# CheckMate: Automated Game-Theoretic Security Reasoning

Lea Salome Brugger[*]
ETH Zurich
Zurich, Switzerland
leasalome.brugger@inf.ethz.ch

Laura Kovács
TU Wien
Vienna, Austria
laura.kovacs@tuwien.ac.at

Anja Petković Komel
TU Wien
Vienna, Austria
anja.komel@tuwien.ac.at

Sophie Rain
TU Wien
Vienna, Austria
sophie.rain@tuwien.ac.at

Michael Rawson
TU Wien
Vienna, Austria
michael@rawsons.uk

## ABSTRACT

We present the CheckMate framework for full automation of game-theoretic security analysis, with particular focus on blockchain technologies. CheckMate analyzes protocols modeled as games for their game-theoretic security — that is, for incentive compatibility and Byzantine fault-tolerance. The framework either proves the protocols secure by providing defense strategies or yields all possible attack vectors. For protocols that are not secure, CheckMate can also provide weakest preconditions under which the protocol becomes secure, if they exist. CheckMate implements a sound and complete encoding of game-theoretic security in first-order linear real arithmetic, thereby reducing security analysis to satisfiability solving. CheckMate further automates efficient handling of case splitting on arithmetic terms. Experiments show CheckMate scales, analyzing games with trillions of strategies that model phases of Bitcoin's Lightning Network.

## CCS CONCEPTS

• **Security and privacy → Formal security models**; **Logic and verification**.

## KEYWORDS

Security Analysis, Automated Reasoning, Game Theory, Secure Protocols, Decentralized Protocols.

**ACM Reference Format:**
Lea Salome Brugger, Laura Kovács, Anja Petković Komel, Sophie Rain, and Michael Rawson. 2023. CheckMate: Automated Game-Theoretic Security Reasoning. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623183

## 1 INTRODUCTION

Applications of blockchain technology such as cryptocurrencies [27] and decentralized finance [35] are becoming increasingly popular. Establishing security guarantees of such applications is mostly driven by formal analysis of the underlying cryptographic protocols [3, 17, 26, 34]. While powerful, these efforts cannot capture malicious actions that are possible in spite of formal cryptographic guarantees. Game-theoretic security analysis has therefore emerged [31, 36], introducing variants of extensive form games (EFGs) [29] for embedding punishment mechanisms within blockchain analysis.

In a nutshell, game-theoretic security analysis enables reasoning about incentive-compatibility: that is, whether malicious yet cryptographically-possible behavior is discouraged via punishment mechanisms. It also enables detecting and even preventing scenarios that could lead directly to security attack vectors [23]. Feasibility of game-theoretic models for investigating an underlying protocol's security partially depends on the game completeness, that is, on expressing all possible interactions between players. In consequence, accurate models are likely to be rather large and complex games. For example, while [31] introduces a so-called Closing Game to precisely model the closing phase in Bitcoin's Lightning Network [30], we show there are trillions of possible joint strategies (combinations of player strategies) for the Closing Game (see Example 3.4). As such, manually analyzing game-theoretic security models is not practically viable.

*In this paper, we therefore introduce the CheckMate framework for automating reasoning about game-theoretic security of blockchain protocols.* To the best of our knowledge, CheckMate provides the first automated reasoning framework for enforcing game-theoretic security, (dis)proving, for example, security of real-world protocols used within Bitcoin's Lightning Network (Section 6). Related reasoning approaches for (extensive form) games exist [18], but current techniques are limited to processing games with *numeric* values as game utility variables enacting punishment or reward mechanisms. In CheckMate, we advocate for the use of *symbolic* values, guaranteeing security for every possible numeric value, e.g. every possible account balance in decentralized finance applications.

The distinctive feature of CheckMate is a formalization of security properties over game strategies in such a way that the result can be (dis)proved using "only" first-order arithmetic reasoning with limited quantification (Section 4). To this end, *we turn applications of blockchain security into a satisfiability modulo theory (SMT) problem*, by showing that first-order linear real arithmetic provides an expressive logic to formulate and prove game-theoretic security (Lemma 4.1). Our first-order encoding is exact and, unlike the work of [18, 19], does not feature probabilistic (reward) operators.

[*]All authors contributed equally to this research.

*CCS '23, November 26–30, 2023, Copenhagen, Denmark*
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0050-7/23/11.
https://doi.org/10.1145/3576915.3623183

Instead, we provide a decidable logic for game-theoretic security and omit the computational burden of reasoning with uncertainty.

The added value of our first-order encoding is witnessed when formalizing that deviating from the protocol is never rational (incentive compatibility), and that even if adversaries deviate, honest users are not financially harmed (Byzantine fault-tolerance) (Section 2). We show the formalization is sound and complete: our security proofs imply game-theoretic security and vice versa (Theorem 5.2). In this respect, we introduce novel reasoning approaches on top of SMT solving, scaling and using formal verification not only for enforcing game-theoretic security, but also providing counterexamples and/or refining preconditions where security properties are violated (Section 5).

*Our Contributions.* We bring the following main contributions[1].

- **(i)** We formalize game-theoretic security properties as first-order arithmetic formulas over EFG joint strategies (Section 4), reducing security analysis of blockchain transactions to arithmetic reasoning over honest game histories with symbolic game utilities (Lemma 4.1). The use of symbolic utilities differentiates our work from other game-theoretic frameworks [18, 19]: symbolic utilities allow us to avoid concurrent game strategies while providing a deterministic, game-theoretic behavior. As such, we also avoid reasoning with probabilistic (reward) operators.
- **(ii)** Since players may be willing to forgo some intangible assets, such as opportunity cost, but not actual resources, we introduce *weaker immunity* (Definition 3.6), strengthening the state of the art in game-theoretic security analysis. Our formalization is sound and complete with respect to their game-theoretic definitions (Theorem 5.2), and inhabits a decidable fragment of first-order arithmetic.
- **(iii)** Unlike [31, 36], our EFG security properties avoid the use of non-trivial sets, functions and quantifier alternations. We show that this arguably simple logical formalization is both sufficient and necessary to precisely capture game-theoretic security (Theorem 5.2). Moreover, we provide tailored automated reasoning approaches over EFG strategies and bring them into the landscape of SMT solving (Algorithm 1).
- **(iv)** Since we reason about symbolic utilities, proving arithmetic relations naturally yields case splits. We guide these case distinctions via unsatisfiable (unsat) core computation in SMT solving (Section 5.1).
- **(v)** For EFG properties that are not secure, we provide attack vectors as concrete counterexamples to a violated security property (Section 5.2, Algorithm 2). In addition, we give weakest ordering conditions on utilities which, if assumed as preconditions, ensure security (Section 5.3, Algorithm 3).
- **(vi)** We implement our approach in the new tool CheckMate, a fully-automated security reasoning engine for EFGs that requires no user guidance (Section 6). We evaluate CheckMate on challenging EFGs, including variants of real-world protocols namely closing and routing phases of Bitcoin's Lightning Network. Experiments demonstrate applicability and scalability of CheckMate, (dis)proving security of EFGs with trillions of strategies and thousands of nodes. While for readability's sake, our running examples use simplified game models of Lightning's closing and routing phases (Figures 1–2), our experiments show that CheckMate succeeds when analyzing the respective protocols in full (Table 1).

[1]with additional proofs and details given in [4]

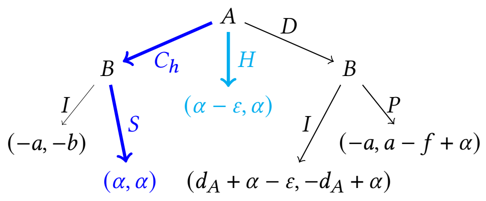


**Figure 1: Simplified Closing Game with $\alpha, a, \varepsilon, f, d_A > 0$ and $\alpha, \varepsilon$ infinitesimals.**

## 2 MOTIVATING EXAMPLES

We motivate and illustrate our work by considering simplified versions of the closing and routing protocol phases of Bitcoin's Lightning Network [31]. Let us emphasize that our running examples from Figures 1–2 are simplified only for the sake of readability. In experiments, we also evaluate CheckMate on full models of the respective protocols of Bitcoin's Lightning Network (Section 6). In particular, the last two entries of Tables 1 and 2 report on our results when analyzing Bitcoin's Lightning Network in its *full* complexity.

The closing and routing phases of the Lightning Network are modeled as EFGs and visualized as trees in Figures 1 and 2. The start of an EFG is the root, and players' choices lead to different branches. The game ends when a leaf is reached, where a player's gain or loss is called their utility.

