

[113] M. Waniek, T. P. Michalak, M. J. Wooldridge, and T. Rahwan. "Hiding individuals and communities in a social network". In: *Nature Human Behaviour* 2.2 (2018), pp. 139–147.

[114] M. Waniek, K. Zhou, Y. Vorobeychik, E. Moro, T. P. Michalak, and T. Rahwan. "How to Hide one's Relationships from Link prediction Algorithms". In: *Scientific reports* 9.1 (2019), pp. 1–10.

[115] T. Was, M. Waniek, T. Rahwan, and T. Michalak. "The Manipulability of Centrality Measures-An Axiomatic Approach". In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. Auckland, New Zealand: AAMAS, 2020, pp. 1467–1475.

[116] N. C. Will and C. A. Maziero. "Intel Software Guard Extensions Applications: A Survey". In: *ACM Computing Surveys* (2023).

[117] X. Yin, Z. Liu, G. Yang, G. Chen, and H. Zhu. *Secure Hierarchical Deterministic Wallet Supporting Stealth Address*. Cryptology ePrint Archive, Paper 2022/627. https://eprint.iacr.org/2022/627. 2022. URL: https://eprint.iacr.org/2022/627.

[118] A. L. Young and M. Yung. "Cryptovirology: Extortion-Based Security Threats and Countermeasures". In: *1996 IEEE Symposium on Security and Privacy, May 6-8, 1996, Oakland, CA, USA*. IEEE Computer Society, 1996, pp. 129–140. DOI: 10.1109/SECPRI.1996.502676.