Informatics

Automating Proofs of Game-Theoretic Security Properties of Off-Chain Protocols

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Lea Salome Brugger, BSc
Matrikelnummer 11712616

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. techn. Laura Kovács, Msc
Mitwirkung: Dipl.-Ing. Sophie Rain, BSc

Wien, 30. August 2022

Lea Salome Brugger   Laura Kovács

Informatics

Automating Proofs of Game-Theoretic Security Properties of Off-Chain Protocols

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering & Internet Computing

by

Lea Salome Brugger, BSc
Registration Number 11712616

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. techn. Laura Kovács, Msc
Assistance: Dipl.-Ing. Sophie Rain, BSc

Vienna, 30th August, 2022

Lea Salome Brugger Laura Kovács

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Lea Salome Brugger, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. August 2022

Lea Salome Brugger

Acknowledgements

I would like to thank Laura Kovács and Sophie Rain for their guidance during the work on this thesis and their diligence as advisors. Further, I would like to thank Anja Petković Komel and Michael Rawson for their valuable work and the countless times they enlightened me with ideas to solve problems which seemed unsolvable to me. Lastly, I would like to thank my boyfriend Timo for his constant and everlasting support and encouragement.

This work was partially supported by the ERC CoG ARTIST 101002685 grant.

Kurzfassung

Off-Chain-Transaktionen stellen eine effektive Maßnahme gegen die mit der niedrigen Skalierbarkeit von Blockchain-Technologie verbundenen Probleme dar, welche besonders im Kontext von Kryptowährungen wie Bitcoin von Relevanz sind. Indem nur wenige Transaktionen auf der Blockchain selbst und stattdessen die meisten in einer zweiten Ebene durchgeführt werden, kann die Durchsatzrate an Transaktionen – welche ein essenzieller Parameter in finanziellen Services ist – signifikant verbessert werden. Das Lightning-Netzwerk für Bitcoin ist ein Beispiel für solch ein Off-Chain-Protokoll.

Im Hinblick auf die Sicherheit von Off-Chain-Protokollen haben sich spieltheoretische Ansätze als nützlich erwiesen. Die Modellierung eines Protokolls als strategisches Spiel und dessen Evaluation anhand von etablierten spieltheoretischen Eigenschaften ermöglicht eine umfangreiche und akkurate Sicherheitsanalyse. Nichtsdestotrotz ist die manuelle Erstellung und Analyse solcher Spiele aufgrund der hohen Komplexität von Off-Chain-Protokollen eine aufwendige und fehleranfällige Aufgabe.

Aus diesem Grund präsentieren wir hiermit einen Prototypen für ein Framework, das den Analyseprozess automatisiert und damit die Evaluation von Off-Chain-Protokollen hinsichtlich ihrer Sicherheit erheblich vereinfacht. Zu diesem Zweck generiert unser Framework automatisiert eine Menge an Formeln basierend auf dem gegebenen Input und eine von drei untersuchten spieltheoretischen Eigenschaften und löst das Problem der Erfüllbarkeit dieser Menge, bekannt als Satisfiability Modulo Theories (SMT). Wir evaluieren unseren Prototypen auf Basis von einigen Beispielen, unter anderem einem spieltheoretischen Modell für die Schließung eines Zahlungskanals im Lightning-Netzwerk.

Abstract

Off-chain transactions are an effective approach to mitigating the issues arising as a consequence of the low scalability of blockchain technology, which are especially problematic in cryptocurrencies like Bitcoin. By conducting only few transactions on the blockchain itself and instead handling most of them in a second layer, the overall throughput of transactions – which is a crucial parameter in financial services – can be increased significantly. An example for such an off-chain protocol is the Bitcoin Lightning Network.

When it comes to the security of off-chain protocols, game-theoretic approaches have been proven useful. Modeling a protocol as a game and evaluating it based on established game-theoretic properties allows for an extensive and accurate security analysis. However, creating and analyzing such a game manually is a tedious and error-prone task due to the high complexity of off-chain protocols.

Therefore, we present a prototype for a framework that automates this analysis, thus facilitating the evaluation of off-chain protocols with respect to their security. To that end, we apply satisfiability modulo theories (SMT) solving for each of three established security properties to a set of constraints which is generated based on the input game and the analyzed property. We evaluate our prototype using several example games, one of which models the closing phase in the Lightning Network protocol.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Over the past few years, blockchain technology has significantly gained in popularity: From 2017 to 2019, the global spending on blockchain solutions nearly tripled, rising from 0.95 billion to 2.7 billion USD according to a study conducted by the International Data Corporation (IDC)[1] [1]. They estimated in 2021 that the annual total amount of money spent on blockchain will increase to 19 billion USD by 2024.

By construction, blockchains yield promising benefits in terms of security, transparency and decentralization, allowing for a broad variety of application fields. One of the most prominent use cases of blockchain technology are cryptocurrencies. The most widely used cryptocurrencies are still Bitcoin [2], with a market share of 38.42% end of 2021, and Ethereum [3], having had 18.92%, although other currencies have become more and more relevant over time [4].

Despite the advantages of blockchain, however, there are also major drawbacks. One downside which particularly impacts cryptocurrencies is low performance. For instance, Bitcoin's transaction throughput is severely limited by the maximum block size which was originally set to 1 MB, allowing for a maximum of approximately 4000 transactions per block, and the average block creation time of ten minutes [5]. In consequence, Bitcoin only supports up to seven transactions per second, a number which is dramatically low for a financial service. For comparison, Visa services scale up to over thousands of transactions per second [6].

There are several approaches to solving this scalability issue, one of which is the concept of off-chain transactions. In so-called layer 2 solutions, only a few transactions are conducted on the blockchain. Most of them are handled outside of the blockchain (off-chain) [6]. An example for the adoption of such off-chain transactions is the Bitcoin

[1] `https://www.idc.com`, accessed on August 30, 2022

Lightning Network [7] where participants can open a shared bidirectional channel in which they deposit money. The opening of a channel is published on the base blockchain (which constitutes layer 1). As long as the channel is open, the deposited money can be redistributed among the participants in arbitrarily many Lightning Network transactions (which is referred to as layer 2); these transactions are conducted off-chain. Ultimately, the channel is closed and the latest deposit distribution state is broadcast on-chain. The money does not have to be directly transferred from one party to the other, but can be routed through a network of nodes which are connected via those bidirectional channels. Such routed payments are called multi-hop payments.

In order for off-chain protocols to be applicable in practice, their security must be verified. In this regard, game-theoretical approaches have been proven to be quite useful. The benefits of employing game theory in the context of the security of off-chain protocols are manifold. Most importantly, by modeling a protocol as a game, the strategic interaction between different agents, also called players, can be expressed. Therefore, not only the honest behavior is considered (where all agents act in the intended way), but all potential scenarios, including ones where a set of agents behaves dishonestly [8]. In such a game, each possible outcome yields a certain utility for each player. As a result, one can investigate whether a player or a set of players gains any benefit from deviating from the intended (honest) behavior by comparing the usefulness of the honest behavior to all other (potentially malicious) strategies.

1.2 Problem Statement

The feasibility of game-theoretic models for investigating the security of the underlying protocol partially depends on their completeness, i.e. expression of all possible interactions between players. In consequence, accurate models are likely to be rather large and complex games. For example, there are trillions of possible joint strategies (combinations of individual players' strategies) for the Closing Game proposed by Rain *et al.* [8] which models the closing phase in the Bitcoin Lightning Network [7], even though it is a two-player game, meaning that it models interactions between only two agents. The complexity of models for scenarios where there are more agents involved (such as multi-hop payments in the Lightning Network [7]) and thus, the number of joint strategies for these games, may be even higher in other protocol instances.

As stated before, an accurate security analysis essentially consists of comparing the utility of the honest behavior to all other strategies since the security of a protocol implies that the intended behavior is most rational, i.e. results in the best outcome. Therefore, due to the copious amount of joint strategies to consider, manually analyzing highly complex game-theoretic models is a tedious task.

The aim of this thesis is thus to automate the process of proving the security of an off-chain protocol modeled as a game. By employing automation, it is possible to overcome two main issues arising in the manual analysis of games: Firstly, manual analysis is error-prone due to the human factor that is introduced, especially when considering highly complex games. With automation, this risk of human errors in a security proof can be eliminated. Additionally, automation allows for higher scalability. Since tremendous amount of effort is required to analyze highly complex games manually, automating this process makes the analysis more efficient by reducing this effort drastically.

1.3 Related Work

Using game-theoretic models is an established approach in the context of security and privacy in general. Do *et al.* [9], for example, provide an overview of existing game-theoretic approaches to solving security problems. They classify those approaches and explore their respective advantages and drawbacks.

The application of game theory in the context of blockchain technology is a relatively new research area. In a paper by Liu *et al.* [10], the authors list a variety of game-theoretic approaches to modeling common blockchain issues such as, for example, selfish mining and denial-of-service attacks, and discuss their respective advantages and disadvantages.

With respect to the security of off-chain protocols in particular, Zappalà *et al.* [11] have taken the first step towards establishing game-theoretic models. They model parts of the Lightning Network protocol [7], such as the channel closing phase and routing, and define game-theoretic security properties of such models. Nevertheless, their approach poses a number of limitations. Firstly, their proposed models do not capture dishonest behavior in detail. However, since for an accurate analysis of the security of a protocol it is necessary to compare the utility of the honest behavior to all other strategies, the models proposed in [11] are not sufficient to reason about the security of the protocol. Furthermore, the most crucial limitation is that the utility values of the possible outcomes are constant which is not realistic in practice where, for instance, transaction fees – which can vary – play a significant role [8].

Mazumdar *et al.* [12] have proposed game models for Hashed Timelock Contracts (HTLCs), a type of contract that enables parties to securely exchange money by setting a specific time frame in which the receiving end must accept the payment. During this time frame, the sender's money that should be transferred is "locked", meaning that they cannot access it until the transaction is completed. If the receiver does not accept the payment within the given time frame, the money is not transferred. This method is also in use in the Bitcoin Lightning Network [7]. In their work, Mazumdar *et al.* [12] focus on the griefing attack which was first mentioned by Robinson [13] and analyze their game-theoretic models with respect to this vulnerability. In this type of attack, a malicious node on the receiving end of a payment stalls the transaction by either not responding or responding only shortly before the respective time frame closes. Note that the attacker does not gain any direct benefit from performing the attack; however, the sender's money is locked for a certain amount of time, possibly hindering them from conducting other transactions. In cases where the payment is routed through several nodes in the Lightning Network, this problem potentially affects a considerable number of participants. In contrast to the games presented in [11] and [8], Mazumdar *et al.* [12]

propose games with *incomplete information*, i.e. games where at least one player does not know the outcome for another player, resulting in a probabilistic approach. Nevertheless, as in [11], they analyze the outcome for each player based on fixed (constant) parameters.

In [8], Rain *et al.* overcome the limitations of [11] and [12] by proposing two detailed game-theoretic models with non-fixed utilities for the Bitcoin Lightning Network [7], namely the Closing Game for the closing phase of the protocol and the Routing Game for multi-hop payments. In their work, they define three general properties of joint strategies for games modeling an off-chain protocol which can be used to verify the protocol's security and apply them to these two instances. They formally prove that – under certain assumptions – the Closing Game and thus, the closing phase of the Lightning Network, is secure, whereas the routing module is not. With respect to the latter conclusion, using their model, they are able to detect that the routing module is susceptible to the wormhole attack (which was first presented by Malavolta *et al.* [14]), a vulnerability that the model developed by Zappalà *et al.* [11] does not identify, and the griefing attack which was also analyzed in [12].

Nevertheless, in [8], the constructed games are analyzed manually and the proof that they (do not) fulfill the respective security properties is done by hand. While there exist several automated tools for the analysis of games based on common properties like Nash equilibria, such as, for example, Gambit [15] or PRISM – games [16], they face the limitation that they are only able to handle games with constant, numeric payoffs. As previously elaborated, however, fixed utilities are not feasible when it comes to evaluating the security of off-chain protocols: Since the outcome for the players partially depends on external factors such as variable transaction fees, for a complete security analysis it is necessary to consider all possible values for such parameters. To the best of our knowledge, there exists no automated framework which can reason over games with variable outcomes such that all possible values for the utilities are considered. As elaborated in Section 1.4, taking a first step towards such a framework is one of the contributions of this thesis.

1.4 Contributions

In this thesis, we present an automation framework conducting security analyses on game models for off-chain protocols. The main contributions of our work are twofold: First of all, our framework is designed such that it can process games with constant as well as variable utilities. The support of variable utilities in the input games is what distinguishes our approach from existing game-theoretic frameworks which are limited to games with constant, numeric payoffs.

Furthermore, our prototype automates the process of proving or refuting the security of off-chain protocols, which is a time-consuming and error-prone task when conducted manually. Consequently, it facilitates the analysis of game-theoretic models with respect to their security tremendously. The considerable reduction of effort required to analyze such game-theoretic models is our most important contribution.

1.5 Outline

In the remainder of this thesis, we discuss our approach to an automated framework for the security analysis of off-chain protocols modeled as games in detail, explaining the concepts we use and evaluating our results.

The following chapters are structured as follows. Chapter 2 contains the formal definitions of all game-theoretic concepts as well as the security properties that we use in our security analysis. In Chapter 3, we present a translation of the game-theoretic model and the security properties to satisfiability modulo theories (SMT) formulas which we use to evaluate the security of the protocol in our framework. Chapter 4 provides a more detailed description of the application of the SMT formulas in our framework and how the framework works. In Chapter 5, we evaluate our prototype by applying it to a set of example games and analyzing its limitations. Lastly, in Chapter 6, we summarize our findings.

CHAPTER 2

Preliminaries

Before elaborating on how to use SMT solving to reason over the security of off-chain protocols modeled as games, first we need to understand the game-theoretic background as well as the concrete security properties. We overview relevant game-theoretic concepts by using and adjusting definitions from [8], [17] to our setting.

2.1 Game Theory

A game models interactions between decision makers [17], so-called *players*. Each player plays a certain *strategy*, i.e. chooses from a set of *actions*. The combination of all individual players' strategies is a *joint strategy*. In the end, depending on the actions the players chose over the course of a game, each player gets a certain *utility*.

There are several types of games. One of the simplest ones is the *normal form game (NFG)*. In an NFG, players choose their actions simultaneously.