*Example 2.1 (Simplified Closing Game).* In the Simplified Closing Game of Figure 1, player $A$ starts the game and chooses between three options: closing honestly ($H$), collaboratively honestly ($C_h$), or dishonestly ($D$). If $A$ chooses $H$, *both players get the benefit* of closing the channel $\alpha$, but player $A$ has to wait until the closing times out, so the utility is reduced by the opportunity cost $\varepsilon$. If $A$ chooses $C_h$, player $B$ gets to choose between ignoring ($I$), i.e. the funds remain locked, or signing ($S$), where both players get the benefit of closing $\alpha$. If $A$ chooses to close dishonestly with some deviating amount $d_A$, then if $B$ chooses to ignore ($I$), the funds for $B$ are lost; however, if $B$ proves ($P$) the attempt was dishonest, all of $A$'s funds ($a$) are redistributed to $B$, but the transaction fee $f$ has to be paid. Note that the utilities in the leaves of the game tree are actual variables ($\alpha$, $a$, $f$, etc.), not numeric values, and $\alpha$, $\varepsilon$ are infinitesimals (see Section 3).

*Example 2.2 (Simplified Routing Game).* CheckMate can handle games with any finite number of players. To show how an attack vector can arise from collusion between players and outline the main structure of the model of the routing protocol, we include in Figure 2 an EFG with five players, modeling a Simplified Routing Game. Player $A$ is the initiator of the routing transaction, player $B$ is the receiver and players $P_1$, $P_2$ and $P_3$ are intermediaries. The routing starts when player $B$ sends a hash of a secret to player $A$; this step is modeled with action $S_H$. Then, a so-called locking phase follows (the four actions $L$), where players lock funds for the next player in the routing path to unlock, provided they can present $B$'s secret: player $A$ locks the amount $m + 3f$ (where $f$ represents the routing fee), player $P_1$ locks $m + 2f$, player $P_2$ locks $m + f$ and player $P_3$ locks $m$. Next, the game enters an unlocking phase, where players choose between the honest action $U$ of unlocking their contracts, or to ignore unlocking ($I_U$), resulting in a state where all contracts still locked expire and a leaf is reached. The end utilities of players depend on which contracts are unlocked, which are expired and additionally on two subjective values: the benefit of updating, modeled as a positive infinitesimal $\rho$, and the opportunity cost, modeled as an infinitesimal $\varepsilon$.

The path that represents honest behavior is depicted in thick blue lines. CheckMate disproves security of the game, as the path depicted in dashed purple deviates from the honest behavior and corresponds to the so-called Wormhole attack [23]. Within this dishonest behavior (attack), player $P_3$ can additionally choose to share the secret with player $P_1$ and thus bypass player $P_2$ entirely. If $P_1$ then chooses to unlock, it results in a negative utility ($-\varepsilon$) for player $P_2$, who is further deprived of earning the routing fee $f$ for enabling the transaction, which they would get in the honest scenario depicted in blue.

## 3 PRELIMINARIES

We introduce game-theoretic material needed for our work. We assume familiarity with standard first-order logic [33], linear (real) arithmetic [32], and SMT solving over both [2, 9, 10].

### 3.1 Game Theory

We define a *game* to be a static finite object with finitely many *players*. Players choose from finitely many *actions* until the game ends, whereupon they receive a *utility*. We focus on perfect-information *Extensive Form Games* (EFGs) in which players choose actions sequentially with full knowledge of all previous actions. Games may yield collective benefit or loss, i.e. they are not necessarily *zero-sum*.

*Definition 3.1 (Extensive Form Game — EFG).* An *extensive form game* $\Gamma = (N, G)$ is determined by a finite non-empty set of players $N$ together with a finite tree $G = (V, E)$. A game path $h = (e_1, ..., e_n), e_i \in E$ starting from the root of $G$ is called a *history*. We denote the set of histories by $\mathcal{H}$. There is a bijection between nodes $v \in V$ and histories $h \in \mathcal{H}$ that lead to these nodes.

- A history that leads to a leaf is called *terminal* and belongs to the set of *terminal histories* $\mathcal{T} \subseteq \mathcal{H}$. Terminal histories $t$ are associated with a *utility* for each player.
- *Non-terminal histories* $h$ are those histories that are not terminal $h \in \mathcal{H} \setminus \mathcal{T}$. Non-terminal histories $h$ have a player $P(h) \in N$ whose turn it is, who may choose from a set of possible actions $A(h)$ to take after $h$.

In game theory, utilities are usually real- or integer-valued numeric constants. Similarly to [31], in our work we however consider utilities as *symbolic* terms in linear arithmetic capturing all possible utilities with certain constraints. We evaluate variables and constants over the real numbers $\mathbb{R}$ extended by a finite set of *infinitesimals*, closer to zero than any real number. Infinitesimals model subjective inconveniences or benefits that do not relate directly to funds, such as opportunity cost. For our purposes, we model infinitesimals by considering linear terms over $\mathbb{R} \times \mathbb{R}$, ordered lexicographically: the first component represents the real part, the second the infinitesimal. We write real for the first projection and avoid writing pairs; that is, we write 0 for $(0, 0)$. In the sequel, we use $a, b, c \ldots$ for real variables, and write $\alpha, \beta, \gamma, \ldots$ for infinitesimals. The utility term $a + \alpha - \varepsilon$ is therefore represented in our work as $(a, 0) + (0, \alpha) - (0, \varepsilon)$, that is $(a, \alpha - \varepsilon)$.

*Example 3.2.* The Simplified Closing Game has two players $N = \{A, B\}$. After empty history $\emptyset$, it is the turn of player $P(\emptyset) = A$ to choose from actions $A(\emptyset) = \{H, C_h, D\}$. After terminal history $(H)$, player $A$ receives utility $\alpha - \varepsilon$ and $B$ receives $\alpha$.

While utilities depend only on terminal histories, we relate utilities with *joint strategies*, facilitating formulation of security properties in the sequel.

*Definition 3.3 (EFG Properties).* Let $\Gamma = (N, G)$ be an EFG.

**Joint Strategy** A *joint strategy* $\sigma$ is a function mapping every non-terminal history $h \in \mathcal{H} \setminus \mathcal{T}$ to an action $a \in A(h)$. The set of joint strategies is $\mathcal{S}$.

**Single Strategy** A *strategy* $\sigma_p \in \mathcal{S}_p$ of player $p$ is a function mapping non-terminal histories $h \in \mathcal{H} \setminus \mathcal{T}$ with $P(h) = p$ to an action $a \in A(h)$. Similarly, a group of players $S \subset N$ may have a strategy $\sigma_S \in \mathcal{S}_S$.

**Strategy Deviation** If player $p$ *deviates* from a joint strategy $\sigma \in \mathcal{S}$ with another strategy $\tau_p \in \mathcal{S}_p$, the resulting joint strategy is denoted $\sigma[\tau_p/\sigma_p]$. Similarly, for a deviating group of players $S \subset N$, we write $\sigma[\tau_S/\sigma_S]$.

**Resulting History** The *resulting terminal history* $H(\sigma)$ of a strategy $\sigma$ is the unique history obtained by following chosen actions in $\sigma$ from root to leaf.

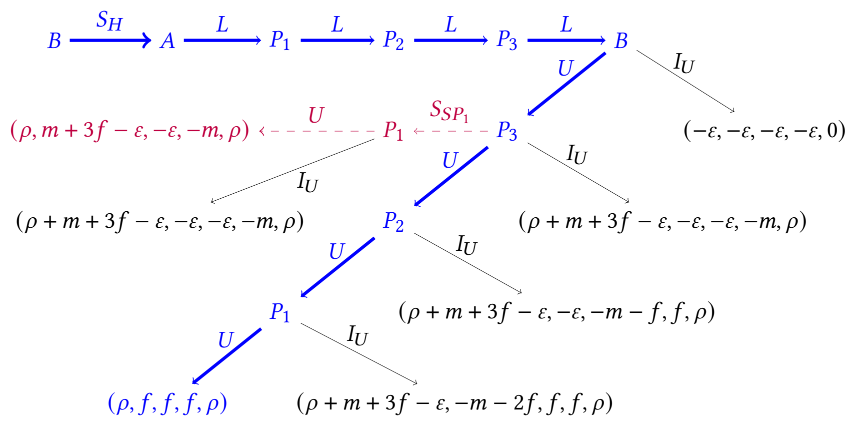
**Extended Strategy** An *extended joint strategy* $\beta \in \mathcal{S}$ of a history $h \in \mathcal{T}$ is a strategy whose resulting history is $h$. Thus, $H(\beta) = h$.

**Players' Subhistories** Let $\mathcal{H}_t^S$ denote the *set of non-empty histories* leading to terminal history $t$ where the last turn was one of the players' $p \in S$. That is, $\mathcal{H}_t^S := \{(h, a) \mid \exists h'.\ t = (h, a, h') \wedge P(h) \in S\}$. For simplicity, let $\mathcal{H}_t^p := \mathcal{H}_t^{\{p\}}$.

**Utility Function** The *utility function* $u_p(\sigma)$ assigns a utility for every joint strategy $\sigma \in \mathcal{S}$ to player $p \in N$. We sometimes write all player utilities for a joint strategy as $u(\sigma)$, denoting a tuple of size $|N|$. Since utilities only depend on $\sigma$'s terminal history $h = H(\sigma)$, we define $u_p(h) := u_p(\sigma)$.

**Subgames** *Subgames* $\Gamma_{|h}$ of $\Gamma$ are formed from the same set $N$ of players and a subtree of $G$, and are therefore identified by a history $h$ leading to the subtree $G_{|h}$. Histories $\mathcal{H}_{|h}$ of $\Gamma_{|h}$ are histories in $\mathcal{H}$ with prefix $h$, and similarly for the utility function $u_{|h}$ and strategies $\sigma_{|h} \in \mathcal{S}_{|h}$.

*Example 3.4.* In Figure 1, a joint strategy $\tau$ could be player $A$ taking action $H$ initially, with player $B$ taking $S$ after $(C_h)$ and $P$ after $(D)$. Player $A$'s single strategy $\tau_A$ takes action $H$ initially. Player $B$ receives $u_B(\tau) = \alpha$. The history resulting from $\tau$ is $(H)$, and $\tau$ is a strategy extending history $(H)$. The subgame for history $(C_h)$ has players $N = \{A, B\}$ and has a tree where player $B$ must choose between action $I$ with utility $(-a, -b)$ and action $S$ with utility $(\alpha, \alpha)$.



$B \xrightarrow{S_H} A \xrightarrow{L} P_1 \xrightarrow{L} P_2 \xrightarrow{L} P_3 \xrightarrow{L} B$
$U$
$I_U$
$(-\varepsilon, -\varepsilon, -\varepsilon, -\varepsilon, 0)$
$(\rho, m + 3f - \varepsilon, -\varepsilon, -m, \rho)$
$U$
$P_1$
$S_{SP_1}$
$P_3$
$I_U$
$(\rho + m + 3f - \varepsilon, -\varepsilon, -\varepsilon, -m, \rho)$
$U$
$I_U$
$(\rho + m + 3f - \varepsilon, -\varepsilon, -\varepsilon, -m, \rho)$
$P_2$
$U$
$I_U$
$(\rho + m + 3f - \varepsilon, -\varepsilon, -m - f, f, \rho)$
$P_1$
$U$
$I_U$
$(\rho, f, f, f, \rho)$
$(\rho + m + 3f - \varepsilon, -m - 2f, f, f, \rho)$

**Figure 2: Simplified Routing Game with $m, f, \rho, \varepsilon > 0$ and $\rho, \varepsilon$ infinitesimals.**

The Simplified Closing Game has $3 \cdot 2 \cdot 2$ joint strategies as player $A$ chooses one out of three possible actions, and independently of that $B$ picks one action out of two in both subtrees. Similarly, we reach the conclusion that the Closing Game [31] listed in Table 1 has $1.6307 \ldots \cdot 10^{13}$ (16 trillion) joint strategies.

## 3.2 Game-Theoretic Security Properties

Since an adversary may perform an attack for one of two reasons (personal gain or harming somebody), a protocol is *game-theoretically secure* according to [31, 36], if the following two properties hold:

(P1) (Byzantine fault-tolerance) Even in the presence of adversaries, honest players do not suffer loss; thus, in a secure protocol an honest player will not receive negative utility, independent of others' behavior. Therefore, there are no "attacks" where somebody is harmed.

(P2) (incentive-compatibility) Rational agents do not deviate from the honest behavior, as the honest behavior yields the best payoff. Hence, in a secure protocol, a rational "attacker" is behaving honestly and no adversary gets personal gain by deviation.

In the sequel, we fix an arbitrary EFG $\Gamma = (N, G)$ and give all definitions relative to $\Gamma$. Based on [31], property (P1) is ensured by *weak immunity* as follows.

*Definition 3.5 (Weak Immunity).* A joint strategy $\sigma \in \mathscr{S}$ in EFG $\Gamma$ is *weak immune* if all players $p$ that follow $\sigma$ always receive non-negative utility:

$$\gamma_{\text{wi}}(\sigma): \qquad \forall p \in N\ \forall \tau \in \mathscr{S}.\ u_p(\tau[\sigma_p/\tau_p]) \geq 0\ . \qquad (\gamma_{\text{wi}})$$

Strategy $\tau$ from Example 3.4 is weak immune, as long as $\alpha \geq \varepsilon$ and $a \geq f$.

In our work, we found weak immunity to be too restrictive (see Section 6). We therefore revise [31] and propose *weaker immunity* to ensure (P1) as follows:

*Definition 3.6 (Weaker Immunity).* A joint strategy $\sigma$ in EFG $\Gamma$ is *weaker immune* if all players $p$ that follow $\sigma$ always receive at least a negative infinitesimal:

$$\gamma_{\text{weri}}(\sigma): \qquad \forall p \in N\ \forall \tau \in \mathscr{S}.\ \mathsf{real}(u_p(\tau[\sigma_p/\tau_p])) \geq 0\ . \quad (\gamma_{\text{weri}})$$

(P2) is ensured by *collusion resilience* and *practicality*. Collusion resilience requires honest behavior to yield the best payoff, even in the presence of collusion.

*Definition 3.7 (Collusion Resilience).* A joint strategy $\sigma$ in EFG $\Gamma$ is *collusion resilient* if colluding players $S \subset N$ cannot profit from deviation:

$$\gamma_{\text{cr}}(\sigma): \qquad \forall S \subset N\ \forall \tau \in \mathscr{S}.\ \sum_{p \in S} u_p(\sigma) \geq \sum_{p \in S} u_p(\sigma[\tau_S/\sigma_S])\ . \qquad (\gamma_{\text{cr}})$$

Strategy $\tau$ from Example 3.4 is not collusion resilient, since player $A$ could deviate by choosing $C_h$ initially and obtain $\alpha$, while by following $\tau$ they receive only $\alpha - \epsilon$. Practicality ensures that for all player decisions, the honest behavior is also "greedy": if all players act selfishly (that is, maximizing their own utilities), the honest choice yields the best utility.

*Definition 3.8 (Practicality).* A joint strategy $\sigma$ in EFG $\Gamma$ is *practical* if it is a subgame perfect equilibrium, i.e. a Nash equilibrium in every subgame:

$$\begin{aligned} \gamma_{\text{pr}}(\sigma): \quad & \forall h \in \mathscr{H}\ \forall p \in N\ \forall \tau \in \mathscr{S}_{|h}. \\ & u_{|h,p}(\sigma_{|h}) \geq u_{|h,p}(\sigma_{|h}[\tau_p/\sigma_{|h_p}])\ . \end{aligned} \qquad (\gamma_{\text{pr}})$$

Strategy $\tau$ from Example 3.4 is not practical. In the subgame after history $(C_h)$, the selfish choice for $B$ is to choose action $S$. Assuming player $B$ acts this way, player $A$'s greedy strategy is to choose action $C_h$ initially with expected utility $\alpha$ instead of $\alpha - \epsilon$. Using $(\gamma_{\text{wi}})$, $(\gamma_{\text{cr}})$, and $(\gamma_{\text{pr}})$, we formalize *security* as in [31]:

*Definition 3.9 (Security).* A terminal history $h^*$ of an EFG $\Gamma$ is *secure* if there are three strategies extending $h^*$ that satisfy $(\gamma_{\text{wi}})$, $(\gamma_{\text{cr}})$ and $(\gamma_{\text{pr}})$, respectively.

Our notion of security given in Definition 3.9 ensures that players can defend every attack. As the defense strategy may vary depending on the attack, different strategies for $(\gamma_{\text{wi}})$, $(\gamma_{\text{cr}})$ and $(\gamma_{\text{pr}})$ are allowed.

## 4 FIRST-ORDER ARITHMETIC THEORY OF SECURITY PROPERTIES

We now introduce our first-order formalization of game-theoretic security, by exploiting and adjusting the EFG security properties of Section 3 within the first-order theory of linear real arithmetic. Our formalization ensures that if a first-order (security) formula is satisfiable, a model for the formula provides a *joint strategy for a given honest history* of an EFG (Section 4.1). We present our first-order formalization piecewise, as constraints on such models, imposing among others that models must form a joint strategy, the joint strategy should result in a given honest history, and user-supplied assumptions should be considered (Section 4.2–Section 4.4). We consider the game utilities to be symbolic terms evaluated over pairs of real values, as mentioned in Section 3.1. We therefore universally quantify their symbolic variables in our encoding. As before, EFG $\Gamma = (N, G)$ is arbitrarily fixed.

### 4.1 Joint Strategies and Honest Histories

A joint strategy for an EFG $\Gamma = (N, G)$ selects exactly one action for each internal node of the tree. We introduce Boolean *action variables* $v_a^h$ to indicate whether at non-terminal history $h$ a player chooses action $a$, and constrain these variables $v_a^h$ so that exactly one variable is assigned for each $h$. We thus have

$$\bigwedge_{h\in\mathscr{H}\setminus\mathscr{T}} \underbrace{\left(\bigvee_{a\in A(h)} v_a^h\right)}_{\text{(ALO)}} \wedge \underbrace{\bigwedge_{a_i,a_j\in A(h),\, a_i\neq a_j} \left(\neg v_{a_i}^h \vee \neg v_{a_j}^h\right)}_{\text{(AMO)}}. \qquad (\phi_{\text{strat}})$$

Constraint (ALO) ensures that each non-terminal history $h$ has at least one action variable set. The at-most-one constraint (AMO) [28] ensures that no more than one $v_a^h$ is set. Our next lemma then concludes that a model $\mathsf{I}$ of $\phi_{\text{strat}}$ uniquely describes a joint strategy $\sigma \in \mathscr{S}$ and vice versa.