Definition 2.1.1 (Normal Form Game – NFG). *A normal form game is a tuple* $\Gamma = (N, \mathscr{S}, u)$*, where* N *denotes the finite set of players,* $\mathscr{S}$ *the set of joint strategies and* u *the utility function. A joint strategy* $\sigma \in \mathscr{S}$ *is a tuple of strategies* $\sigma = (\sigma_{p_1}, \dots, \sigma_{p_{|N|}})$*, where* $\sigma_{p_i} \in \mathscr{S}_{p_i}$ *and* $\mathscr{S}_{p_i}$ *denotes the non-empty set of strategies, i.e. actions, that player* p_i *can choose from. The tuple* $u = (u_{p_1}, \dots, u_{p_{|N|}})$ *determines the outcome of the game for each player: Each function* $u_{p_i} : \mathscr{S} \to \mathbb{R}$ *maps a joint strategy* $\sigma \in \mathscr{S}$ *to the resulting utility for player* p_i*.*

NFGs can be represented by a table. For instance, Table 2.1 visualizes one of the most famous NFGs, the Prisoner's Dilemma.

Game 2.1.1 (Prisoner's Dilemma)**.** In this version of the Prisoner's Dilemma [17], the two players A and B are arrested for a crime they committed together. The players are in solitary confinement, meaning that they cannot communicate with each other. Each player has two options: They can either cooperate or defect. If both players cooperate (action c), they each serve two years in prison, denoted by the utility -2 in Table 2.1. In the case where both A and B defect (action d), they both have to serve five years in prison, denoted by utility -5. If one of the players defects and the other does not, the defecting prisoner runs free while the other has to serve ten years, which is denoted as (0, -10) or (-10, 0), respectively, in Table 2.1.

A \ B	c	d
c	(-2, -2)	(-10, 0)
d	(0, -10)	(-5, -5)

Table 2.1: Prisoner's Dilemma

In this thesis, however, we consider *extensive form games (EFGs)* as models for off-chain protocols. In contrast to NFGs, players in an EFG choose their actions sequentially, which introduces the concept of *game histories*, which are sequences of actions.

Definition 2.1.2 (Extensive Form Game – EFG)**.** *An extensive form game is a tuple $\Gamma = (N, \mathscr{H}, P, u)$, where N and u are defined as in NFGs. $\mathscr{H}$ denotes the set of histories, $\mathscr{T} \subseteq \mathscr{H}$ the set of terminal histories and P is the next player function.*

The sets of histories $\mathscr{H}$ and $\mathscr{T}$, respectively, satisfy the following properties:

1) $\emptyset \in \mathscr{H}$, with $\emptyset$ denoting the empty history ().

2) If the history $(a_k)_{k=1}^K \in \mathscr{H}$ and $L < K$, then $(a_k)_{k=1}^L \in \mathscr{H}$.

3) A history $(a_k)_{k=1}^K$ is terminal, i.e. $(a_k)_{k=1}^K \in \mathscr{T}$, if there is no action a_{K+1} with $(a_k)_{k=1}^{K+1} \in \mathscr{H}$.

The next player function P assigns to every non-terminal history $(a_k)_{k=1}^K \in \mathscr{H} \setminus \mathscr{T}$ the player $p \in N$ whose turn it is to choose an action next, i.e. $P((a_k)_{k=1}^K) = p$. Thus, after a non-terminal history $h = (a_k)_{k=1}^K \in \mathscr{H} \setminus \mathscr{T}$, it is player $P(h)$'s turn to pick an action from the action set $A(h) := \{a : (h, a) \in \mathscr{H}\}$.

A strategy of player p is a function σ_p mapping every non-terminal history $h \in \mathscr{H} \setminus \mathscr{T}$ with $P(h) = p$ to an action from the action set $A(h)$. As in NFGs, a joint strategy is a tuple of the individual players' strategies. The utilities of all joint strategies with a terminal history $\beta \in \mathscr{T}$ are the same.

In order to make the relation between joint strategies and (terminal) histories more clear, Rain *et al.* [8] have introduced the concept of *extended strategies*.

Definition 2.1.3 (Extended Strategy)**.** *Let β be a terminal history in an EFG Γ, i.e. $\beta \in \mathscr{J}$. Then, all strategies σ_β that result in history β are extended strategies of β.*

EFGs can be visualized as a tree (see Definition 2.1.4). For example, Figure 2.1 depicts a representation of the Market Entry Game, as introduced below.

Definition 2.1.4 (Tree Representation of EFGs)**.** *An EFG $\Gamma = (N, \mathscr{H}, P, u)$ can be represented by the tree $G = (V, E)$ with the following constraints:*

1) *For every history $h \in \mathscr{H}$, there exists exactly one node $v_h \in V$. If h is not terminal, i.e. $h \notin \mathscr{J}$, v_h is labeled $P(h)$, i.e. the next player as assigned by P. Otherwise, if $h \in \mathscr{J}$, v_h is labeled $(u_{p_1}(\sigma_h), \ldots, u_{p_{|N|}}(\sigma_h))$, the joint utility of playing a joint strategy σ_h that yields history h.*

2) *Two nodes $v_h, v_{h'} \in V$ are connected via a directed edge $(v_h, v_{h'}) \in E$ if and only if there exists an action a such that $h' = (h, a)$. This edge $(v_h, v_{h'})$ is labeled a.*

The graph G is a tree with root $v_\emptyset$ representing the empty history, which is labeled with the player whose turn it is first, and with one leaf node v_β for each terminal history $\beta \in \mathscr{J}$.

Game 2.1.2 (Market Entry Game)**.** This game is an adapted version of the Chain-Store Game from [17]. The two players are the companies T and E, where company T thinks about entering a market and E is already established in that market. Hence, T has two options: Firstly, T could stay out of the market (action o). In this case, they get zero profit and company E gets all the profit, which is denoted by $(0, 2p)$ in Figure 2.1. The second option for T is to enter the market (action e). In consequence, company E has to react: They either ignore their new competitor T – in this case, the two companies share the profit, i.e. they both get the utility p – or start a price war, which causes both companies to lose money, resulting in a negative utility, denoted as $(-a, -a)$ in Figure 2.1.

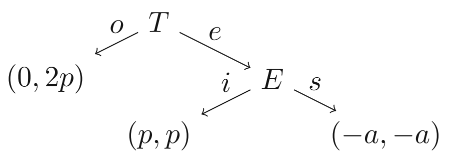

Figure 2.1: Market Entry Game with $a, p > 0$

Furthermore, EFGs can be divided into several *subgames*.

Definition 2.1.5 (Subgame of EFG)**.** *The subgame of an EFG $\Gamma = (N, \mathscr{H}, P, u)$ that follows history $h \in \mathscr{H}$ is the EFG $\Gamma_{|h} = (N, \mathscr{H}_{|h}, P_{|h}, u_{|h})$ such that for every history $h' \in \mathscr{H}_{|h}$, it holds that $(h, h') \in \mathscr{H}$, $P_{|h}(h) = P((h, h'))$ and $u_{|h} = (u_{|h_{p_1}}, \ldots, u_{|h_{p_{|N|}}})$, where $u_{|h_p}(\sigma_{h'}) = u_p(\sigma_{(h,h')})$ for each player p. The joint strategies $\sigma_{h'}$ and $\sigma_{(h,h')}$, respectively, denote arbitrary strategies extending history h' and (h, h'), respectively.*

Note that as all extended joint strategies of some history result in the same utility, the value of $u_p(\sigma_{(h,h')})$ is the same for each $\sigma_{(h,h')}$ and hence, independent of the specific strategy $\sigma_{(h,h')}$.

In order to find the optimal outcome of a game, it can be helpful to use the concept of *Nash equilibria*. Note that we use the notation $t[y/x]$ to denote that in a tuple t, x is replaced by y.

Definition 2.1.6 (Nash Equilibrium). *A Nash equilibrium of an NFG or EFG Γ is a joint strategy $\sigma \in \mathscr{S}$ such that no player can increase their utility by unilaterally deviating from σ. Hence,*

$$\forall p \in N \quad \forall \sigma' \in \mathscr{S}: \quad u_p(\sigma) \geq u_p(\sigma[\sigma'_p/\sigma_p]).$$

For instance, the strategy (d, d) is a Nash equilibrium for the Prisoner's Dilemma, since unilaterally deviating would yield a worse utility for each player (that is, -10 instead of -5).

For EFGs in particular, it is possible to use the concept of Nash equilibria and strengthen it by applying it to all subgames. This is known as the *subgame perfect equilibrium*.

Definition 2.1.7 (Subgame Perfect Equilibrium). *A subgame perfect equilibrium for the EFG $\Gamma = (N, \mathscr{H}, P, u)$ is a joint strategy $\sigma \in \mathscr{S}$ such that $\sigma_{|h}$ is a Nash equilibrium of the subgame $\Gamma_{|h}$ for every $h \in \mathscr{H}$. Formally,*

$$\forall h \in \mathscr{H} \quad \forall p \in N \quad \forall \sigma'_{|h_p} \in \mathscr{S}_{|h_p}: \quad u_{|h_p}(\sigma_{|h}) \geq u_{|h_p}(\sigma_{|h}[\sigma'_{|h_p}/\sigma_{|h_p}]).$$

Here, $\mathscr{S}_{|h_p}$ is the set of individual strategies for player p in the subgame $\Gamma_{|h}$ following history h, i.e. a set of functions mapping non-terminal histories to actions. That is,

$$\mathscr{S}_{|h_p} := \{\sigma_{|h_p} : \quad \sigma_{|h_p} : \mathscr{H}_{|h_p} \to \mathscr{A}_p \text{ such that } \forall h' \in \mathscr{H}_{|h_p} : \sigma_{|h_p}(h') \in A((h, h'))\},$$

where $\mathscr{H}_{|h_p}$ is the set of non-terminal histories in the subgame $\Gamma_{|h}$ after which it is player p's turn to choose an action, i.e.

$$\mathscr{H}_{|h_p} := \{h' : \quad h' \in \mathscr{H}_{|h} \setminus \mathscr{J}_{|h} \text{ and } P_{|h}(h') = p\},$$

and $\mathscr{A}_p$ is defined as the set of all actions that player p can choose from at some point in the game Γ.

2.2 Security Properties

In the context of off-chain protocols modeled as games, we define the terms *honest player* and *honest behavior*, respectively. An honest player, i.e. a player that behaves honestly, is a player that behaves as intended by the protocol. On the other hand, a dishonest player tries to cheat in some way.

Using this terminology, we can introduce the concept of *honest histories* of game-theoretic models of off-chain protocols.

Definition 2.2.1 (Honest History)**.** *An honest history of an EFG* $\Gamma = (N, \mathscr{H}, P, u)$ *modeling an off-chain protocol is a terminal history* $\beta_o \in \mathscr{J}$ *that results from all players behaving honestly.*

In [8], Rain *et al.* have established three game-theoretic security properties for off-chain protocols modeled as games. The first one is *weak immunity.*

Definition 2.2.2 (Weak Immunity – WI)**.** *A joint strategy* $\sigma \in \mathscr{S}$ *for an NFG or EFG* Γ *is weak immune if no player* p *that follows strategy* σ *can get a negative utility, regardless of how the other players behave. Thus,*

$$\forall p \in N \quad \forall \sigma' \in \mathscr{S} : \quad u_p(\sigma'[\sigma_p/\sigma'_p]) \geq 0.$$

Weak immunity ensures that behaving honestly never causes the players to lose resources. In order to guarantee that there is no incentive in forming a group of players and jointly deviating from the honest behavior, Rain *et al.* [8] have defined *collusion resilience* as the second security property.

Definition 2.2.3 (Collusion Resilience – CR)**.** *A joint strategy* $\sigma \in \mathscr{S}$ *for an NFG or EFG* Γ *is collusion resilient if no strict subgroup of players* $S := \{s_1, \ldots, s_j\}$ *with* $j < |N|$ *benefits from deviating from strategy* σ*. Formally,*

$$\forall S \subset N \quad \forall \sigma'_{s_i} \in \mathscr{S}_{s_i} : \quad \sum_{p \in S} u_p(\sigma) \geq \sum_{p \in S} u_p(\sigma[\sigma'_{s_1}/\sigma_{s_1}, \ldots, \sigma'_{s_j}/\sigma_{s_j}]).$$

Lastly, the third security property defined by Rain *et al.* [8] is *practicality.*

Definition 2.2.4 (Practicality – P)**.** *A joint strategy* $\sigma \in \mathscr{S}$ *for an EFG* Γ *is practical if it is a subgame perfect equilibrium of* Γ.

Note that this definition only applies to EFGs, as opposed to the other two security properties. For a definition of practicality in the context of NFGs, the reader is referred to [8].

Similarly to the definition of the properties on joint strategies (Definition 2.2.2, 2.2.3 and 2.2.4), a terminal history of an EFG is said to be weak immune, collusion resilient or practical, respectively, if there exists a weak immune, collusion resilient or practical joint strategy, respectively, extending it. Given the definition of the three security properties, the concept of *secure strategies* and *secure histories*, respectively, as proposed by Rain *et al.* [8], can be introduced.

Definition 2.2.5 (Secure Strategy)**.** *A joint strategy $\sigma \in \mathscr{S}$ of an NFG or EFG is secure if it is weak immune, practical and collusion resilient.*

Definition 2.2.6 (Secure History)**.** *A terminal history β of an EFG is secure if there exist extended strategies σ_1, σ_2, and σ_3 of β such that σ_1 is weak immune, σ_2 is collusion resilient and σ_3 is practical.*

Note that in Definition 2.2.6, it is possible (but not necessary) that $\sigma_1 = \sigma_2 = \sigma_3$.

Using Definition 2.2.6, we say that a protocol modeled as an EFG together with an honest history β_o is secure if β_o is secure.

Example 2.2.1 (Security of Market Entry Game with History (o))**.** Consider the Market Entry Game depicted in Figure 2.1 and assume that the only honest history is (o). The Market Entry Game together with history (o) is secure if there is an extended strategy of (o) that is weak immune, another that is collusion resilient and a third one which is practical.

Starting with weak immunity, consider the joint strategy where company T takes action o and company E picks action i, denoted by $[\emptyset \to o, (e) \to i]$. $[\emptyset \to o, (e) \to i]$ is an extended strategy of history (o). It is also weak immune since no player can get a negative utility when following the joint strategy, even if the other deviates: If E deviates, company T still gets utility 0. Similarly, if T deviates, E gets utility p, which is fixed to be greater than 0.