Lemma 4.1 (Model-Strategy Translation). *Consider the EFG $\Gamma$ with joint strategies $\mathscr{S}$. Let $\mathcal{M}$ be models of the formula $\phi_{strat}$. Then, $f : \mathcal{M} \to \mathscr{S}$, where*

$$f(\mathsf{I}) = \sigma \quad \Longleftrightarrow \quad \forall h \in \mathscr{H} \setminus \mathscr{T} : \sigma(h) = a \leftrightarrow \mathsf{I}(v_a^h) = \top \quad (1)$$

*is a well-defined bijection.*

Proof. Recall that

$$\mathscr{S} = \{\sigma |\ \sigma : \mathscr{H} \setminus \mathscr{T} \to \bigcup_{h\in\mathscr{H}\setminus\mathscr{T}} A(h),\ \sigma(h) \in A(h)\} \quad \text{and}$$

$$\mathcal{M} = \{\mathsf{I} |\ \mathsf{I} : (v_a^h)_{a\in A(h)}^{h\in\mathscr{H}\setminus\mathscr{T}} \to \{\top, \bot\},\ \mathsf{I}(\phi_{\text{strat}}) = \top\}.$$

We start by showing that $f$ is well-defined. Let $\mathsf{I} \in \mathcal{M}$ and define $\sigma := f(\mathsf{I})$. From $\mathsf{I}(\phi_{\text{strat}}) = \top$, it follows that for all $h \in \mathscr{H} \setminus \mathscr{T}$, there exists an action $a \in A(h)$ such that $\mathsf{I}(v_a^h) = \top$ as well as there cannot be two different $a_i, a_j \in A(h)$ such that $\mathsf{I}(v_{a_i}^h) = \mathsf{I}(v_{a_j}^h) = \top$. By definition of $f$, this says exactly that for every $h \in \mathscr{H} \setminus \mathscr{T}$, there exists precisely one action $a \in A(h)$ such that $\sigma(h) = a$. Therefore, $\sigma$ is a function from $\mathscr{H} \setminus \mathscr{T}$ to $\bigcup_{h\in\mathscr{H}\setminus\mathscr{T}} A(h)$ with $\sigma(h) \in A(h)$ for all $h$. Hence, $\sigma \in \mathscr{S}$.

Next, we show the injectivity of $f$. Let $\mathsf{I}, \mathsf{I}' \in \mathcal{M}$, $\mathsf{I} \neq \mathsf{I}'$. Then, there exists a $v_a^h$ such that $\mathsf{I}(v_a^h) \neq \mathsf{I}'(v_a^h)$. Without loss of generality, we assume $\mathsf{I}(v_a^h) = \top$. For $\sigma := f(\mathsf{I})$, $\sigma' := f(\mathsf{I}')$, it follows $\sigma(h) = a \neq \sigma'(h)$. Thus, $\sigma \neq \sigma'$ and hence, $f$ is injective.

Lastly, we show the surjectivity of $f$. We pick an arbitrary $\sigma \in \mathscr{S}$ and consider $\mathsf{I}$ with $\mathsf{I}(v_a^h) = \top$ iff $\sigma(h) = a$. This $\mathsf{I}$ is a function $(v_a^h)_{a\in A(h)}^{h\in\mathscr{H}\setminus\mathscr{T}} \to \{\top, \bot\}$. Since $\sigma$ is a function with $\sigma(h) \in A(h)$, we know that for all $h \in \mathscr{H} \setminus \mathscr{T}$, there exists exactly one $a \in A(h)$ such that $\mathsf{I}(v_a^h) = \top$. Therefore, $\mathsf{I}$ is a model of $\phi_{\text{strat}}$ and $\mathsf{I} \in \mathcal{M}$. We conclude $f$ is surjective and thus a well-defined bijection. □

Lemma 4.1 is the crux of our work, *reducing game-theoretic security analysis to satisfiability modulo first-order linear real arithmetic:* game-theoretic security holds iff first-order formulas describing security properties are satisfiable. In what follows, we introduce the first-order formulas capturing game-theoretic security, which then together with Lemma 4.1 enable the automation of (dis)proving game-theoretic security (Section 5). To this end, we extend honest histories and hence further constrain EFG joint strategies. We do so by ensuring that all action variables in the honest history are set. That is, for an honest history $h^* = (a_1, \ldots, a_n)$, we obtain

$$v_{a_1}^{\emptyset} \wedge \cdots \wedge v_{a_n}^{(a_1,\ldots,a_{n-1})}. \qquad (\phi_{\text{hist}})$$

*Example 4.2.* Consider the Simplified Closing Game with honest history $(C_h, S)$. From $\phi_{\text{strat}}$ and $\phi_{\text{hist}}$, we obtain the following constraints on action variables:

$$\underbrace{\left(v_H^{\emptyset} \vee v_{C_h}^{\emptyset} \vee v_D^{\emptyset}\right) \wedge \left(\neg v_H^{\emptyset} \vee \neg v_{C_h}^{\emptyset}\right)}_{\text{Constraints from } \phi_{\text{strat}} \text{ for } A.}$$

$$\wedge \underbrace{\left(\neg v_H^{\emptyset} \vee \neg v_D^{\emptyset}\right) \wedge \left(\neg v_{C_h}^{\emptyset} \wedge v_D^{\emptyset}\right)}_{\text{Constraints from } \phi_{\text{strat}} \text{ for } A.}$$

$$\wedge \underbrace{\left(v_I^{(C_h)} \vee v_S^{(C_h)}\right) \wedge \left(\neg v_I^{(C_h)} \vee \neg v_S^{(C_h)}\right)}_{\text{Constraints from } \phi_{\text{strat}} \text{ for } B.}$$

$$\wedge \underbrace{\left(v_I^{(D)} \vee v_P^{(D)}\right) \wedge \left(\neg v_I^{(D)} \vee \neg v_P^{(D)}\right)}_{\text{Constraints from } \phi_{\text{strat}} \text{ for } B.} \wedge \underbrace{v_{C_h}^{\emptyset} \wedge v_S^{(C_h)}}_{\text{Constraints from } \phi_{\text{hist}}.}$$

### 4.2 Weak and Weaker Immunity

A joint strategy is weak immune ($\gamma_{\text{wi}}$) if the utility of each player following the strategy is non-negative, no matter how other players behave. For each possible terminal history, we thus need to ensure that if a player takes the corresponding actions, the resulting utility for the player is greater than or equal to 0. The set of non-terminal histories leading to a terminal history $t$ where it is the turn of player $p$ is $\mathscr{H}_t^p$, as defined in Definition 3.3. We then formalize weak immunity as

$$\bigwedge_{p\in N} \bigwedge_{t\in\mathscr{T}} \left[\bigwedge_{(h,a)\in\mathscr{H}_t^p} v_a^h\right] \to u_p(t) \geq 0. \qquad (\phi_{\text{wi}})$$

Moreover, we express weaker immunity ($\gamma_{\text{weri}}$) as

$$\bigwedge_{p\in N} \bigwedge_{t\in\mathscr{T}} \left[\bigwedge_{(h,a)\in\mathscr{H}_t^p} v_a^h\right] \to \mathsf{real}(u_p(t)) \geq 0. \qquad (\phi_{\text{weri}})$$

To ensure that models of $\phi_{\text{wi}}$ and $\phi_{\text{weri}}$ yield weak(er) immune joint strategies of our EFG $\Gamma$, we respectively combine the constraints $\phi_{\text{strat}}$ and $\phi_{\text{hist}}$ with $\phi_{\text{wi}}$ and $\phi_{\text{weri}}$.