Regarding collusion resilience, we are considering strict subgroups of players. Since the Market Entry Game is a two-player-game, the only strict subgroups are the single players. Collusion resilience implies that no strict subgroups of players can benefit from deviating from the joint strategy. Consider the joint strategy where player T takes action o and E action s, denoted by $[\emptyset \to o, (e) \to s]$. Once again, $[\emptyset \to o, (e) \to s]$ is an extended strategy of history (o) and it is also collusion resilient: If player T deviates, i.e. takes action e instead of o, they would get the utility $-a$ which is strictly worse since $-a < 0$. For E, deviating does not make a difference, as they still get the same utility $(2p)$ even if they choose action i over action s.

When it comes to practicality, we also have to consider each subgame of the original game. There are two subgames in the Market Entry Game, namely the entire game and the subgame where only player E gets to choose an action. A history is practical if there exists an extended strategy of this history that is practical. As we have already established, there are two extended joint strategies of the history (o), namely $[\emptyset \to o, (e) \to i]$ and $[\emptyset \to o, (e) \to s]$, which are also the only extended strategies of (o). However, neither of them is practical. Consider the joint strategy $[\emptyset \to o, (e) \to i]$ first and the subgame which is the entire game. In this case, T could improve their utility by deviating, i.e. choosing action e over action o since $p > 0$. When it comes to the joint strategy $[\emptyset \to o, (e) \to s]$, this is not the case since T would get utility $-a$ by deviating and $-a < 0$. However, in the second subgame which starts with E choosing an action, player E can actually get a better outcome by deviating from action s to action i since $p > -a$. As there exists no extended strategy of history (o) that is practical, the history is not practical.

In conclusion, the Market Entry Game together with history (o) is weak immune and collusion resilient, but not practical, and thus, not secure.

CHAPTER 3

Game-Based Security as SMT Formulas

We assume familiarity with SMT-based logics and SMT solving and refer to [18], [19] for details. In our framework, we use SMT solving to automate the security analysis of off-chain protocols modeled as EFGs based on the three properties discussed in Section 2.2. To this end, we construct formulas from which it is possible to extract an extended joint strategy of an honest history for the game modeling a specific protocol. Hence, the game as well as the security properties have to be encoded as SMT constraints.

3.1 Joint Strategies and Honest Histories

The core idea of our SMT encoding is that if the set of resulting constraints is satisfiable, we are able to extract a joint strategy from a model which fulfills the checked security property and results in the verified honest history. We introduce so-called *action variables*, which are Boolean variables that essentially correspond to the branches of the game tree representing the EFG. In order to achieve the extraction of a joint strategy from a model for the complete set of formulas, we add constraints that ensure that at every internal node of the game tree (i.e. every point in the game where it is some player's turn to choose an action), exactly one action variable representing an outgoing edge of this node is set to `true`. Thus, we construct the following constraints from an EFG $\Gamma = (N, \mathscr{H}, P, u)$:

$$\bigwedge_{h \in \mathscr{H} \setminus \mathscr{T}} (\underbrace{(\bigvee_{a \in A(h)} v_a^h)}_{(1)} \wedge \underbrace{\bigwedge_{(a_i,a_j) \in \{(a_i,a_j) : \{a_i,a_j\} \subseteq A(h) \wedge a_i \neq a_j\}} (\neg v_{a_i}^h \vee \neg v_{a_j}^h)}_{(2)}), \tag{3.1}$$

where v_a^h is an action variable representing that for a non-terminal history h with $P(h) = p$, $\sigma_p(h) = a$, i.e. the player p whose turn it is takes action a in the resulting joint strategy. Constraint (1) in this formula ensures that at least one of the action variables resulting from a non-terminal history is set to `true` in the resulting model, whereas subformula (2) further strengthens the constraint by making sure that at most one of those action variables can be `true`. In the end, a joint strategy σ can be obtained from the model for the complete set of formulas: For each non-terminal history h, we map h to the action a such that in the model, v_a^h is `true`, in the individual strategy of the player p whose turn it is at the point in the game with history h (i.e. $P(h) = p$), σ_p. Formally,

$$\forall h \in \mathscr{H} \setminus \mathscr{T} : \quad P(h) = p \Rightarrow (v_a^h \Leftrightarrow \sigma_p(h) = a),$$

where the joint strategy σ is the tuple of all individual strategies σ_p with $p \in N$.

In order to verify the security of an off-chain protocol modeled as an EFG, we need to find joint strategies which extend the honest histories. Therefore, we have to ensure that an extracted joint strategy yields the currently checked honest history. We achieve this by adding a constraint which guarantees that the action variables corresponding to the actions constituting the honest history are set to `true` in a model for the complete set of formulas. Thus, for an honest history $\beta_o = (a_1, \dots, a_n)$, we get the constraint

$$v_{a_1}^{\emptyset} \land \dots \land v_{a_n}^{(a_1,\dots,a_{n-1})}. \tag{3.2}$$

Example 3.1.1 (Joint Strategy and Honest History Constraints for Market Entry Game)**.** Consider once again the Market Entry Game with the honest history (o) from Example 2.2.1. Combining the two constraints (3.1) and (3.2) and applying them to the game and the honest history yields the following formula:

$$\underbrace{(v_o^{\emptyset} \lor v_e^{\emptyset}) \land (\neg v_o^{\emptyset} \lor \neg v_e^{\emptyset}) \land (v_i^{(e)} \lor v_s^{(e)}) \land (\neg v_i^{(e)} \lor \neg v_s^{(e)})}_{(1)} \land \underbrace{v_o^{\emptyset}}_{(2)}. \tag{3.3}$$

In this formula, subformula (1) is obtained by applying (3.1) and (2) by applying (3.2).

3.2 Security Properties

In Section 3.1, we discuss how we encode extended joint strategies of honest histories in general. In order to check one of the security properties, however, we need to add constraints that guarantee that a joint strategy resulting from a model satisfies the respective property. The following subsections each cover one of the three security properties established in Section 2.2.

3.2.1 Weak Immunity

For joint strategies fulfilling weak immunity as per Definition 2.2.2, we have to make sure that the utility for each player following the strategy is non-negative, no matter how the other players behave. In order to encode this as an SMT formula, we "fix" one player at a time and once again use action variables to model the behavior of the fixed player. To this end, we add an implication for each terminal history which states that if the fixed player takes the actions that are included in the history, the resulting utility for the player is greater than or equal to 0. Thus, when obtaining a joint strategy from a model for the set of formulas resulting from combining (3.1) and (3.2) with the constraint for weak immunity (3.4), the latter ensures that each player's utility is positive given that they follow the joint strategy. Formally, the weak immunity constraint for a game $\Gamma = (N, \mathscr{H}, P, u)$ can be written as follows:

$$\bigwedge_{p \in N} \bigwedge_{\beta \in \mathscr{J}} \Big(\big(\bigwedge_{h \in \mathscr{H}^p_{a \to \beta}} v^h_a \big) \Rightarrow u^{\mathscr{J}}_p(\beta) \geq 0 \Big), \tag{3.4}$$

where $\mathscr{H}^p_{a \to \beta}$ is the set of histories leading up to the terminal history β where it is player p's turn to choose an action next. That is, $\mathscr{H}^p_{a \to \beta}$ includes all non-terminal histories h where $P(h) = p$ and appending some action a and another (possibly empty) sequence of actions h' to h yields the terminal history β. Formally,

$$\mathscr{H}^p_{a \to \beta} := \{h : \quad h \in \mathscr{H} \setminus \mathscr{J} \land P(h) = p \land \exists h' \in \mathscr{H}_{|(h,a)} : \beta = (h, a, h')\}.$$

The function $u^{\mathscr{J}}_p$ is an adapted utility function which maps terminal histories to the corresponding utility for player p:

$$u^{\mathscr{J}}_p : \mathscr{J} \to \mathbb{R} \text{ such that } \forall \beta \in \mathscr{J} : u^{\mathscr{J}}_p(\beta) = u_p(\sigma_\beta),$$

where σ_β is an arbitrary extended joint strategy of the terminal history β. Since all joint histories that result in the same terminal history yield the same utility, the value of $u_p(\sigma_\beta)$ is independent of the specific strategy σ_β.

Example 3.2.1 (Weak Immunity Constraints for Market Entry Game)**.** In order to find a weak immune joint strategy for the Market Entry Game extending the honest history (o), we would add the following formula to the constraints from Example 3.1.1, given in (3.3):

$$\begin{aligned} &\underbrace{(v^\emptyset_o \Rightarrow 0 \geq 0) \land (v^\emptyset_e \Rightarrow p \geq 0) \land (v^\emptyset_e \Rightarrow -a \geq 0)}_{(1)} \land \\ &\underbrace{2p \geq 0 \land (v^{(e)}_i \Rightarrow p \geq 0) \land (v^{(e)}_s \Rightarrow -a \geq 0)}_{(2)}. \end{aligned} \tag{3.5}$$

Here, the constraints in subformula (1) are the constraints added for player T, while subformula (2) consists of the constraints for the utility for player E. Note that the first constraint in (2), $2p \geq 0$, does not contain an implication because it is the constraint for the utility for E given the terminal history (o). In this history, player E does not get to make a choice. Thus, the set $\mathscr{H}^{E}_{a\to(o)}$ is empty, causing the left-hand side of the implication in (3.4) to be an empty conjunction, which is equivalent to true. Consequently, the left-hand side can be omitted.

By setting the action variables $v_o^{\emptyset}$ and $v_i^{(e)}$ to true and $v_e^{\emptyset}$ and $v_s^{(e)}$ to false in this set of formulas, which corresponds to the weak immune joint strategy for history (o) from Example 2.2.1, one can satisfy the constraints for all possible values $a, p > 0$.

3.2.2 Collusion Resilience

The definition of collusion resilience (see Definition 2.2.3) implies that no strict subgroup of players can benefit from deviating from the honest behavior. Since we are dealing with groups of players now, in contrast to weak immunity, for this property we have to consider the sums of utilities for a certain group and compare those to the sum of utilities for this group resulting from the considered honest history. Similarly to encoding weak immunity, we fix a subgroup of players. However, since we check if there is an incentive in deviating from a joint strategy, we do not model the behavior of the fixed players in this case, but the behavior of the players which are not fixed. Therefore, for an honest history β_o, we add an implication for each terminal history saying that if the non-fixed players take the actions included in the terminal history, the sum of utilities for the fixed players resulting from playing a joint strategy extending β_o is greater than or equal to the sum of utilities for this checked terminal history. Given an EFG $\Gamma = (N, \mathscr{H}, P, u)$, the resulting formula reads as follows:

$$\bigwedge_{S \subset N} \bigwedge_{\beta \in \mathscr{J}} \Big(\Big(\bigwedge_{h \in \mathscr{H}^{N\setminus S}_{a\to\beta}} v_a^h \Big) \Rightarrow \sum_{p\in S} u_p^{\mathscr{J}}(\beta_o) \geq \sum_{p \in S} u_p^{\mathscr{J}}(\beta) \Big), \tag{3.6}$$

where $u_p^{\mathscr{J}}$ is defined as in (3.4) and $\mathscr{H}^{N\setminus S}_{a\to\beta}$ denotes the set of histories leading up to the terminal history β where one of the players which are not in the subgroup S picks an action next. It is therefore the set of non-terminal histories h such that $P(h) \in N \setminus S$ and appending some action a and another (possibly empty) sequence of actions h' results in the terminal history β. Thus,

$$\mathscr{H}^{N\setminus S}_{a\to\beta} \coloneqq \{h : \quad h \in \mathscr{H} \setminus \mathscr{J} \wedge P(h) \in N \setminus S \wedge \exists h' \in \mathscr{H}_{|(h,a)} : \beta = (h, a, h')\}.$$

Example 3.2.2 (Collusion Resilience Constraints for Market Entry Game)**.** Now we want to find a collusion resilient joint strategy for the Market Entry Game which extends the honest history (o). To achieve this, we add the following constraints to the formula from Example 3.1.1, provided in (3.3):

$$\begin{aligned} &\underbrace{0 \geq 0 \wedge (v_i^{(e)} \Rightarrow 0 \geq p) \wedge (v_s^{(e)} \Rightarrow 0 \geq -a)}_{(1)} \wedge \\ &\underbrace{(v_o^{\emptyset} \Rightarrow 2p \geq 2p) \wedge (v_e^{\emptyset} \Rightarrow 2p \geq p) \wedge (v_e^{\emptyset} \Rightarrow 2p \geq -a)}_{(2)} . \end{aligned} \tag{3.7}$$

In formula (3.7), subformula (1) contains all constraints for the subgroup $\{T\}$, whereas subformula (2) includes all constraints for the subgroup $\{E\}$. Since the Market Entry Game is a two-player game, these two sets consisting of one single player are the only strict subgroups of players.

With the action variables $v_o^{\emptyset}$ and $v_s^{(e)}$ set to `true` and $v_e^{\emptyset}$ and $v_i^{(e)}$ to `false`, modeling the collusion resilient joint strategy for history (o) from Example 2.2.1, the resulting set of formulas is satisfied for all possible values $a, p > 0$.

3.2.3 Practicality

For the practicality property provided in Definition 2.2.4 to be fulfilled, in each subgame of the EFG, it is not possible for a single player to gain an advantage from deviating from a joint strategy extending the checked honest history. In order to satisfy this condition, we use so-called *subgame utility variables*, denoted as y_p^h, which are (numeric) variables representing the utility for a player p in a subgame following history h. For a given subgame with history h of a game $\Gamma = (N, \mathscr{H}, P, u)$, we construct the following formula to model the subgame utilities of the players:

$$\bigwedge_{\beta=(a_1,\dots,a_n)\in\mathscr{I}_{|h}} \bigwedge_{p\in N} (v_{a_1}^{(h)} \wedge \dots \wedge v_{a_n}^{(h,\dots,a_{n-1})} \Rightarrow y_p^h = u_{|h_p}^{\mathscr{I}}(\beta)), \tag{3.8}$$

with the utility functions $u_{|h_p}^{\mathscr{I}}$ being defined analogously to $u_p^{\mathscr{I}}$ from before, but for the subgame following history h. We have one implication per player p for each terminal history β of the subgame which says that if the players follow the sequence of actions constituting the terminal history β (denoted by the conjunction of the respective action variables), then p's resulting utility in this subgame is given by the value of $u_{|h_p}^{\mathscr{I}}(\beta)$. In other words, if the action variables representing the actions in a history β are all set to `true`, then the utility for a player p in the subgame starting from history h is required to be equal to $u_{|h_p}^{\mathscr{I}}(\beta)$.