*Example 4.3.* In order to find a weak immune joint strategy for the Simplified Closing Game, we add the following formula to the constraints from Example 4.2:

$$\left.\begin{array}{r}(v_H^{\emptyset} \to \alpha - \varepsilon \geq 0) \land (v_{C_h}^{\emptyset} \to -a \geq 0)\\ \land\ (v_D^{\emptyset} \to d_A + \alpha - \varepsilon \geq 0)\\ \land\ (v_D^{\emptyset} \to -a \geq 0) \land (v_{C_h}^{\emptyset} \to \alpha \geq 0)\end{array}\right\}\text{Constraints for } A.$$
$$\left.\begin{array}{r}\land\ \alpha \geq 0 \land (v_I^{(C_h)} \to -b \geq 0)\\ \land\ (v_S^{(C_h)} \to \alpha \geq 0)\\ \land\ (v_I^{(D)} \to -d_A + \alpha \geq 0)\\ \land\ (v_P^{(D)} \to a - f + \alpha \geq 0)\end{array}\right\}\text{Constraints for } B.$$

Note that the first constraint for player $B$ does not contain an implication. This is because player $B$ does not make a choice at terminal history $(H)$ and consequently, the empty antecedent is omitted. If we consider the honest history $(C_h, S)$, we can simplify the formula by $\phi_{\text{hist}}$ and propositional reasoning, but weak immunity does not hold as the constraint $v_{C_h}^{\emptyset} \to -a \geq 0$ is not satisfied for $a > 0$. When in turn checking for weaker immunity, we can disregard infinitesimal terms, so the constraints from $\phi_{\text{weri}}$ simplify to:

$$\underbrace{(v_{C_h}^{\emptyset} \to -a \geq 0) \land (v_D^{\emptyset} \to d_A \geq 0) \land (v_D^{\emptyset} \to -a \geq 0)}_{\text{Constraints for } A.} \quad \land$$
$$\underbrace{(v_I^{(C_h)} \to -b \geq 0) \land (v_I^{(D)} \to -d_A \geq 0) \land (v_P^{(D)} \to a - f \geq 0)}_{\text{Constraints for } B.}$$

## 4.3 Collusion Resilience

Within a collusion resilient joint strategy, no subgroup of players benefits when deviating from the honest behavior ($\gamma_{\text{cr}}$). We thus need to ensure that all possible deviations of a group of players receive *total* utility less than that obtained by honest behavior. Hence, our formalization in this respect needs to include only the action variables of the players that do *not* belong to the deviating subgroup, as these are the players whose choices are in accordance with the desired joint strategy. For an honest history $h^*$, we formalize collusion resilience as:

$$\bigwedge_{S \subset N} \bigwedge_{t \in \mathcal{T}} \left[\left(\bigwedge_{(h,a) \in \mathcal{H}_t^{N \setminus S}} v_a^h\right) \to \sum_{p \in S} u_p(h^*) \geq \sum_{p \in S} u_p(t)\right]. \quad (\phi_{\text{cr}})$$

*Example 4.4.* Collusion resilience of the Simplified Closing Game is captured by:

$$\left.\begin{array}{r}(v_I^{(C_h)} \to \alpha \geq -a) \land (v_S^{(C_h)} \to \alpha \geq \alpha)\\ \land\ (v_I^{(D)} \to \alpha \geq d_A + \alpha - \varepsilon)\\ \land\ (v_P^{(D)} \to \alpha \geq -a)\end{array}\right\}\text{Subgroup } \{A\}.$$
$$\left.\begin{array}{r}\land\ (v_H^{\emptyset} \to \alpha \geq \alpha) \land (v_{C_h}^{\emptyset} \to \alpha \geq -b)\\ \land\ (v_{C_h}^{\emptyset} \to \alpha \geq \alpha)\\ \land\ (v_D^{\emptyset} \to \alpha \geq -d_A + \alpha)\\ \land\ (v_D^{\emptyset} \to \alpha \geq a - f + \alpha)\end{array}\right\}\text{Subgroup } \{B\}.$$

Since the game has only two players, the two singleton sets of players are the only strict subgroups of players. If we consider the honest history $(C_h, S)$, we set the action variables $v_{C_h}^{\emptyset}, v_S^{(C_h)}, \neg v_H^{\emptyset}$ and $\neg v_I^{(C_h)}$. The resulting formula is satisfied for all possible values of $a, b, \alpha$ and $\varepsilon$ that satisfy the initial conditions. Hence, the Simplified Closing Game is collusion resilient.

*Example 4.5.* We also illustrate how our work disproves collusion resilience using the Simplified Routing Game with the honest history $(S_H, L, L, L, U, U, U, U)$. For the subgroup $\{P_1, P_3\}$ and the terminal history $(S_H, L, L, L, U, S_{SP_1}, U)$, we get the following implication as an instance of $\phi_{\text{cr}}$:

$$v_{S_H}^{\emptyset} \land v_L^{(S_H)} \land v_L^{(S_H,L,L)} \land v_U^{(S_H,L,L,L)} \to 2f \geq 3f - \varepsilon$$

All action variables are set to $\top$ as they are part of the honest history. As $f > \varepsilon > 0$, the formula is not satisfiable. The Simplified Routing Game is thus not collusion resilient as players $P_1$ and $P_3$ can collude profitably.

## 4.4 Practicality

In practical joint strategies, no player has an incentive to deviate in any subgame ($\gamma_{\text{pr}}$). Thus, we need to inspect deviations from a joint strategy in a subgame starting from some history $h$, so we write $\mathcal{H}_{|h,t}^S$ to denote $\mathcal{H}_t^S$ in the subgame $h$.

As already presented in [31], a practical strategy can be constructed iteratively bottom-up: at every internal node, assuming we have a practical strategy (and thus utility) for its subgames, we can choose the action that yields the best utility for the current player. Using this idea, we formalize practicality as follows:

$$\bigwedge_{h \in \mathcal{H} \setminus \mathcal{T}} \bigwedge_{t,r \in \mathcal{T}_{|h}} \left[\bigwedge_{(h',a) \in \mathcal{H}_{|h,t}^N,\, h' \neq \emptyset} v_a^{(h,h')} \land \bigwedge_{(\tilde{h},c) \in \mathcal{H}_{|h,r}^N} v_c^{(h,\tilde{h})}\right] \to u_{P(h)}(h,r) \geq u_{P(h)}(h,t). \quad (\phi_{pr})$$

The formula first quantifies (as a conjunction) over all subgames, represented by non-terminal histories $h$, and then over terminal histories in the subgames starting at $h$. We read the implication as follows: the terminal history $r$ is the one that the practical strategy yields, as on the left-hand side of the implication all of $r$'s actions are asserted. On the right-hand side of the implication it is required that the utility at $r$ of the current player (at history $h$) is better than the utilities from the practical strategies of other children (note that for terminal history $t$ we do not require the first action to be asserted, but only the actions in the child subgame).

*Example 4.6.* We analyze practicality of the Simplified Closing Game. For the subgame starting at history $(C_h)$, we obtain:

$$\underbrace{(v_I^{(C_h)} \rightarrow -b \geq \alpha)}_{\text{With } t=(S) \text{ and } r=(I).} \wedge \underbrace{(v_S^{(C_h)} \rightarrow \alpha \geq -b)}_{\text{With } t=(I) \text{ and } r=(S).} \tag{2}$$

For the honest history $(C_h, S)$, this is satisfiable with $v_S^{(C_h)}$ and $\neg v_I^{(C_h)}$. If $(C_h, I)$ were the honest history, there would be no strategy, as in the subgame starting at $(C_h)$ we should have $-b \geq \alpha$, which contradicts initial conditions.

## 5 AUTOMATED REASONING OF GAME-THEORETIC SECURITY

The first-order formulas of Section 4, combined with Lemma 4.1, provide us with the theoretical foundations for automating security analysis of blockchain protocols presented as EFG trees. We now present our algorithmic advancement on top of satisfiability checking modulo linear real arithmetic (Algorithm 1), allowing us to (dis)prove formulas from Section 4 and hence provide game-theoretic security guarantees. Further, our work yields natural extensions for generating concrete counterexamples whenever security properties are violated (Section 5.2) and infer preconditions to enforce security (Algorithm 3).

For proving the security formulas of Section 4, we focus on automating reasoning about a tuple

$$\Pi = (\Gamma, \mathscr{O}, \mathit{inf}, C, C_{\text{wi}}, C_{\text{weri}}, C_{\text{cr}}, C_{\text{pr}}), \text{ where}$$

- $\Gamma$ is an EFG modeling the protocol of interest;
- $\mathscr{O} \subseteq \mathscr{T}$ is a set of honest histories representing expected behavior;
- $\mathit{inf}$ is a set of infinitesimals occurring in the players' utilities;
- $C$ is the set of initial constraints on variables occurring in player utilities;
- $C_{\text{wi}}$, $C_{\text{weri}}$, $C_{\text{cr}}$, and $C_{\text{pr}}$ are sets of constraints on variables to hold when checking formulas of Section 4, namely weak immunity ($C_{\text{wi}}$), weaker immunity ($C_{\text{weri}}$), collusion resilience ($C_{\text{cr}}$) and practicality ($C_{\text{pr}}$), respectively.

Note that the sets $C$, $C_{\text{wi}}$, $C_{\text{weri}}$, $C_{\text{cr}}$ and $C_{\text{pr}}$ may possibly be empty. For every honest history in $\mathscr{O}$, our work constructs the respective first-order security formula of Section 4 and uses SMT solving for establishing satisfiability of the respective formula: if a model is found, we construct a joint strategy as described in Lemma 4.1. Since security properties must hold for all possible values of the utility variables that adhere to the initial conditions $C$, we universally quantify over all variables and add an implication to account for preconditions. We hence translate game-theoretic analysis from Section 4 into the satisfiability solving of

$$\forall \vec{x}. \left[ \bigwedge_{c \in C \cup C_{\text{sp}}} c[\vec{x}] \rightarrow (\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]) \right], \tag{3}$$

where $\vec{x} = (x_1, x_2, \ldots, x_\ell)$ are all variables appearing in utilities of players and $C_{\text{sp}} \in \{C_{\text{wi}}, C_{\text{weri}}, C_{\text{cr}}, C_{\text{pr}}\}$. By Lemma 4.1, a model $\mathsf{I} \in \mathcal{I}$ of (3) is a joint strategy satisfying the respective security property of Section 4.