We use (3.8) to find the maximum utility for each player in each subgame by constructing an implication:

$$\bigwedge_{h\in\mathscr{H}\setminus\mathscr{T}} \Big(\Big(\bigwedge_{\beta=(a_1,\ldots,a_n)\in\mathscr{T}_{|h}} \bigwedge_{p\in N} (v_{a_1}^{(h)} \wedge \cdots \wedge v_{a_n}^{(h,\ldots,a_{n-1})} \Rightarrow y_p^h = u_{|h_p}^{\mathscr{T}}(\beta))\Big) \Longrightarrow \Big(\bigwedge_{p\in N}\bigwedge_{\beta\in\mathscr{T}_{|h}} ((\bigwedge_{h'\in\mathscr{H}^{N\setminus\{p\}}_{|h_{a\to\beta}}} v_a^{(h,h')}) \Rightarrow y_p^h \geq u^{\mathscr{T}}_{|h_p}(\beta))\Big)\Big). \tag{3.9}$$

Here, $\mathscr{H}^{N\setminus\{p\}}_{|h_{a\to\beta}}$ is the set of histories leading up to the terminal history β in the subgame starting at history h where one of the players that is not player p chooses an action next. It is therefore the set of non-terminal histories h' such that $P_{|h}(h')\in N\setminus\{p\}$ and appending some action a and another (possibly empty) sequence of actions h'' results in the terminal history β. Therefore,

$$\mathscr{H}^{N\setminus\{p\}}_{|h_{a\to\beta}} := \{h' : \quad h'\in\mathscr{H}_{|h}\setminus\mathscr{T}_{|h} \wedge P_{|h}(h')\in N\setminus\{p\}\wedge \exists h''\in\mathscr{H}_{|(h,h',a)} : \beta=(h',a,h'')\}.$$

Through the comparison of the subgame utility variables to the actual utility values in the right-hand side of the implication, constraint (3.9) (together with the formulas for joint strategies and honest histories, (3.1) and (3.2)) guarantees that in a model for the resulting set of formulas, the players' utility has the highest possible value in each subgame. Thus, there is no incentive in deviating, making the resulting joint strategy practical.

Example 3.2.3 (Practicality Constraints for Market Entry Game)**.** In this example, the goal is to find a practical joint strategy for the Market Entry Game which extends the honest history (o). Hence, we construct the following implication according to constraint (3.9) for the subgame starting from the empty history, i.e. the entire game:

$$\begin{aligned}
&(\underbrace{(v_o^\emptyset \Rightarrow y_T^\emptyset = 0)\wedge(v_o^\emptyset \Rightarrow y_E^\emptyset = 2p)}_{(1.1)}\wedge\\
&\underbrace{(v_e^\emptyset \wedge v_i^{(e)} \Rightarrow y_T^\emptyset = p)\wedge(v_e^\emptyset \wedge v_i^{(e)} \Rightarrow y_E^\emptyset = p)}_{(1.2)}\wedge\\
&\underbrace{(v_e^\emptyset \wedge v_s^{(e)} \Rightarrow y_T^\emptyset = -a)\wedge(v_e^\emptyset \wedge v_s^{(e)} \Rightarrow y_E^\emptyset = -a)})_{(1.3)} \quad \Longrightarrow\\
&\quad(\underbrace{y_T^\emptyset \geq 0 \wedge (v_i^{(e)} \Rightarrow y_T^\emptyset \geq p)\wedge(v_s^{(e)} \Rightarrow y_T^\emptyset \geq -a)}_{(2.1)}\wedge\\
&\quad\underbrace{(v_o^\emptyset \Rightarrow y_E^\emptyset \geq 2p)\wedge(v_e^\emptyset \Rightarrow y_E^\emptyset \geq p)\wedge(v_e^\emptyset \Rightarrow y_E^\emptyset \geq -a)}_{(2.2)}),
\end{aligned} \tag{3.10}$$

where the subformulas (1.1), (1.2) and (1.3) are obtained by applying (3.8) for the terminal histories (o), (e, i) and (e, s), respectively, and the constraints in (2.1) and (2.2) represent the comparisons of the utility for player T and E, respectively, in the subgame with the empty history, as defined in the right-hand side of the implication in (3.9).

Additionally, for the subgame starting from history (e), we obtain the following formula:

$$\begin{aligned}
&(\underbrace{(v_i^{(e)} \Rightarrow y_T^{(e)} = p) \wedge (v_i^{(e)} \Rightarrow y_E^{(e)} = p)}_{(1.1)} \wedge \\
&\underbrace{(v_s^{(e)} \Rightarrow y_T^{(e)} = -a) \wedge (v_s^{(e)} \Rightarrow y_E^{(e)} = -a)}_{(1.2)}) \quad \Longrightarrow \\
&\quad (\underbrace{(v_i^{(e)} \Rightarrow y_T^{(e)} \geq p) \wedge (v_s^{(e)} \Rightarrow y_T^{(e)} \geq -a)}_{(2.1)} \wedge \\
&\quad \underbrace{y_E^{(e)} \geq p \wedge y_E^{(e)} \geq -a}_{(2.2)}).
\end{aligned} \tag{3.11}$$

Similarly to formula (3.10), here, subformula (1.1) and (1.2) consist of the constraints obtained from (3.8) for the terminal histories (i) and (s), respectively, of the subgame starting from history (e). The constraints in (2.1) and (2.2) are the comparisons of the subgame utility for player T and E, respectively.

Together with formula (3.3) from Example 3.1.1, the conjunction of the constraints for the two subgames is not satisfiable for each possible value of the constants a and p and the subgame utility variables: Since $v_o^{\emptyset}$ needs to be set to `true` due to (3.3), $y_T^{\emptyset}$ is equal to 0 according to constraint (1.1) in (3.10). However, as either $v_i^{(e)}$ or $v_s^{(e)}$ needs to be set to `true` as well because of (3.3), there is a contradiction: In the case where $v_i^{(e)}$ is `true` and $v_s^{(e)}$ is not, $y_T^{\emptyset}$ must be greater than or equal to p ($y_T^{\emptyset} \geq p$) according to the subformula (2.1) in (3.10). However, this is not possible because $y_T^{\emptyset} = 0$ and $p > 0$. If, on the other hand, $v_s^{(e)}$ is `true` and $v_i^{(e)}$ is `false`, constraint (3.11) is not fulfilled since according to subformula (1.2), $y_E^{(e)} = -a$, but $y_E^{(e)}$ is supposed to be greater than or equal to p ($y_E^{(e)} \geq p$) due to subformula (2.2). As $a, p > 0$, this is a contradiction. Consequently, there is no practical joint strategy for the honest history (o).

CHAPTER 4

Game-Theoretic SMT Reasoning

In this chapter, we discuss the concepts and procedures we use in our automated framework to establish the security of an off-chain protocol. First, we describe what a problem instance for our framework looks like. Then, we elaborate on how we apply the findings from Chapter 3 to such an instance and what other aspects we have to consider.

4.1 Input

An input instance for our framework is a tuple $\Pi = (\Gamma, \mathscr{O}, C, C_{\mathrm{WI}}, C_{\mathrm{CR}}, C_{\mathrm{P}})$. Γ is an EFG modeling the checked off-chain protocol and $\mathscr{O} \subseteq \mathscr{I}$ is a set of honest histories of said protocol. In some games, like the Market Entry Game or the Closing Game defined by Rain *et al.* [8], the utilities of the players might not have fixed (numeric) values, but consist of terms of variables whose value or ordering may be constrained in some way. For example, in the Market Entry Game, the variables a and p are defined to be greater than 0. Since those constraints are not contained in the tuple Γ, but are essential preconditions in the verification of the security of the input game, we must include them in the input Π: C is a set of constraints on variables occurring in the utilities that always hold (such as $a, p > 0$ in the Market Entry Game). In other words, the constraints in C must always be fulfilled, regardless of the specific security property. C_{WI}, C_{CR} and C_{P} are the sets of such constraints that are assumed to hold when checking a specific property – weak immunity (C_{WI}), collusion resilience (C_{CR}) or practicality (C_{P}), respectively. Note that all sets containing constraints – C, C_{WI}, C_{CR} and C_{P} – can possibly be empty.

4.2 Finding Joint Strategies

Given our input Π as defined in Section 4.1, we can use the encoding from Chapter 3 to find joint strategies extending an honest history $\beta_o \in \mathscr{O}$ which fulfill the security properties. As discussed in Chapter 3, constraint (3.1) is needed to make sure that a joint strategy can be derived from a model for the resulting constraint set. Additionally, constraint (3.2) further restricts the resulting joint strategy in terms of the terminal history it leads to.

For the constraints which are specific to one of the three security properties – (3.4) for weak immunity, (3.6) for collusion resilience and (3.9) for practicality –, we have to make two slight adjustments in order for them to be applicable: Firstly, we have yet to consider the preconditions for the respective security property in the constraints. Hence, we form an implication with the preconditions on the left-hand side and the respective constraint on the right-hand side.

Secondly, a security property is only satisfied if there is a joint strategy fulfilling the respective criteria for every possible value of the variables occurring in the utilities (i.e. every value that satisfies the preconditions) since the security of a game must not depend on a specific variable assignment. Therefore, we need to universally quantify these variables. We denote the finite set of variables occurring in the utility terms as Var; for example, the set of variables for the Market Entry Game would be $Var = \{a, p\}$. Note that in the remainder of this chapter, we assume that the set Var for a game Γ is non-empty. If Var is empty, i.e. all utilities are constant and numeric, the universal quantification can be omitted.

The resulting constraint for weak immunity given an input instance Π with the finite, non-empty set of variables Var and an honest history $\beta_o \in \mathscr{O}$ – previously constraint (3.4) – thus reads as follows:

$$\forall x \in Var \quad \Big(\big(\bigwedge_{c \in C \cup C_{\mathrm{WI}}} c \big) \Rightarrow \bigwedge_{p \in N} \bigwedge_{\beta \in \mathscr{J}} \big(\big(\bigwedge_{h \in \mathscr{H}^{p}_{a \to \beta}} v^h_a \big) \Rightarrow u^{\mathscr{J}}_p(\beta) \geq 0 \big) \Big). \tag{4.1}$$

Similarly, we can redefine the constraint for collusion resilience, constraint (3.6):

$$\forall x \in Var \quad \Big(\big(\bigwedge_{c \in C \cup C_{\mathrm{CR}}} c \big) \Rightarrow \bigwedge_{S \subset N} \bigwedge_{\beta \in \mathscr{J}} \big(\big(\bigwedge_{h \in \mathscr{H}^{N \setminus S}_{a \to \beta}} v^h_a \big) \Rightarrow \sum_{p \in S} u^{\mathscr{J}}_p(\beta_o) \geq \sum_{p \in S} u^{\mathscr{J}}_p(\beta) \big) \Big). \tag{4.2}$$

Since we introduce new variables, our so-called subgame utility variables (denoted by y_p^h), for the practicality property, we need to quantify over those as well. Hence, this is the resulting constraint for practicality, previously constraint (3.9):

$$
\begin{aligned}
\forall x \in Var \quad & \Bigg(\big(\bigwedge_{c \in C \cup C_{\mathrm{P}}} c \big) \Longrightarrow \\
& \bigwedge_{h \in \mathscr{H} \setminus \mathscr{I}} \Bigg(\forall y_p^h \in Y^h \Big(\big(\bigwedge_{\beta=(a_1,\dots,a_n) \in \mathscr{I}_{|h}} \bigwedge_{p \in N} (v_{a_1}^{(h)} \wedge \dots \wedge v_{a_n}^{(h,\dots,a_{n-1})} \Rightarrow y_p^h = u^{\mathscr{I}}_{|h_p}(\beta)) \big) \Longrightarrow \\
& \qquad \big(\bigwedge_{p \in N} \bigwedge_{\beta \in \mathscr{I}_{|h}} \big(\big(\bigwedge_{h' \in \mathscr{H}^{N \setminus \{p\}}_{|h_{a \to \beta}}} v_a^{(h,h')} \big) \Rightarrow y_p^h \geq u^{\mathscr{I}}_{|h_p}(\beta) \big) \big) \Big) \Bigg) \Bigg),
\end{aligned}
\tag{4.3}
$$

where Y^h is the set of subgame utility variables for the subgame with history h. Formally,

$$Y^h := \{y_p^h : p \in N\}.$$

When checking whether an honest history satisfies one of the security properties, we can use the constraints (3.1) and (3.2) together with the corresponding property constraint – (4.1), (4.2) or (4.3), respectively – as input for any SMT solver that supports universal quantification (such as, for example, Z3 [20]). As mentioned in Chapter 3, if the solver reports satisfiability, we can derive a joint strategy fulfilling the property from the assignments to the Boolean action variables in the model that the solver finds.

Example 4.2.1 (Input Constraints for Market Entry Game – Weak Immunity)**.** Let us revisit Example 3.2.1. We want to check whether the Market Entry Game together with the honest history (o) is weak immune. Therefore, we construct the following set of formulas from (3.1), (3.2) and (4.1):

$$(v_o^{\emptyset} \vee v_e^{\emptyset}) \wedge (\neg v_o^{\emptyset} \vee \neg v_e^{\emptyset}) \wedge (v_i^{(e)} \vee v_s^{(e)}) \wedge (\neg v_i^{(e)} \vee \neg v_s^{(e)}) \wedge v_o^{\emptyset}, \tag{4.4}$$

$$
\begin{aligned}
\forall a, p \quad & \Big(a > 0 \wedge p > 0 \Rightarrow \big((v_o^{\emptyset} \Rightarrow 0 \geq 0) \wedge (v_e^{\emptyset} \Rightarrow p \geq 0) \wedge (v_e^{\emptyset} \Rightarrow -a \geq 0) \wedge \\
& 2p \geq 0 \wedge (v_i^{(e)} \Rightarrow p \geq 0) \wedge (v_s^{(e)} \Rightarrow -a \geq 0) \big) \Big).
\end{aligned}
\tag{4.5}
$$

These constraints can serve as input for an SMT solver: Listing 4.1 displays the resulting encoding in SMT-LIB [19] format. The input from Listing 4.1 can be passed to any SMT solver that supports SMT-LIB [19]. Z3 [20], for instance, yields the output displayed in Listing 4.2[1].