We next detail our solution towards solving (3), yielding our automated reasoning approach to prove game-theoretic security in Algorithm 1.

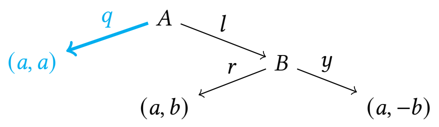


**Figure 3: EFG Splits$_{\text{wi}}$ necessitating case splits during reasoning. We assume $a > 0$.**

### 5.1 Security Reasoning with Case Splitting

We note that formula (3) is too restrictive, as next illustrated.

*Example 5.1 (Splits$_{wi}$).* Consider the EFG of Figure 3 with $N = \{A, B\}$ and honest history $(q)$, where $a > 0$. We aim to find a weak immune strategy for this game. Clearly, $A$ must take action $q$, but if $A$ deviates, $B$ must have non-negative utility. The action of $B$ depends on $b$: if $b > 0$, $B$ should choose $r$; if $b < 0$, $y$ should be chosen; and otherwise, either choice is possible. The game is therefore weak immune for history $(q)$, but requires different strategies for different cases.

Example 5.1 shows that during security analysis, we may need to consider several different orderings of linear terms within utilities. Such *case splits* turn out to be also necessary for real-world protocols, such as the Closing Game [31]. In order to account for possible case splits, we modify (3) and introduce preconditions to order terms. To this end, we compute the set $T_u$ of linear terms appearing in the constructed formulas; for example, for collusion resilience we have:

$$T_u = \left\{ \sum_{p \in S} u_p(t) \mid S \subset N, t \in \mathscr{T} \right\}. \tag{4}$$

We then consider all consistent *total orders* $\preceq$ over $T_u$. As we must find models for all such orders $\preceq$, we reduce solving (3) to solving

$$\boxed{\begin{array}{l} \forall\ (\preceq, T_u) \text{ total order. } \exists\ \mathsf{I} \in \mathcal{I}. \\ \mathsf{I}\left(\forall \vec{x}. \displaystyle\bigwedge_{c \in \preceq \cup C \cup C_{\text{sp}}} c[\vec{x}] \rightarrow (\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}])\right) = \top, \end{array}} \tag{5}$$

where quantification over total orders $\preceq$ happens outside SMT solving (see Remark 1) and $\preceq$ is interpreted as the set of ordering constraints over elements of $T_u$, representing the total order it yields. We conclude the following result.

Theorem 5.2 (Game-Theoretic Security). *Let $\Gamma = (N, G)$ be an EFG with honest history $h^*$, the formula* (5) *for $sp \in \{wi, weri, cr, pr\}$ is equivalent to its game-theoretic analog:*

$$\forall \vec{x}.\ \forall c \in C \cup C_{sp}.\ c[\vec{x}] \rightarrow \exists\, \sigma \in \mathscr{S}.\ H(\sigma) = h^* \wedge\ \gamma_{sp}(\sigma)[\vec{x}], \tag{6}$$

*where $\vec{x} = (x_1, \ldots, x_\ell)$ are the variables occurring in the utility terms (interpreted over reals), and $C$ and $C_{sp}$ are finite sets of linear preconditions on $\vec{x}$.*

Note that every model I of (5) translates to a strategy $\sigma$ in (6) and vice versa, by letting $\sigma(h) = a$ iff $\mathrm{I}(v_a^h) = \top$. The above theorem yields the following result.

Corollary 5.3 (Soundness and Completeness of Encoding). *The encoding of the game-theoretic properties weak immunity, weaker immunity, collusion resilience and practicality using the formulas $\phi_{strat}$, $\phi_{hist}$ together with $\phi_{wi}$, $\phi_{weri}$, $\phi_{cr}$ and $\phi_{pr}$, respectively, including case splits* (6)*, is sound and complete.*

Note that our first-order formulas encoding game-theoretic security properties are expressed in the decidable theory of first-order linear real arithmetic [6], [16]. Since the number of total orders in (5) is finite, our formalization is *decidable*.

Remark 1. *For efficient handling of* (5)*, we compute* unsatisfiable (unsat) cores *to detect which linear terms require case splitting. We introduce labels for inequalities appearing in the formula $\phi_{sp}$ — one of $\phi_{wi}$, $\phi_{weri}$, $\phi_{cr}$ or $\phi_{pr}$. For example, if $\phi_{sp}$ contains $x < y$, we introduce a label $\ell_{(x,y)}$ as a new Boolean variable*[2]*, replace the inequality with the implication $\ell_{(x,y)} \to x < y$, and add $\ell_{(x,y)}$ to our encoding. The weak immunity formula $\phi_{wi}$ then becomes*

$$\bigwedge_{p\in N}\bigwedge_{t\in\mathscr{T}}\left(\left[\bigwedge_{(h,a)\in\mathscr{H}_t^p} v_a^h\right] \to \ell_{(u_p(t),0)} \to u_p(t) \geq 0\right) \wedge \ell_{(u_p(t),0)}$$

*and similarly for $\phi_{weri}$, $\phi_{cr}$ and $\phi_{pr}$ (see [4]).If unsatisfiability of the formula is established, we extract an unsat core composed of our labels and retrieve candidate linear terms for case splits. Example 5.1 yields an unsat core composed of labels $\ell_{0,-b}$ and $\ell_{b,0}$, from which we conclude that the following three cases need to be analyzed: $b > 0$, $b = 0$ and $b < 0$. Yet, even for a small number of linear terms, there might be a very large number of possible orderings $\leq$ between terms to be considered. The number of $\leq$ can however be significantly reduced by making two observations: (i) orderings $\leq$ must be compatible with $C$ and $C_{sp}$; and (ii) orderings must be consistent with real arithmetic. For example, with the set of linear terms $a, -a, 0$, we should consider $a > 0 > -a$, $-a > 0 > a$, and $a = -a = 0$, but not e.g. $a > -a > 0$ or $a = 0 > -a$.*

Based on the above, Algorithm 1 summarizes our reasoning approach for proving game-theoretic security properties based on our first-order formalization with case splits. Given an input Π, Algorithm 1 establishes satisfiability of a game-theoretic property (cf. Lemma 4.1), by using an auxiliary SMT solver A to track (remaining) possible terms orderings. If at any point A reports unsatisfiability, there are no more possible orderings $\leq$. Constraints $C$ and $C_{\mathrm{sp}}$ are added immediately to A. To get an ordering, we ask A for a model — which maps real variables to numeric values — from which we infer an ordering $\leq$ on linear terms over variables (`EvaluateModel`). For example, if the model is $a = 1, b = 2, c = 3$, we obtain the ordering on linear terms $2c - b > a + b = c > 0 > a - c$. If we find a strategy for a case split, we add a *conflict clause* to ensure we do not consider the same ordering twice.

We note that in Algorithm 1 the only properties considered are weak(er) immunity, collusion resilience and practicality; these properties are enough to enforce game-theoretic security (Definition 3.9).

[2]We actually use *two* labelling Boolean variables in our implementation, one each for the infinitesimal and real components of the inequality.

**Algorithm 1:** Game-Theoretic Security Reasoning

```
input  : an input instance
         $\Pi = (\Gamma, \mathscr{O}, inf, C, C_{\mathrm{wi}}, C_{\mathrm{weri}}, C_{\mathrm{cr}}, C_{\mathrm{pr}})$, an honest
         history $h_o \in \mathscr{O}$ and the name of a security
         property sp $\in \{\mathrm{wi}, \mathrm{weri}, \mathrm{cr}, \mathrm{pr}\}$
output : $\top$ if $\phi_{\mathrm{sp}}$ is satisfiable in $\Pi$, $\bot$ otherwise
S ← Solver()
A ← Solver()
AddConstraints (S, ComputeStrategyConstraints ($\Gamma$, $h_o$))
AddConstraints (A, $C \cup C_{sp}$)
$T \leftarrow \emptyset$
while Solve (A) = sat do
    $M \leftarrow$ GetModel (A)
    $O \leftarrow$ {EvaluateModel ($M, x, y$) : $(x, y) \in$ Combinations ($T$)}
    if Check (S, $sp$, $C \cup C_{sp} \cup O$) = sat then
        // found strategy for current ordering, add conflict
        clause AddConstraints (A, $\{\bigvee_{c\in O} \neg c\}$)
    end
    else
        $T' \leftarrow \emptyset$
        foreach $\ell_{x,y} \in$ GetUnsatCore(S) do
            $T' \leftarrow T' \cup \{t : t \in \{x, y\}, t \notin T\}$
        if $T' = \emptyset$ then // no new expressions, considered every case
            return $\bot$
        end
        $T \leftarrow T \cup T'$
    end
end
return $\top$
```

However, Algorithm 1 can be applied to any game-theoretic formula about an EFG as long as this formula can be expressed as a first-order linear inequality over two computable functions $f, g$, with utility function $u$, set of players $N$ and set of strategies $\mathscr{S}$, that is $f(u, N, \mathscr{S}) \leq g(u, N, \mathscr{S})$. Our work yields a generic way to translate such game-theoretic formulas to SMT formulas, as showcased in our proofs in [4].