[1]We used Z3 version 4.8.7 to produce the displayed output.

```
; declare Boolean action variables
; v>x>y means that action y is taken at history x
(declare-fun v>>o () Bool)
(declare-fun v>>e () Bool)
(declare-fun v>e>i () Bool)
(declare-fun v>e>s () Bool)

; add constraints
; joint strategy constraints
(assert (or v>>o v>>e))
(assert (or (not v>>o) (not v>>e)))
(assert (or v>e>i v>e>s))
(assert (or (not v>e>i) (not v>e>s)))

; honest history constraint
(assert v>>o)

; weak immunity constraint
(assert (forall
      ((a Real) (p Real))
      (=> (and (> a 0) (> p 0))
        (and (=> v>>o (>= 0 0))
           (and (=> v>>e (>= p 0))
              (and (=> v>>e (>= (- a) 0))
                 (and (>= (* 2 p) 0)
                    (and (=> v>e>i (>= p 0))
                     (=> v>e>s (>= (- a) 0))))))))))

; solve and get model
(check-sat)
(get-model)
```

Listing 4.1: SMT-LIB Input for Weak Immunity of Market Entry Game with History (o)

```
sat
(model
  (define-fun v>e>s () Bool false)
  (define-fun v>e>i () Bool true)
  (define-fun v>>o () Bool true)
  (define-fun v>>e () Bool false)
)
```

Listing 4.2: Z3 [20] Ouput for SMT-LIB Input from Listing 4.1

The output confirms that the set of constraints is satisfiable (line 1 in Listing 4.2) and provides a model (line 3 to 6). In this model, each action variable is assigned a Boolean value: $v_e^\emptyset$ and $v_s^{(e)}$ are set to `false` whereas $v_o^\emptyset$ and $v_i^{(e)}$ are `true`. Hence, the resulting joint strategy is $[\emptyset \to o, (e) \to i]$, which is the same strategy that we have already identified in Example 2.2.1.

4.3 Term Ordering

As mentioned in Section 4.2, for a security property to be fulfilled, it is necessary to consider all possible values which the variables the game contains can take. The security of the off-chain protocol must not depend on specific values; thus, we need to ensure for each security property that there is a joint strategy extending the honest history for each potential variable assignment. Since we are comparing utilities, i.e. terms composed of those variables, in the security properties (either to 0 in the case of weak immunity or to some other utility), the variable assignment is not directly of interest, but rather the ordering of the utility terms.

Depending on the game, there might be one joint strategy for one of the security properties which satisfies all requirements regardless of the concrete term ordering. This is the case in our Market Entry Game example for the weak immunity property, for instance (see Example 4.2.1): The joint strategy $[\emptyset \to o, (e) \to i]$ fulfills the property for any $a, p > 0$. Thus, the specific ordering of a and p is not relevant. With our revised constraints from Section 4.2 – (4.1), (4.2) and (4.3), respectively –, we are able to find such joint strategies. However, for some games, it might not be possible to find one such joint strategy even though the property is satisfied. Instead, we might have to consider multiple cases depending on the ordering of the utilities, meaning that there is more than one joint strategy to find. An example for this is the weak immunity property in the Money Bag Game (see Example 4.3.1).

Game 4.3.1 (Money Bag Game)**.** This game, as displayed in Figure 4.1, consists of two players, D and G, who each have to make up to one decision. There are two bags, a red one and a yellow one, each containing a fixed amount of money. Player G might get a chance to pick one of the bags and gain the money which is in that bag in return. However, whichever bag G chooses, they lose the amount of money that is in the other bag. Player D on the other hand makes one decision at the beginning: whether to let G choose a bag or not. Regardless of how they decide, D gets a reward $c > 0$ for making the decision. Hence, the outcome for D is always the same; they always get the utility c. If they decide not to let player G pick a bag and instead quit the game instantly (denoted by action q in Figure 4.1), G gets the same reward. Otherwise, G has to choose, either the red bag (action r in Figure 4.1) with some monetary value $a > 0$ or the yellow bag (action y) with the monetary value $b > 0$.

In the Money Bag Game, the variables a, b and c are fixed to be greater than 0, but the ordering of the variables, e.g. whether $a > b$ or $b < c$, is not defined. As elaborated in

$$\begin{tikzcd}
 & D \arrow[dl, "q"'] \arrow[dr, "l"] & & \\
(c,c) & & G \arrow[dl, "r"'] \arrow[dr, "y"] & \\
 & (c,a-b) & & (c,b-a)
\end{tikzcd}$$

Figure 4.1: Money Bag Game with $a, b, c > 0$

Example 4.3.1, however, in order to determine a joint strategy which satisfies the weak immunity property, for example, it is crucial to know the ordering of a and b.

Example 4.3.1 (Weak Immunity of Money Bag Game)**.** Consider the Money Bag Game and let us fix the only honest history to be (q) and evaluate the weak immunity property of the game together with this history. To that end, we try to find a weak immune joint strategy σ extending the honest history (q). Weak immunity implies that no honest player can lose resources, regardless of how the other players behave. Since σ must extend (q), there are two possibilities for σ: either $[\emptyset \to q, (l) \to r]$ or $[\emptyset \to q, (l) \to y]$. In both cases, player D chooses action q at the beginning and gains the utility c – which is fixed to be greater than 0 –, no matter which action G would choose. Nevertheless, σ must ensure that if D would deviate from the strategy, i.e. take action l instead, the resulting outcome for G is non-negative as well. G has two options in that case, action r and action y. However, it depends on the ordering of a and b which option yields the better outcome and more importantly, a non-negative utility: If $a > b$, then $a - b > 0$, but $b - a < 0$. Thus, in this case, $[\emptyset \to q, (l) \to r]$ would be a weak immune joint strategy while $[\emptyset \to q, (l) \to y]$ would not. If, on the other hand, $b > a$, then $b - a > 0$ and $a - b < 0$, resulting in the joint strategy $[\emptyset \to q, (l) \to y]$ being weak immune and $[\emptyset \to q, (l) \to r]$ not satisfying the property. If $a = b$, both joint strategies are weak immune. Hence, we fix our weak immune joint strategy σ to be $[\emptyset \to q, (l) \to r]$ if $a > b$ and $[\emptyset \to q, (l) \to y]$ otherwise.

We utilize the concept of *unsatisfiable cores* to overcome the issue of a property depending on the ordering of some terms. Given a set of constraints that is unsatisfiable, an unsatisfiable core is an unsatisfiable subset of those constraints. A *minimal unsatisfiable core* is a subset of the original set of constraints such that removing one constraint from the subset would make the resulting set satisfiable, i.e. the subset contains the minimal number of constraints while still being an unsatisfiable core.

Several SMT solvers, for example, Z3 [20], cvc5 [21] (the successor of CVC4 [22]) and MathSAT5 [23], are able to extract unsatisfiable cores from an unsatisfiable set of constraints. By using Boolean expressions which "label" term comparisons and are asserted and tracked in the unsatisfiable core, we are able to identify necessary case splits for utility terms. This means that whenever two utility terms are compared – either to 0 or to another term and either in an equality or inequality comparison –, instead of just the comparison, we use an implication with a so-called comparison label, i.e. a Boolean expression that represents this term comparison, on the left-hand side and the actual comparison on the right-hand side. For example, instead of the comparison $x > y$ of some terms x and y, we use the equivalent constraints $l_{x,y} \Rightarrow x > y$ and $l_{x,y}$, with $l_{x,y}$ being the label for the comparison of the terms x and y. The latter constraint ($l_{x,y}$) asserts the label, i.e. ensures that it evaluates to `true`.

Following this principle, the constraint for weak immunity, (4.1), is adjusted as follows:

$$\forall x \in Var \quad \Big(\big(\bigwedge_{c \in C \cup C_{\mathrm{WI}}} c \big) \Rightarrow \bigwedge_{p \in N} \bigwedge_{\beta \in \mathscr{I}} \Big(\big(\bigwedge_{h \in \mathscr{H}^p_{a \to \beta}} v^h_a \big) \Rightarrow \big(l_{u^{\mathscr{I}}_p(\beta),0} \Rightarrow u^{\mathscr{I}}_p(\beta) \geq 0 \big) \Big) \Big), \tag{4.6}$$

where Var is the set of variables as before.

Likewise, constraint (4.2) for collusion resilience is rewritten as

$$\begin{aligned} &\forall x \in Var \quad \Big(\big(\bigwedge_{c \in C \cup C_{\mathrm{CR}}} c \big) \Rightarrow \\ &\quad \bigwedge_{S \subset N} \bigwedge_{\beta \in \mathscr{I}} \Big(\big(\bigwedge_{h \in \mathscr{H}^{N \setminus S}_{a \to \beta}} v^h_a \big) \Rightarrow \big(l_{\sum_{p \in S} u^{\mathscr{I}}_p(\beta_o), \sum_{p \in S} u^{\mathscr{I}}_p(\beta)} \Rightarrow \sum_{p \in S} u^{\mathscr{I}}_p(\beta_o) \geq \sum_{p \in S} u^{\mathscr{I}}_p(\beta) \big) \Big) \Big), \end{aligned} \tag{4.7}$$

with β_o denoting the honest history as previously.

Lastly, concerning the practicality property, we revise (4.3) as such:

$$\begin{aligned} &\forall x \in Var \quad \Big(\big(\bigwedge_{c \in C \cup C_{\mathrm{P}}} c \big) \Longrightarrow \\ &\quad \bigwedge_{h \in \mathscr{H} \setminus \mathscr{I}} \Big(\forall y^h_p \in Y^h \Big(\big(\bigwedge_{\beta = (a_1, \dots, a_n) \in \mathscr{I}_{|h}} \bigwedge_{p \in N} (v^{(h)}_{a_1} \wedge \cdots \wedge v^{(h, \dots, a_{n-1})}_{a_n} \Rightarrow \\ &\qquad\qquad (l_{y^h_p, u^{\mathscr{I}}_{|h_p}(\beta)} \Rightarrow y^h_p = u^{\mathscr{I}}_{|h_p}(\beta)))) \Longrightarrow \\ &\qquad (\bigwedge_{p \in N} \bigwedge_{\beta \in \mathscr{I}_{|h}} ((\bigwedge_{h' \in \mathscr{H}^{N \setminus \{p\}}_{|h_{a \to \beta}}} v^{(h,h')}_a) \Rightarrow (l_{y^h_p, u^{\mathscr{I}}_{|h_p}(\beta)} \Rightarrow y^h_p \geq u^{\mathscr{I}}_{|h_p}(\beta))))\Big)\Big)\Big). \end{aligned} \tag{4.8}$$

In all these constraints – (4.6), (4.7) and (4.8), respectively –, the Boolean expression $l_{x,y}$ refers to the label of the comparison of some terms x and y.

As previously mentioned, we assert the comparison labels, i.e. the label expressions must evaluate to `true`, and track only them in the unsatisfiable core, meaning that the unsatisfiable core can only consist of such labels. If the input set of constraints is unsatisfiable, we retrieve a minimal unsatisfiable core from the SMT solver. If this minimal unsatisfiable core contains some comparison label, then there is a contradiction in the comparison of the terms this label represents in the original set of constraints. An example for such a case would be that some term x is supposed to be smaller than some term y according to one comparison ($x < y$) and greater than y in another comparison ($x > y$). Since not both of these constraints can be fulfilled at the same time, i.e. in the same joint strategy, we need to split on the terms x and y and try to find a (different) joint strategy for each case ($x = y$, $x < y$ and $x > y$).

Example 4.3.2 (Input Constraints for Money Bag Game with Labels – Weak Immunity)**.** When applying the comparison labels as introduced in (4.6) to Example 4.3.1, the weak immunity constraints for the Money Bag Game read as follows:

$$(v_q^{\emptyset} \vee v_l^{\emptyset}) \wedge (\neg v_q^{\emptyset} \vee \neg v_l^{\emptyset}) \wedge (v_r^{(l)} \vee v_y^{(l)}) \wedge (\neg v_r^{(l)} \vee \neg v_y^{(l)}) \wedge v_q^{\emptyset}, \tag{4.9}$$

$$\begin{aligned}\forall a, b, c \quad \Big(a > 0 \wedge b > 0 \wedge c > 0 \Rightarrow \big(&(v_q^{\emptyset} \Rightarrow (l_{c,0} \Rightarrow c \geq 0))\wedge \\ &(v_l^{\emptyset} \Rightarrow (l_{c,0} \Rightarrow c \geq 0))\wedge \\ &(v_l^{\emptyset} \Rightarrow (l_{c,0} \Rightarrow c \geq 0))\wedge \\ &c \geq 0 \wedge \\ &(v_r^{(l)} \Rightarrow (l_{a-b,0} \Rightarrow a - b \geq 0))\wedge \\ &(v_y^{(l)} \Rightarrow (l_{b-a,0} \Rightarrow b - a \geq 0))\big)\Big),\end{aligned} \tag{4.10}$$

$$l_{c,0} \wedge l_{a-b,0} \wedge l_{b-a,0}. \tag{4.11}$$

Because of the assertion of the term comparison labels, $l_{c,0}$, $l_{a-b,0}$ and $l_{b-a,0}$, i.e. them being required to be true in the third constraint, the second constraint is not satisfied since $l_{a-b,0}$ and $l_{b-a,0}$ imply that $a - b \geq 0$ and $b - a \geq 0$, respectively, which cannot both be fulfilled unless $a = b$. Hence, we obtain $\{l_{a-b,0}, l_{b-a,0}\}$ as an unsatisfiable core. From this unsatisfiable core, we can derive three possible cases for the ordering of the terms $a - b$ and $b - a$: $a - b = b - a$, $a - b < b - a$ and $a - b > b - a$. We can use SMT solving to find a joint strategy for each of these cases, resulting in the joint strategies described in Example 4.3.1.

In our framework, for the case splitting part, we use an approach similar to the algorithm used in the AVATAR [24] architecture for theorem provers, where two components – a first-order theorem prover and a SAT or an SMT [25] solver – "interact" with each other in order to efficiently perform clause splitting. Algorithm 4.1 displays our approach in pseudocode.