## 5.2 Generating Counterexamples to Security

In case there is no joint strategy fulfilling the security property $\phi_{\mathrm{sp}}$ within Algorithm 1, our work provides automated formal analysis explaining why the conditions of $\phi_{\mathrm{sp}}$ are violated and derives concrete *counterexamples* to security.

***Weak(er) Immunity.*** For the weak(er) immunity property $\phi_{\mathrm{wi}}$, a counterexample is a harmed player $p$ together with a partial strategy of the other players $N - p$ such that — while following the honest history — no matter how $p$ acts, they cannot avoid receiving a real-valued negative utility. We say that a strategy for a player $p$ follows the honest history $h^*$ if at every node appearing in $h^*$, where $p$ is making a choice, the strategy will choose the action in $h^*$.

*Definition 5.4 (Counterexamples of Weak(er) Immunity).* Let $\Gamma$ be an EFG and $h^*$ the considered honest history. A *counterexample to $h^*$ being weak(er) immune* is a player $p$ together with a partial strategy $s_{N-p} \subseteq \tau_{N-p} \in \mathscr{S}_{N-p}$ of the other players such that $s_{N-p}$ extended by any strategy $\sigma_p$ of player $p$, which follows the honest history $h^*$, yields a terminal history $t$ and it is *minimal* with that property. Further, we have

$$\forall \sigma_p \in \mathscr{S}_p \ (\forall (h,a) \in \mathscr{H}^p_{h^*}.\ \sigma_p(h) = a) \rightarrow u_p(t) < 0 \tag{7}$$

for weak immunity (wi), and respectively

$$\forall \sigma_p \in \mathscr{S}_p \ (\forall (h,a) \in \mathscr{H}^p_{h^*}.\ \sigma_p(h) = a) \rightarrow \mathrm{real}(u_p(t)) < 0 \tag{8}$$

for weaker immunity (weri).

*Minimality* of the partial strategy $s_{N-p}$ states that, if any information point $s_{N-p}(h) = a$ is removed, there exists a strategy $\sigma_p$ of player $p$ such that the tuple $(\sigma_p, s'_{N-p})$ does not yield a terminal history, where the $s'_{N-p}$ is $s_{N-p}$ without action $a$. That means, when following only actions of $(\sigma_p, s'_{N-p})$, we get stuck at an internal node of the tree.

*Example 5.5.* The Simplified Closing Game of Figure 1 with honest history $(C_h, S)$ is not weak immune. The counterexample is the player $A$ and the partial strategy for player $B$ taking action $I$ after $C_h$ (and therefore making the utility of $A$ negative). The partial strategy is minimal, because if we remove the choice of action $I$ for player $B$, after $A$ chooses action $C_h$, we get stuck at the internal node of the game (where $B$ is making a choice), rather than ending in a leaf node, where the utilities of players are known. Note that the partial strategy says nothing about the tree in the subgame after $D$, as we only consider strategies where player $A$ chooses $C_h$ at the root node, to follow the honest history.

***Collusion Resilience.*** A counterexample to collusion resilience $\phi_{\mathrm{cr}}$ consists of a group of deviating players $S$ and their partial strategy $s_S \in \mathscr{S}$ leading to better-than-honest joint utility for $S$, no matter how the other players $N - S$ react – while following the honest history.

*Definition 5.6 (Counterexamples Collusion Resilience).* Let $\Gamma$ be an EFG and $h^*$ the considered honest history. A *counterexample to $h^*$ being collusion resilient (cr)* is a set of deviating players $S$ together with their partial strategy $s_S \subseteq \tau_S \in \mathscr{S}_S$ such that $s_S$ extended by any strategy $\sigma_{N-S}$ of players $N - S$, which follows the honest history $h^*$, yields a terminal history $t$ and it is minimal with that property. Further,

$$\forall \sigma_{N-S} \in \mathscr{S}_{N-S}\ . \tag{9}$$
$$(\forall (h,a) \in \mathscr{H}^{N-S}_{h^*}.\ \sigma_{N-S}(h) = a) \rightarrow \sum_{p\in S} u_p(t) > \sum_{p\in S} u_p(h^*)\ .$$

The minimality of $s_S$ is similar to the minimality of the partial strategy for weak(er) immunity.

*Example 5.7.* In the Simplified Routing Game of Figure 2, the terminal history $(S_H, L, L, L, L, U, S_{SP_1}, U)$ results in a strictly better outcome for the subgroup $\{P_1, P_3, B\}$ than the honest history, modeling the Wormhole attack (see Section 2). While choosing the honest actions $L$ and $L$, the other players $A$ and $P_2$ are powerless in this scenario.

***Practicality.*** Intuitively, a counterexample to practicality of $h^*$ has to provide a reason why a rational player would not follow $h^*$. That is, somewhere along $h^*$, assume after a prefix $h$, there exists an action $a$ which promises the current player $P(h)$ a strictly better utility than $h^*$. However, a "promised" utility has to be one that results from a practical history $t$ in the subgame after $(h, a)$, otherwise it would not be an actual counterexample to the practicality of $h^*$. Therefore, we define a counterexample to practicality as follows.

*Definition 5.8 (Counterexamples Practicality).* For an EFG $\Gamma$ and an honest history $h^*$, a *counterexample $c$ to the practicality (pr) of $h^*$* is a terminal history $c = (h, a, t)$, where $h$ is the maximal common prefix with $h^*$, $a \in A(h)$ is a possible action after history $h$, and $t$ is a practical terminal history in $\Gamma_{|(h,a)}$ such that $u_{P(h)}(h^*) < u_{P(h)}(c)$.

*Example 5.9.* The Simplified Closing Game of Figure 1 with honest history $(H)$ is not practical. The counterexample is the terminal history $(C_h, S)$, that deviates from the honest history already at the root of the game tree. Choosing the action $S$ in the subgame after history $(C_h)$ is practical, as it yields a better utility $(\alpha)$ for player $B$ than action $I$ (which gives $-b$). At the root it is therefore better for player $A$ to choose action $C_h$ over the honest action $H$, as $C_h$ gives utility $\alpha$, which is strictly better than $\alpha - \varepsilon$ given by $H$. The history $(C_h, S)$ is therefore a valid counterexample to practicality of honest history $(H)$.

***Correctness of Counterexamples.*** The various counterexamples to security properties, as introduced above, are evidence for a violated security property, as stated below.

Theorem 5.10 (Counterexamples to Security). *For an EFG $\Gamma$ and an honest history $h^*$, there exists a counterexample to wi, weri, cr or pr of $h^*$ according to Definition 5.4–Definition 5.8 iff $h^*$ is not weak(er) immune, collusion resilient or practical, respectively.*

***Computing Counterexamples to $\phi_{wi}$, $\phi_{weri}$ and $\phi_{cr}$.*** Within our work, we find counterexamples to (violated) $\phi_{\mathrm{wi}}$, $\phi_{\mathrm{weri}}$ and $\phi_{\mathrm{cr}}$ using an approach similar to case splitting over term orderings. We first amend formulas $\phi_{\mathrm{wi}}$, $\phi_{\mathrm{weri}}$ and $\phi_{\mathrm{cr}}$ with suitable labels. This allows us to detect at which histories the checked property is violated and for which player(s), by inspecting labels in the reasoning core. Each history reflects one choice in the counterexample strategy. For the purpose of counterexample generation, the formula for weak immunity $\phi_{\mathrm{wi}}$ is then rewritten as follows:

$$\bigwedge_{p\in N} \bigwedge_{t\in\mathscr{T}} \left( \ell_{p,t} \rightarrow \left[ \bigwedge_{(h,a)\in\mathscr{H}^p_t} v^h_a \right] \rightarrow u_p(t) \geq 0 \right) \wedge\ \ell_{p,t}\ .$$

Formulas $\phi_{\mathrm{weri}}$ and $\phi_{\mathrm{cr}}$ are adjusted analogously and can be found in [4].

We generate counterexamples to $\phi_{\mathrm{wi}}$, $\phi_{\mathrm{weri}}$ and $\phi_{\mathrm{cr}}$ by enumerating minimal unsat cores [20]. From these unsat cores, (groups of) players and partial strategies are identified, yielding counterexamples to the respective security property. Each unsat core represents a counterexample; thus, by listing and interpreting all unsat cores, we generate all counterexamples to $\phi_{\mathrm{wi}}$, $\phi_{\mathrm{weri}}$ and $\phi_{\mathrm{cr}}$.