Algorithm 4.1: Term Ordering with Case Splits

```
input  : An input instance $\Pi = (\Gamma, \mathscr{O}, C, C_{\text{WI}}, C_{\text{CR}}, C_{\text{P}})$, an honest history
         $\beta_o \in \mathscr{O}$ and a security property $s \in \{\text{WI, CR, P}\}$
output : true if $s$ is fulfilled in $\Pi$, false otherwise
1  S ← Solver()
2  A ← Solver()
3  AddConstraints (S, ComputeStrategyConstraints ($\Gamma$, $\beta_o$))
4  AddConstraints (A, $C \cup C_s$)
5  $T \leftarrow \emptyset$
6  while Solve (A) = sat do
7      $I \leftarrow$ GetModel (A)
8      $O \leftarrow \{$EvaluateModel $(I, x, y) : (x, y) \in$ Combinations $(T)\}$
9      if Check (S, $s$, $C \cup C_s \cup O$) = sat then // found strategy for current
        ordering
10         AddConstraints (A, $\{\bigvee_{c \in O} \neg c\}$) // add conflict clause
11     else
12         $T' \leftarrow \emptyset$
13         foreach $l_{x,y} \in$ GetUnsatCore(S) do
14             $T' \leftarrow T' \cup \{t : t \in \{x, y\}, t \notin T, t$ is not a subgame utility variable$\}$
15         if $T' = \emptyset$ then // no new expressions, considered every case
16             return false
17         end
18         $T \leftarrow T \cup T'$
19     end
20 end
21 return true
```

The functions used in the pseudocode in Algorithm 4.1 have the following semantics:

- `AddConstraints`(S, F): Adds all constraints in the set F to the SMT solver S.
- `ComputeStrategyConstraints`(Γ, β_o): Returns the conjunction of the constraints (3.1) and (3.2) for the input game Γ and the honest history β_o.
- `Solve`(S): Invokes the SMT solver S, i.e. computes the satisfiability of the set of constraints that were added to the solver, and returns unsat if it is unsatisfiable or sat otherwise.
- `GetModel`(S): Retrieves a model from an SMT solver S on which the previously conducted `Solve` call returned sat, i.e. a variable assignment that satisfies the set of constraints.

- `EvaluateModel`(I, x, y): Returns an (in)equality for the terms x and y, depending on their ordering in the interpretation I. If $I(x) \bullet I(y)$, then the function returns $x \bullet y$, with $\bullet \in \{=, <, >\}$.
- `Combinations`(T): Returns all combinations of size 2 of the terms in T.
- `Check`$(S,\ s,\ P)$: Temporarily adds the universally quantified constraint for the security property s with the set of preconditions P to the SMT solver S and reports satisfiability. In this step, the labels for the term comparisons are asserted and tracked in the unsatisfiable core.
- `GetUnsatCore`(S): Retrieves a minimal unsatisfiable core from an SMT solver S on which a previously conducted satisfiability check returned unsat.

The idea of the algorithm is as follows: We use two SMT solvers, one for finding joint strategies (S in Algorithm 4.1) and the other for identifying the ordering of the utility terms (A). First, we add the constraints (3.1) and (3.2) to S and the preconditions for the variables occurring in the utility terms to A (lines 3 and 4) and initialize a set T of utility terms whose ordering we investigate. In the loop starting at line 6, as a first step, we check whether A reports unsatisfiability on its current set of constraints. If yes, then the preconditions contradict each other, so we do not have to find another joint strategy and the checked security property is fulfilled as we have found a joint strategy for every possible case. Thus, we exit the loop and return true in line 21. Otherwise, we retrieve a model from A and construct a set of comparisons of the terms in T, the set O. We achieve this by computing all combinations of the terms in T (x, y) and adding the constraint of the form $x \bullet y$, with $\bullet \in \{=, <, >\}$, to O, depending on the ordering of x and y in the model. Then, we use solver S to find a joint strategy which fulfills the checked security property given the initial preconditions together with the term ordering obtained from the previous step hold (line 9). If there is a strategy, i.e. the solver returns sat, we add a conflict clause, i.e. a negation of the retrieved term ordering, to solver A so that we can repeat the procedure for the other possible orderings of the terms in T. If there is no joint strategy (meaning that S reports unsatisfiability), we compute a minimal unsatisfiable core. Using our comparison labels, we obtain a "new" set of utility terms, i.e. terms that are not yet considered whose specific ordering is, however, crucial (lines 13 and 14). In the case where the newly computed set is empty, there are no more terms to examine. This means that we have found a term ordering for which there is no joint strategy, i.e. the security property is not fulfilled. Hence, we return false (line 16). If the new set of terms is not empty, we add the new terms to the set T and go back to the start of the loop (line 6).

Example 4.3.3 (Algorithm Execution for Money Bag Game – Weak Immunity)**.** Consider Example 4.3.1 once again. The goal is to find a joint strategy extending the only honest history (q) which is weak immune. Hence, the input for Algorithm 4.1 is $\Pi = (\Gamma_{\text{money}}, \{(q)\}, \{a > 0, b > 0, c > 0\}, \{\}, \{\}, \{\})$, where Γ_{money} is the game depicted in Figure 4.1, $\beta_o = (q)$ and $s = \text{WI}$. At the beginning of the execution of the algorithm, the solvers S and A are instantiated. The first constraint, (4.9), is added to S and the constraints $a > 0, b > 0$ and $c > 0$ are added to A. T is initialized as the empty set. Next, we enter the loop starting at line 6 in Algorithm 4.1. `Solve`(A) returns sat since the set of constraints $\{a > 0, b > 0, c > 0\}$ is satisfiable, e.g. with the assignment $[a \to 1, b \to 1, c \to 1]$. Thus, we enter the loop and get a model of the initial constraints in line 7. In line 8, the set O is assigned the empty set since T is empty. The function call `Check`(S, s, $C \cup C_s \cup O$) in line 9 solves the set of constraints provided in Example 4.3.2 and returns unsat because of the contradiction that for all values of a and b, both $a-b \geq 0$ and $b - a \geq 0$ must hold. Hence, we enter the branch starting at line 12. In line 13 to 14, the terms $a - b$, $b - a$ and 0 (extracted from the labels $l_{a-b,0}$ and $l_{b-a,0}$ in the unsatisfiable core) are added to a new set T'. Since T' is not empty, we add the terms in T' to T (line 18) and start with the next iteration of the loop. `Solve`(A) still returns sat in line 6. Assume that `GetModel`(A) returns the variable assignment from before, $[a \to 1, b \to 1, c \to 1]$. Then, O is assigned the value $\{a-b = 0, b-a = 0, a-b = b-a\}$ in line 8 (calling `Combinations`(T) yields the set $\{(a-b, 0), (b-a, 0), (a-b, b-a)\}$). In line 9, we check if weak immunity is satisfied given that $a, b, c > 0$ and $a-b = 0$, $b-a = 0$ and $a - b = b - a$, which is equivalent to the case $a = b$. As elaborated in Example 4.3.1, the property is fulfilled under these assumptions, meaning that the `Check` call in line 9 returns sat. Consequently, we add the conflict clause $a-b \neq 0 \vee b-a \neq 0 \vee a-b \neq b-a$ to solver A and restart the loop. `Solve`(A) once again reports satisfiability since the set of constraints $\{a > 0, b > 0, c > 0, a-b \neq 0 \vee b-a \neq 0 \vee a-b \neq b-a\}$ is satisfied, for instance, with the variable assignment $[a \to 2, b \to 1, c \to 1]$. Assume that `GetModel`(A) returns this variable assignment. Then, O is fixed to be $\{a-b > 0, b-a < 0, a-b > b-a\}$, which is equivalent to the case $a > b$. The `Check` call in line 9 once again reports satisfiability. Hence, we add the conflict clause $a - b \leq 0 \vee b - a \geq 0 \vee a - b \leq b - a$ to solver A and enter the next iteration. The resulting set of constraints $\{a > 0, b > 0, c > 0, a - b \neq 0 \vee b - a \neq 0 \vee a - b \neq b - a, a - b \leq 0 \vee b - a \geq 0 \vee a - b \leq b - a\}$ is again satisfiable, e.g. with the variable assignment $[a \to 1, b \to 2, c \to 1]$. As before, assume that `GetModel`(A) returns the given variable assignment which satisfies the constraints. As a result, the set O is $\{a - b < 0, b - a > 0, a - b < b - a\}$ in line 8, which represents the case $a < b$. The solver S returns sat in line 9 and the conflict clause $a - b \geq 0 \vee b - a \leq 0 \vee a - b \geq b - a$ is added to the solver A. Next, we go to line 6 again. However, `Solve`(A) returns unsat now because the set of constraints $\{a > 0, b > 0, c > 0, a - b \neq 0 \vee b - a \neq 0 \vee a - b \neq b - a, a - b \leq 0 \vee b - a \geq 0 \vee a - b \leq b - a, a - b \geq 0 \vee b - a \leq 0 \vee a - b \geq b - a\}$ is unsatisfiable. In consequence, we exit the loop and return **true** in line 21 in Algorithm 4.1, indicating that the weak immunity property is fulfilled in the Money Bag Game.

4.4 Infinitesimals

In complex games, like the Closing Game proposed by Rain *et al.* [8], for instance, it might be the case that not all variables in the utilities are (standard) reals. Some variables denote infinitesimals, i.e. numbers which are closer to 0 than any real number, but are not 0. This distinction is highly relevant to our framework since we make comparisons of utility terms which possibly contain such variables.

Our solution is to split each utility term up into a "real" part and an "infinitesimal" part: For example, the term $d_A + \alpha - \epsilon$ from the Closing Game [8], with α and ϵ being infinitesimals, would be split up into the two parts $\langle d_A, \alpha - \epsilon \rangle$. If the term does not contain any infinitesimals, the infinitesimal part is 0, and vice versa. For instance, the term α, which also occurs in the Closing Game [8], would be divided into $\langle 0, \alpha \rangle$. The sum of those two parts constitutes the original term.

In our prototype, we define the arithmetic operations addition, subtraction and multiplication on these utility tuples: For some $\bullet \in \{+, -, \cdot\}$, $\langle r, i \rangle \bullet \langle r', i' \rangle \coloneqq \langle r \bullet r', i \bullet i' \rangle$.

Furthermore, we define the comparative operators, $=$, $\neq$, $\leq$, $\geq$, $<$ and $>$ according to lexicographic order. In the case of equality, both parts of the compared instances must be strictly equal: $\langle r, i \rangle = \langle r, i \rangle \Leftrightarrow r = r' \wedge i = i'$. The inequality operator $\neq$ is defined as the negation of equality, $\langle r, i \rangle \neq \langle r, i \rangle \Leftrightarrow \neg(\langle r, i \rangle = \langle r, i \rangle)$. In the case of the ordering relation $\leq$, there are two possibilities for $t \leq t'$ with $t = \langle r, i \rangle$ and $t' = \langle r', i' \rangle$: Either the real part of t is smaller than the real part of t', i.e. $r < r'$, which means that t is smaller than t' since adding infinitesimals (which are smaller than any real number) cannot result in t getting greater than t'. In this case, $t \leq t'$ holds. Otherwise, if $r = r'$, the result of $t \leq t'$ depends on the infinitesimal parts. Hence, $\langle r, i \rangle \leq \langle r', i' \rangle \Leftrightarrow (r < r' \vee (r = r' \wedge i \leq i'))$. With the same reasoning, it is clear that $\langle r, i \rangle \geq \langle r', i' \rangle \Leftrightarrow (r > r' \vee (r = r' \wedge i \geq i'))$. Similarly, $\langle r, i \rangle < \langle r', i' \rangle \Leftrightarrow (r < r' \vee (r = r' \wedge i < i'))$ and $\langle r, i \rangle > \langle r', i' \rangle \Leftrightarrow (r > r' \vee (r = r' \wedge i > i'))$.

In our framework, each term is parsed to an internal representation of a utility tuple. All operations on terms are actually conducted on the corresponding utility tuples. Note that this slight adjustment also influences the implementation of Algorithm 4.1 and the utility comparisons in the property constraints (4.6), (4.7) and (4.8). However, this is only an implementation detail; the core concept of the algorithm and the formulas stays the same.

CHAPTER 5

Evaluation

We implemented a prototype for our analysis framework CheckMate, as described in Chapter 4, using Python (version 3.10) and the Python API[1] for the Z3 [20] SMT solver (version 4.8.17). The implementation can be found on GitHub[2]. Our framework takes a JSON file encoding an instance Π as defined in Section 4.1 as input, parses it to an internal representation and then checks each security property for every specified honest history using the Z3 API in the algorithm that we introduced in Chapter 4 (Algorithm 4.1). It outputs the result of each analysis as well as the necessary term ordering case splits which occur during the analysis.

5.1 Additional Capabilities

Aside from the evaluation of a security property by finding suitable joint strategies with SMT solving, which is the main purpose of our framework, our work can also be used with the following two features.

Firstly, it can support the enumeration of all possible joint strategies for the input game. This can be achieved by first computing all individual strategies for each player and then constructing all possible combinations of the single players' strategies.

Secondly, our prototype is able to check whether a given joint strategy satisfies one of the three security properties. To this end, we once again use SMT solving: We construct feasible SMT constraints from the joint strategy and the input game and a given term ordering, similarly to our approach to finding a joint strategy. However, in this case, we do not need action variables since we already have a joint strategy at hand.

[1] https://ericpony.github.io/z3py-tutorial/guide-examples.htm, accessed on August 30, 2022

[2] https://github.com/apre-group/checkmate, accessed on August 30, 2022

Note that combining these two features yields a second way of assessing a security property: In theory, it is possible to iterate over all joint strategies and check for each strategy if it satisfies the checked property. If yes, the property is fulfilled. If no joint strategy satisfies the property, the property is not fulfilled. The applicability of this – more naive – approach to evaluating a security property is, however, limited due to the iteration of joint strategies: While it is useful for small games where there are few joint strategies, it is not feasible to evaluate complex games such as the Closing Game [8] which has trillions of possible joint strategies.

5.2 Experimental Examples

We tested our framework using a set of small example games, namely the Market Entry Game, the Money Bag Game, the Application Game and the Pirate Game. Apart from these example games which do not model real off-chain protocols, we also tested the framework on one real-world example, namely the Closing Game defined by Rain *et al.* [8] modeling the protocol for the closing phase in the Bitcoin Lightning Network [7].

Game 5.2.1 (Application Game)**.** The game is depicted in Figure 5.1. It models the decision-making process of a player A that thinks about applying for a job at a company C. Hence, A has two options in the beginning: Either do nothing (action n), or apply (a). If they do not apply, they get the utility $t > 0$ which stands for the time they save by not having to write an application. Otherwise, their outcome depends on the company C which they send the application to. If the company offers player A a job (action o), player A gets the outcome j, representing the benefits of receiving a job offer. In the case where C declines, denoted by d in Figure 5.1, the applicant A might get feedback on their application, denoted by utility f. In this application scenario, there are two types of jobs: attractive jobs, meaning that they have excellent working conditions, and unappealing jobs with bad working conditions. If A files an application, their best outcome depends on the type of job that they apply to: If it is an attractive job, there are many applicants. As a consequence, company C does not have time to provide feedback on every single application. Therefore, the utility of not applying, t, would be greater than the utility of applying and being declined (f). On the other hand, if the application pays off, i.e. they apply and get a job offer, the resulting utility (j) is greater than the utility of not applying. Similarly, if it is an application for a rather unattractive job, the company receives few applications and hence, has the time to provide detailed feedback to each applicant. Hence, the utility of not applying, t, is greater than the benefit of getting a job offer (j). However, if player A does write an application and gets declined, the utility of receiving feedback (f) is greater than the utility of not applying. We denote this as $j > t > f \vee f > t > j$.

Game 5.2.2 (Pirate Game)**.** In this version of the Pirate Game (originally introduced in [26] as “a puzzle for pirates”), we have four pirates who found a chest full of gold coins. They have to decide how to distribute the discovered gold among them. To achieve this, they follow a certain scheme: The oldest pirate proposes a distribution first. Note that

$$\begin{tikzcd}
 & A \arrow[dl, "n"'] \arrow[dr, "a"] & & \\
(t,0) & & C \arrow[dl, "o"'] \arrow[dr, "d"] & \\
 & (j,0) & & (f,0)
\end{tikzcd}$$

Figure 5.1: Application Game with $t > 0$ and $j > t > f \vee f > t > j$

in our version, choosing a distribution is not part of the game. Then, all pirates submit a vote, either for the proposed distribution or against it, starting with the proposing pirate in (descending) order of age. If the majority of the pirates votes yes (action y in Figure 5.2), the gold is distributed according to the proposal. In case of a tie, the outcome is determined by the vote that the proposing pirate has submitted after making the proposal. For example, if the oldest pirate votes yes in the beginning and after that, another pirate votes yes as well, the proposed distribution is accepted. If the majority votes no (action n), the proposing pirate is thrown overboard and dies (resulting in utility $-d$ with $d > 0$) and the process starts over with the next oldest pirate.

$$\begin{tikzcd}
 & & A \arrow[dl, "y"'] \arrow[dr, "n"] & & \\
 & B \arrow[dl, "y"'] \arrow[dd, "n"'] & & B \arrow[dr, "n"] \arrow[dd, "y"] & \\
(a_A, a_B, a_C, a_D) & & & & S_B \\
 & C \arrow[dl, "y"'] \arrow[dd, "n"'] & & C \arrow[dr, "n"] \arrow[dd, "y"] & \\
(a_A, a_B, a_C, a_D) & & & & S_B \\
 & D \arrow[dl, "y"'] \arrow[dd, "n"'] & & D \arrow[dr, "n"] \arrow[dd, "y"] & \\
(a_A, a_B, a_C, a_D) & & & & S_B \\
 & S_B & & (a_A, a_B, a_C, a_D) &
\end{tikzcd}$$

Figure 5.2: Pirate Game with $d, g > 0$, $a_A, a_B, a_C, a_D, b_B, b_C, b_D, c_C, c_D \geq 0$ and $a_A + a_B + a_C + a_D = b_B + b_C + b_D = c_C + c_D = g$

In Figure 5.2, player A denotes the oldest pirate, B the second oldest and so on. Furthermore, the utilities a_A, a_B, a_C and a_D denote the share of the gold coins that player A, B, C or D, respectively, receives according to the proposal by A. Analogously, b_B, b_C and b_D denote the outcome for player B, C and D, respectively, when following player B's proposal, and c_C and c_D the utilities for player C and D, respectively, in the case where the pirates vote for pirate C's proposed distribution. If the three oldest pirates are dead, the youngest pirate gets all the gold, denoted by utility g, with $g > 0$. Hence, $a_A + a_B + a_C + a_D = b_B + b_C + b_D = c_C + c_D = g$.

The subgames where the second or third oldest pirate, player B or C, respectively, proposes a distribution (subgames S_B and S_C), are depicted in Figure 5.3 and Figure 5.4, respectively.

$$\begin{tikzcd}
 & & B \arrow[dl, "y"'] \arrow[dr, "n"] & & \\
 & C \arrow[dl, "y"'] \arrow[ddd, "n"'] & & C \arrow[ddd, "y"] \arrow[dr, "n"] & \\
(-d, b_B, b_C, b_D) & & & & S_C \\
 & & & & \\
 & D \arrow[dl, "y"'] \arrow[dd, "n"'] & & D \arrow[dd, "y"] \arrow[dr, "n"] & \\
(-d, b_B, b_C, b_D) & & & & S_C \\
 & S_C & & (-d, b_B, b_C, b_D) &
\end{tikzcd}$$

Figure 5.3: Sugame S_B in Figure 5.2

$$\begin{tikzcd}
 & C \arrow[dl, "y"'] \arrow[dr, "n"] & \\
(-d, -d, c_C, c_D) & & (-d, -d, -d, g)
\end{tikzcd}$$

Figure 5.4: Sugame S_C in Figure 5.3

For testing purposes, we fixed the set of honest histories of the Market Entry Game, the Money Bag Game and the Application Game to be the set of all respective terminal histories. In the Pirate Game, we checked the terminal histories (y, y), (y, n, n, n, y, y) and (n, n, n, n, n). When it comes to the Closing Game [8], the honest histories are (H) and (C_h, S).

Table 5.1 shows an overview of the game instances together with the respective checked histories that we used for testing, including an indicator whether the history is weak immune (WI), collusion resilient (CR) and practical (P) and a reference to the corresponding (informal) justifications. If the joint strategy satisfying a property is dependent on the ordering of some terms, this is indicated in the respective column by listing those terms.

In the following subsections, for each game that is not the Closing Game [8], we provide a justification for the assessment of all tested terminal histories with respect to the three security properties, as displayed in Table 5.1.

Game	**Honest History**	**WI**	**CR**	**P**	**Justification**
Market Entry Game	(o)	yes	yes	no	Section 5.2.1
	(e, i)	no	yes	yes	
	(e, s)	no	no	no	
Money Bag Game	(q)	yes – a, b	yes	yes – a, b	Section 5.2.2
	(l, r)	no	no	no	
	(l, y)	no	no	no	
Application Game	(n)	yes	yes – f, j, t	yes – f, j, t	Section 5.2.3
	(a, o)	no	no	no	
	(a, d)	no	no	no	
Pirate Game	(y, y)	no	no	no	Section 5.2.4
	(y, n, n, n, y, y)	no	no	no	
	(n, n, n, n, n)	no	no	no	
Closing Game [8]	(H)	yes	yes	no	[8]
	(C_h, S)	yes	yes	yes – c, p_A	

Table 5.1: Test Instances

5.2.1 Market Entry Game

We tested the Market Entry Game with all terminal histories, namely (o), (e, i) and (e, s). The security properties for history (o) are proven in Example 2.2.1.

Consider the Market Entry Game together with history (e, i). The only joint strategy extending this history is $[\emptyset \to e, (e) \to i]$. This strategy is not weak immune because player T could lose resources if player E deviates, i.e. takes action s instead of i, due to the resulting utility of $-a$ and the precondition $a > 0$. However, it is collusion resilient because neither T nor E (which constitute the only strict subgroups) benefits from deviating: If T would deviate, they would get utility 0 which is smaller than p as $p > 0$. Similarly, if E would pick action s over i, this would result in the utility $-a$ for E which is smaller than the outcome they get when choosing action i as $a, p > 0$. Lastly, the joint strategy $[\emptyset \to e, (e) \to i]$ is practical as well since in each subgame, no player gets a better outcome when unilaterally deviating from the joint strategy. The proof for the practicality property follows the same reasoning as for collusion resilience. In conclusion, the history (e, i) is collusion resilient and practical, but not weak immune and thus, not secure.

When it comes to history (e, s) of the Market Entry Game, the only extending joint strategy is $[\emptyset \to e, (e) \to s]$. $[\emptyset \to e, (e) \to s]$ is not weak immune since it yields a negative utility for both players, that is, $-a$, since $a > 0$. The joint strategy is not collusion resilient either because both players can actually get a better utility by deviating from the strategy: Player T could take action o instead of e and receive the utility 0 which is greater than $-a$ and E could deviate from action s to i and get the outcome p which is greater than $-a$ as well since $a, p > 0$. Following the same reasoning, the joint strategy does not satisfy the practicality property. To sum up, the history (e, s) does not fulfill any of the three security properties and is therefore not secure.

5.2.2 Money Bag Game

As for the Market Entry Game, we used all terminal histories of the Money Bag Game for testing: (q), (l, r) and (l, y).

Consider the terminal history (q) first. There are two joint strategies extending history (q), namely $[\emptyset \to q, (l) \to r]$ and $[\emptyset \to q, (l) \to y]$. For a proof that the history is weak immune, see Example 4.3.1. Regarding the second property, both joint strategies extending (q) are collusion resilient because the only strict subgroups of players are the single players and neither player D nor player G gains any benefit from deviating: If D deviates, D gets the same utility, c. If G deviates, the outcome of the game does not change because in both strategies, D picks action q. With respect to practicality, it has to be ensured that in each subgame, no player can get a better outcome by choosing an action that differs from the joint strategy. For the subgame that is the entire game, this can be proven to be true using the same reasoning as for collusion resilience. In the subgame following history (l), however, it depends on the ordering of a and b which action yields the better utility for player G, similarly to the weak immunity property. When setting the played joint strategy to be $[\emptyset \to q, (l) \to r]$ if $a > b$ and $[\emptyset \to q, (l) \to y]$ otherwise, the practicality property is satisfied. In consequence, the history (q) of the Money Bag Game is weak immune, collusion resilient and practical and hence, secure.

Next, consider the Money Bag Game together with the terminal history (l, r). The only joint strategy that extends (l, r) is $[\emptyset \to l, (l) \to r]$. This joint strategy is not weak immune because it yields a negative utility for player G in the case where $a < b$. Since there is no joint strategy satisfying the weak immunity property for each possible value of a, b and c, the property is not fulfilled. Similarly, when it comes to collusion resilience, player G could get a better outcome by deviating from $[\emptyset \to l, (l) \to r]$ if $a < b$, causing the property to not be fulfilled. Following the same reasoning, it becomes clear that the joint strategy is not practical either. As a result, the history (l, r) does not fulfill any of the security properties and is thus not secure.

Lastly, consider history (l, y). The only joint strategy extending this history is $[\emptyset \to l, (l) \to y]$. However, analogously to the reasoning for the history (l, r), this joint strategy can be proven to be neither weak immune nor collusion resilient nor practical. Consequently, the history (l, y) is not secure.

5.2.3 Application Game

For the Application Game, we once again considered all terminal histories in our testing. These are (n), (a, o) and (a, d).

First, consider history (n), which is extended by two joint strategies, $[\emptyset \to n, (a) \to o]$ and $[\emptyset \to n, (a) \to d]$. Both of these joint strategies are weak immune: If player A adheres to the strategy and player C deviates, the outcome for A (t) does not change and is greater than 0. On the other hand, if A chooses action a over q, no matter what action player C chooses (o or d), their utility is 0; hence, they cannot lose resources either. With respect to collusion resilience, the joint strategy depends on the ordering of f, j and t: If $j > t > f$, then the collusion resilient joint strategy is $[\emptyset \to n, (a) \to d]$ because in this case, the subgroup $\{A\}$ does not benefit from deviating, i.e. choosing action a instead of n, as $t > f$. Player C (the other strict subgroup) gets the same utility, no matter which action they choose. Similarly, if $f > t > j$, the joint strategy $[\emptyset \to n, (a) \to o]$ is collusion resilient. Since there are no other possible orderings for f, j and t according to the preconditions, there is a joint strategy for every possible case and the history (n) is therefore collusion resilient. Following the same reasoning, it is clear that (n) is practical as well. Hence, the history (n) is weak immune, collusion resilient and practical and thus, secure.

Next, consider the terminal history (a, o). The only joint strategy extending this history is $[\emptyset \to a, (a) \to o]$. This joint strategy is not weak immune because it could be the case that the resulting utility for A (j) is smaller than 0. It is not collusion resilient either because in this case, i.e. if $f > t > j$, A would get a better outcome if they deviate to action n. The same issue arises when considering the practicality property. Since the history (a, o) is neither weak immune nor collusion resilient nor practical, it is not secure.

Likewise, the history (a, d) is not secure, which can be proven following the same reasoning as for history (a, o), but with the ordering $j > t > f$. Like history (a, o), it does not fulfill any of the security properties.

5.2.4 Pirate Game

We tested the Pirate Game together with the terminal histories (y, y), (y, n, n, n, y, y) and (n, n, n, n, n).

Regarding the first security property, in the Pirate Game, no joint strategy and thus, no terminal history, is weak immune. This is because the oldest pirate cannot prevent their own death, i.e. getting utility $-d$ which is smaller than 0 as $d > 0$, since they only get to make a choice in the beginning and the resulting outcome of the game entirely depends on how the other players vote.

Moreover, no terminal history can be collusion resilient: All possible terminal histories can be categorized by the outcome they yield; either they result in implementing pirate A's proposal or pirate B's or pirate C's. However, consider, for example, any arbitrary joint strategy that extends a history resulting in the outcome (a_A, a_B, a_C, a_D) (representing the oldest pirate's proposal). If a strict subgroup of players, e.g. $\{B, C, D\}$, decides to vote no on A's proposed distribution of the gold and choose a proposal which is possibly better for them instead (for instance, pirate B's proposal), i.e. to jointly deviate from the joint strategy, they can achieve a better outcome. This is because the resulting sum of the utilities of B, C and D, e.g. $b_B + b_C + b_D$, is greater than the sum $a_B + a_C + a_D$ in the case where $a_A > 0$. The same holds true for all other proposed distributions (terminal histories); there is always a case (i.e. assignment of values to the variables a_A, a_B etc.) in which no joint strategy extending the terminal history is collusion resilient.

Lastly, none of the considered terminal histories is practical. For each joint strategy extending one of the checked histories, there exists an ordering of the distribution variables (a_A, a_B etc.) such that in some subgame, some player benefits from deviating. For example, consider the history (y, y) and the ordering $b_B > a_B$, $c_C > b_C > a_C$, $b_D > a_D > c_D$. In any joint strategy extending history (y, y), A must take action y in the beginning and B must choose action y next, i.e. the joint strategy must be of the form $[\emptyset \to y, (y) \to y, \dots]$. Consider the smallest subgame, S_C, first: Player C must choose action y in order for the joint strategy to be a Nash equilibrium, so S_C can be replaced with the utility $(-d, -d, c_C, c_D)$. In the two subgames where D picks an action in S_B, D picks action y for the same reason since $b_D > c_D$. Thus, those subgames can be replaced with the utility $(-d, b_B, b_C, b_D)$. At the point in subgame S_B where pirate C submits their vote on B's proposal after B has voted yes, both possible actions result in the same payoff; hence, the subgame starting with C can be replaced with the outcome $(-d, b_B, b_C, b_D)$. Similarly, at the point in subgame S_B where pirate B has voted no and it is C's turn to vote, C must take action n because $c_C > b_C$, causing the subgame starting with C to be substituted with the utility $(-d, -d, c_C, c_D)$. This results in B choosing action y in S_B because $b_B > -d$. Therefore, the entire subgame S_B can be substituted with the utility representing B's proposal, $(-d, b_B, b_C, b_D)$. Consequently, as $b_D > a_D$, pirate D would vote no on A's proposed distribution in the subgame following history (y, n, n). Further, pirate C would need to take action n in the subgame starting from history (y, n) because $b_C > a_C$. Thus, this subgame can be replaced with the utility of pirate B's proposal. However, as $b_B > a_B$, B would get a better outcome by deviating from the joint strategy $[\emptyset \to y, (y) \to y, \dots]$. Hence, the history (y, y) is not practical. A similar justification can be provided for the other tested terminal histories – (y, n, n, n, y, y) and (n, n, n, n, n), respectively – as well.

In conclusion, neither of the tested terminal histories is weak immune, collusion resilient or practical. Hence, none of them is secure.

5.3 Experimental Results

Our framework provided a correct assessment for all test instances listed in Section 5.2. Most notably, it detected the insecurity of the honest history (H) and security of the honest history (C_h, S) of the Closing Game, which are both proven manually in [8].

With respect to the performance of our framework, we conducted our test runs using an Intel Core™ i7-8550U CPU @ 1.80 GHz with 8 GB of RAM. Table 5.2 displays the execution time in seconds (rounded to two decimals) for a complete security analysis of each checked history of our test instances. A complete security analysis consists of assessing the checked history considering each of the three security properties.

Most importantly, the evaluation of the only real-world example, the Closing Game [8], was finished within about one minute in total. It is worth mentioning that analyzing the honest history (H) of the Closing Game [8] took approximately twice as much time as conducting the analysis on the other honest history, (C_h, S). This can be explained

Game	**Honest History**	**Execution Time (s)**
Market Entry Game	(o)	0.20
	(e, i)	0.17
	(e, s)	0.24
Money Bag Game	(q)	0.33
	(l, r)	0.27
	(l, y)	0.27
Application Game	(n)	0.20
	(a, o)	0.20
	(a, d)	0.19
Pirate Game	(y, y)	314.63
	(y, n, n, n, y, y)	22.96
	(n, n, n, n, n)	7.48
Closing Game [8]	(H)	46.73
	(C_h, S)	23.07

Table 5.2: Test Results

by taking the number of expressions to compare into consideration: For the analysis of practicality of history (H), the ordering of up to twelve terms is considered in one iteration of the loop in Algorithm 4.1. In comparison, the only two terms whose ordering is taken into account when evaluating history (C_h, S) are c and p_A, resulting in only two very simple case splits.

Another interesting result is the execution time of approximately five minutes for history (y, y) of the Pirate Game: For history (y, y), the failing case for practicality – as elaborated in Section 5.2.4 – specifies the ordering of several different terms. Since in every iteration of the loop in Algorithm 4.1, one possible ordering of the terms is considered (after these terms have been identified), many iterations are potentially necessary to find the particular ordering with which the practicality property is not satisfied, thus resulting in a relatively long run time.

Lastly, for the rather simple Market Entry Game, Money Bag Game and Application Game, the execution time was below one second.

5.4 Limitations

Although our framework provided a correct analysis of the games that we tested it on, our approach has a few limitations.

First of all, our prototype was almost exclusively tested on small sample inputs that do not represent real models of off-chain protocols. Since at the time of writing there exist only very few accurate game models, the number of tests on representative instances is limited. As a consequence, our assessment regarding the applicability of our prototype is limited as well: While our framework works well on the small example games that we used for testing and on the Closing Game [8], it remains yet to be seen how it performs on other models for off-chain protocols. We plan to solve this issue in the future by developing more game-theoretic models and using them as input for our framework.

With respect to testing, it would also be interesting to see if our approach is applicable to games with many players as well since our framework was exclusively tested on games with a rather small number of players. To that end, a more extensive analysis with a stronger focus on the performance of the framework would be a possible next step in future work.

Moreover, our prototype heavily relies on the preconditions for the game and for the security properties. The framework assesses the properties under the assumption that the preconditions hold; hence, the formulation of proper preconditions still has to be done manually. For an extensive automated analysis, however, it would be critical to evaluate what conditions would have to be fulfilled for a property to hold. In future work, we aim to overcome this limitation by incorporating a search for conditions under which each property holds into the automated security analysis.

Additionally to the dependency on preconditions which have to be identified manually, we emphasize that our prototype is currently limited to automating the analysis of the security of off-chain protocols modeled as games. The game-theoretic models which serve as input for our framework still have to be developed manually. Implementing an automated creation of such games will be subject of future work.

Lastly, in its current state, our framework only assesses whether the checked security property holds given the preconditions and outputs the resulting joint strategy if it does. On the other hand, if the property is not satisfied, the framework currently does not provide a detailed justification. However, regarding the security analysis, it is of great interest to see which dishonest (malicious) strategies yield a better outcome than the honest behavior in order to identify vulnerabilities in the underlying protocol. One approach to solve this would be to compute and provide counterexamples, i.e. joint strategies which are strictly better than all honest strategies with regard to the checked security property.

CHAPTER 6

Conclusion

Game-theoretic models for off-chain protocols allow for an accurate and extensive representation of all interactions between different agents. Therefore, employing game theory is a promising approach when it comes to identifying vulnerabilities and proving the security of off-chain protocols. However, the manual creation and analysis of such game-theoretic models is a tedious and error-prone task due to the protocols' complexity.

In this thesis, we presented a prototype for a framework which facilitates this process by automating the evaluation of the security of an off-chain protocol modeled as a game based on three established game-theoretic security properties. In this regard, we propose an encoding of the input game and the security properties as SMT formulas. By applying SMT solving, it is possible to deduce whether the properties are fulfilled. We use this assessment to reason about the security of the underlying protocol.

Our approach produces a correct evaluation of several small example games for scenarios which are not related to off-chain protocols. Regarding real protocols, we show that our framework correctly assesses the security of the Closing Game presented by Rain *et al.* [8] which models the closing phase of the Bitcoin Lightning Network [7]. With respect to performance, our prototype provided a result of the security analysis within approximately one minute for the honest histories of the highly complex Closing Game [8].

Nevertheless, our prototype poses certain limitations. The most important issue is that our framework only automates the proof that the properties are (not) fulfilled in a provided game, meaning that the game-theoretic models which are used as input for our framework still have to be created manually. Thus, the process of analyzing the security of off-chain protocols is not fully automated yet. Furthermore, despite the prototype being relatively efficient when applied to our test instances, due to the limited availability of game models of real off-chain protocols, more tests on real instances are required to extensively evaluate the applicability of the framework to real-world examples.

Despite the limitations of our framework, we have taken a first step towards automating the security analysis of off-chain protocols based on game-theoretic models, which is our main contribution. Future research on this topic might possibly focus on improving our prototype, e.g. by providing a detailed justification in case a security property is not satisfied, and extending it to automate the creation of game-theoretic models as well.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] IDC, Apr. 19, 2021, “Worldwide spending on blockchain solutions from 2017 to 2024 (in billion U.S. dollars),” Statista. [Online]. Available: https://www.statista.com/statistics/800426/worldwide-blockchain-solutions-spending. [Accessed: Aug. 30, 2022].

[2] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2009. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed: Aug. 30, 2022].

[3] G. Wood, “Ethereum: A Secure Decentralised Generalised Transaction Ledger,” 2014. [Online]. Available: https://gavwood.com/paper.pdf. [Accessed: Aug. 30, 2022].

[4] Statista, Feb. 21, 2022, “Distribution of Bitcoin and other crypto in the overall market from 2nd quarter of 2013 to 4th quarter of 2021,” Statista. [Online]. Available: https://www.statista.com/statistics/730782/cryptocurrencies-market-capitalization. [Accessed: Aug. 30, 2022].

[5] J. Göbel and A. E. Krzesinski, “Increased block size and bitcoin blockchain dynamics”, in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, Melbourne, Australia: IEEE, Nov. 2017. DOI: 10.1109/ATNAC.2017.8215367.

[6] A. Hafid, A. S. Hafid, and M. Samih, “Scaling blockchains: A comprehensive survey”, *IEEE Access*, vol. 8, pp. 125 244–125 262, Jul. 2020. DOI: 10.1109/ACCESS.2020.3007251.

[7] J. Poon and T. Dryja, “The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments,” 2016. [Online]. Available: https://lightning.network/lightning-network-paper.pdf. [Accessed: Aug. 30, 2022].

[8] S. Rain, Z. Avarikioti, L. Kovács, and M. Maffei, “Towards a game-theoretic security analysis of off-chain protocols”, Jul. 2022. DOI: 10.48550/ARXIV.2109.07429.

[9] C. T. Do, N. H. Tran, C. Hong, *et al.*, “Game theory for cyber security and privacy”, *ACM Comput. Surv.*, vol. 50, no. 2, pp. 30:1–30:37, May 2017. DOI: 10.1145/3057268.

[10] Z. Liu, N. C. Luong, W. Wang, *et al.*, “A survey on blockchain: A game theoretical perspective”, *IEEE Access*, vol. 7, pp. 47 615–47 643, Apr. 2019. DOI: 10.1109/ACCESS.2019.2909924.

[11] P. Zappalà, M. Belotti, M. Potop-Butucaru, and S. Secci, "Game theoretical framework for analyzing blockchains robustness", in *35th International Symposium on Distributed Computing (DISC 2021)*, S. Gilbert, Ed., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 209, Freiburg, Germany: Schloss Dagstuhl, Oct. 2021, pp. 42:1–42:18. DOI: 10.4230/LIPIcs.DISC.2021.42.

[12] S. Mazumdar, P. Banerjee, A. Sinha, S. Ruj, and B. Roy, "Strategic analysis of griefing attack in lightning network", Jul. 2022. DOI: 10.48550/ARXIV.2203.10533.

[13] D. Robinson, "HTLCs considered harmful," in *Stanford Blockchain Conference (SBC 2019)*, Stanford, CA, USA, Jan. 2019. [Online]. Available: https://cbr.stanford.edu/sbc19. [Accessed: Aug. 30, 2022].

[14] G. Malavolta, P. A. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability", in *Network and Distributed System Security Symposium (NDSS 2019)*, San Diego, CA, USA, Feb. 2019. DOI: 10.14722/ndss.2019.23330.

[15] R. D. McKelvey, A. M. McLennan, and T. L. Turocy, *Gambit: Software tools for game theory, version 16.0.0*, 2014. [Online]. Available: http://www.gambit-project.org. [Accessed: Aug. 30, 2022].

[16] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos, "PRISM-games 3.0: Stochastic game verification with concurrency, equilibria and time", in *Computer Aided Verification (32nd International Conference, CAV 2020)*, S. K. Lahiri and C. Wang, Eds., Los Angeles, CA, USA: Springer, Jul. 2020, pp. 475–487. DOI: 10.1007/978-3-030-53291-8_25.

[17] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*. Cambridge, Massachusetts: MIT Press, 1994, ISBN: 978-0-262-65040-3.

[18] C. Barrett and C. Tinelli, "Satisfiability modulo theories", in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds., Cham: Springer, May 2018, ch. 11, pp. 305–343. DOI: 10.1007/978-3-319-10575-8.

[19] C. Barrett, P. Fontaine, and C. Tinelli, "The Satisfiability Modulo Theories Library (SMT-LIB)," 2016. [Online]. Available: https://smtlib.cs.uiowa.edu/index.shtml. [Accessed: Aug. 30, 2022].

[20] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver", in *Tools and Algorithms for the Construction and Analysis of Systems (14th International Conference, TACAS 2008)*, C. R. Ramakrishnan and J. Rehof, Eds., ser. Lecture Notes in Computer Science, vol. 4963, Budapest, Hungary: Springer, Mar. 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

[21] H. Barbosa, C. Barrett, M. Brain, *et al.*, "Cvc5: A versatile and industrial-strength SMT solver", in *Tools and Algorithms for the Construction and Analysis of Systems (28th International Conference, TACAS 2022)*, D. Fisman and G. Rosu, Eds., ser. Lecture Notes in Computer Science, vol. 13243, Munich, Germany: Springer, Apr. 2022, pp. 415–442. DOI: 10.1007/978-3-030-99524-9_24.

[22] C. Barrett, C. L. Conway, M. Deters, *et al.*, "CVC4", in *Computer Aided Verification (23rd International Conference, CAV 2011)*, G. Gopalakrishnan and S. Qadeer, Eds., ser. Lecture Notes in Computer Science, vol. 6806, Snowbird, UT, USA: Springer, Jul. 2011, pp. 171–177. DOI: `10.1007/978-3-642-22110-1_14`.

[23] A. Cimatti, A. Griggio, B. Schaafsma, and R. Sebastiani, "The MathSAT5 SMT solver", in *Tools and Algorithms for the Construction and Analysis of Systems (19th International Conference, TACAS 2013)*, N. Piterman and S. A. Smolka, Eds., ser. Lecture Notes in Computer Science, vol. 7795, Rome, Italy: Springer, Mar. 2013, pp. 93–107. DOI: `10.1007/978-3-642-36742-7_7`.

[24] A. Voronkov, "AVATAR: The architecture for first-order theorem provers", in *Computer Aided Verification (26th International Conference, CAV 2014)*, A. Biere and R. Bloem, Eds., ser. Lecture Notes in Computer Science, vol. 8559, Vienna, Austria: Springer, Jul. 2014, pp. 696–710. DOI: `10.1007/978-3-319-08867-9_46`.

[25] G. Reger, N. Bjørner, M. Suda, and A. Voronkov, "AVATAR modulo theories", in *2nd Global Conference on Artificial Intelligence (GCAI 2016)*, C. Benzmüller, G. Sutcliffe, and R. Rojas, Eds., ser. EPiC Series in Computing, vol. 41, EasyChair, Sep. 2016, pp. 39–52. DOI: `10.29007/k6tp`.

[26] I. Stewart, "A puzzle for pirates", *Scientific American*, vol. 280, no. 5, pp. 98–99, May 1999, ISSN: 19467087.