***Computing Counterexamples to $\phi_{pr}$*** Generating counterexamples to $\phi_{pr}$ is summarized in Algorithm 2, requiring a different approach than for $\phi_{\mathrm{wi}}$, $\phi_{\mathrm{weri}}$ and $\phi_{\mathrm{cr}}$.

**Algorithm 2:** Counterexamples to Practicality

```
input  : an input instance
         $\Pi = (\Gamma, \mathcal{O}, inf, C, C_{\mathrm{wi}}, C_{\mathrm{weri}}, C_{\mathrm{cr}}, C_{\mathrm{pr}})$, an honest
         history $h_o \in \mathcal{O}$ and the case *case* for which
         practicality failed
output : the set of all counterexamples $CE$ to the
         practicality of $h_o$ in a subcase of *case*, together
         with that subcase *subcase*
1  CE ← ∅
2  GT ← Γ
3  ho ← $h_o$
4  subgame ← ()
5  subcase ← case
6  while ho ¬pr ∧ GT ≠ leaf do
7      prStrat ← $\{h' | h'\text{pr in GT} \wedge u_{P(h)}(\text{ho}) < u_{P(h)}(h')\}$
8      // $h$ is the maximal common prefix of $h'$ with ho
9      if prStrat = ∅ then
10         // in subgame no more counterexamples
11         $h$ ← first action of ho
12     end
13     else
14         $h$ ← shortest prefix $h$ in prStrat
15     end
16     subcase.RefineSubcase ()
17     // if further case split was done in line 7
18     foreach $c$ ∈ prStrat do
19         CE.Add ( (subgame, $c$) )
20     end
21     ho ← ho − $h$
22     subgame ← subgame + $h$
23     GT.Remove (direct subtrees of $h$ that occur in prStrat)
24     GT ← $\mathrm{GT}_{|h}$
25 end
26 return CE, subcase
```

Algorithm 2 takes as input a game $\Gamma$, an honest history $h_o$ and the *case* of term orderings for which practicality analysis failed. Algorithm 2 returns a set of counterexamples (CE) to $\phi_{pr}$, that is, a set of terminal histories, and a subcase of the initial *case*. Note that the set CE contains all the counterexamples to $\phi_{pr}$ in the given refined problematic case. In each iteration of the while-loop of Algorithm 2 (lines 6–25), all practical histories that yield a strictly better utility for the deviator are listed, then the game tree is cut such that new counterexamples can be revealed (line 7). The variables GT (current game tree), $h$ (guidance on what to cut) and *subgame* (subgame to be considered) keep track of the cutting and ensure that the correct terminal histories are added to CE. Additionally, it might be necessary to further specify the ordering of the utility terms, which is considered in line 16 of Algorithm 2.

Based on the above considerations, all counterexamples to practicality for a specific term ordering are thus generated via Algorithm 2, as proven next.

Theorem 5.11 (Correctness of Algorithm 2). *Consider the output* (CE, subcase) *of Algorithm 2. Then, the set* CE *computed by Algorithm 2 contains exactly all counterexamples to* $h^*$ *being practical in* subcase.

Proof. (i) We first prove that $c \in \mathrm{CE}$ implies $c$ is a counterexample in subcase. Let the honest history be $h^*$ and $c \in \mathrm{CE}$. Assuming subcase, we show that $c$ is of the form $(\text{prefix}, a, h'')$, where prefix is the maximal common prefix of $c$ with $h^*$, $a$ an action such that $u_{P(\text{prefix})}(h^*) < u_{P(\text{prefix})}(c)$ and $h''$ practical in $\Gamma_{|(\text{prefix},a)}$.

By Algorithm 2 (line 7), when $c = (\text{subgame}, h')$, we have that $h'$ is practical for some ho, *subcase* and GT. Additionally, $u^{\mathrm{GT}}_{P(h)}(\text{ho}) < u^{\mathrm{GT}}_{P(h)}(h')$, where $h$ is the maximal common prefix of $h'$ with ho. Note that in every iteration of the loop ho is a suffix of $h^*$, as only prefixes $h$ of $h^*$ are cut off (line 11, 14, and 21). Further, subcase is a subcase of *subcase*, since *subcase* is only refined in the algorithm (line 16). The game GT is the subgame of $\Gamma$ after subgame, possibly without some already removed direct subtrees. This holds as we always consider subgames after $h$ (line 24), which are pieces of $h^*$ and are collected in *subgame* (line 22). The direct subtree removal happens in line 23 of Algorithm 2. We hence conclude that $u_{P(\text{subgame},h)}(h^*) = u^{\mathrm{GT}}_{P(h)}(\text{ho}) < u^{\mathrm{GT}}_{P(h)}(h') = u_{P(\text{subgame},h)}(c)$ in *subcase*. Additionally, $c$ is a terminal history of the input game $\Gamma$. The property left to show is that $h' = (a, h'')$, such that $h''$ is practical in $\Gamma_{|(\text{subgame},a)}$. Note that only direct subtrees of $\Gamma_{|\text{subgame}}$ have been removed in GT, hence $\mathrm{GT}_{|(a)} = \Gamma_{|(\text{subgame},a)}$. Since $h' = (a, h'')$ is practical in GT, $h''$ has to be practical in $\mathrm{GT}_{|a} = \Gamma_{|(\text{subgame},a)}$ in *subcase*. Therefore, $c$ is a counterexample in *subcase* and thus also in subcase, concluding direction (i) of the proof.

(ii) We next show the reverse direction of (i); that is, $c$ is a counterexample in subcase implies $c \in \mathrm{CE}$. Let $C$ be the set of all counterexamples in the case subcase. As subcase does not necessarily yield a total order on the utility terms, we fix an arbitrary total order $\leq_s$ on the utility terms which is compatible with subcase. We now order elements $c \in C$ according to

(1) length of the common prefix $h$ of $c$ with the honest history $h^*$ (shortest first) and within this group into
(2) subgroups according to the value $u_{P(h)}(c)$ (decreasingly). Within these subgroups, the order does not matter.

Towards a contradiction, we assume $c \in C$ is the first (according to the ordering above) that is not in CE. Let $c = (h, a, t)$ and $u_{P(h)}(c) =: u$, such that $t$ is practical in $\Gamma_{|(h,a)}$, $u > u_{P(h)}(h^*)$ and $h$ the maximal common prefix with $h^*$. We distinguish between the following possible cases.

*Case 1:* $(a, t)$ is not practical in $\Gamma_{|h}$ in total order $\leq_s$. Thus, there is a $(a', t') \in \mathscr{H}_{|h}$ such that $u' := u_{P(h)}(h, a', t') >_s u$, and $t'$ practical in $\Gamma_{|(h,a')}$ which is a counterexample to $(a, t)$ being practical in $\leq_s$. This also yields a counterexample $c' = (h, a', t')$ to $c$ being practical in $\leq_s$. It also is another counterexample to $h^*$ being practical ($u' >_s u > u_{(P(h))}(h^*)$) in $\leq_s$ and thus subcase. Therefore $c' \in C$ and it occurs in the same group but in a strictly earlier subgroup than $c$.

Thus, by our assumption that $c$ is the first to not appear in CE, we get $c' \in \mathrm{CE}$.

In Algorithm 2, every $c \in \mathrm{CE}$ occurs once in prStrat[3] as the element with the shortest common prefix with $h^*$. This holds as we always remove at least one direct subtree or move further along $h^*$. Let us now consider the last occurrence of $c'$ in prStrat. In this iteration, the tree GT will be cut to at most $\Gamma_{|h} - \{a' : c' = (h, a', t')\}$. Since all counterexamples $c''$ with the same prefix $h$ as $c$ and with greater utilities $u_{P(h)}(c'') >_s u_{P(h)}(c)$ have been found in CE, the corresponding branches are removed from GT in the following iterations of Algorithm 2 as well.

We consider now under which condition $(a, t)$ from $c = (h, a, t)$ becomes practical in a partial tree of $\Gamma_{|h}$ in subcase: since t is practical in $\Gamma_{|(h,a)}$ it suffices if $u_{P(h)}(h, a, t) \geq u_{P(h)}(h, a'', t'')$ in subcase, for all $a'' \in A(h)$, $t''$ practical in $\Gamma_{|(h,a)}$. Note that each such $(a'', t'')$ that violates this property yields a counterexample $c'' = (h, a'', t'')$ to $h^*$'s practicality in subcase with a utility strictly better or incomparable to $c$. We observe that incomparability at this step in the algorithm could not have happened, as it would have caused a further case split (line 16) which contradicts the fact that subcase is the output value of the case split. Let us thus consider the case of strictly better utility: Since $\leq_s$ is compatible with subcase, we know $u_{P(h)}(c'') >_s u_{P(h)}(c)$ implies $u_{P(h)}(c'') > u_{P(h)}(c)$ in subcase. Thus, all such $(a'', t'')$ have at this point already been removed from GT.

Therefore, $(a, t)$ of $c = (h, a, t)$ is practical in this tree, implying that $(a, t)$ has to appear in prStrat in the next iteration and hence also $c \in CE$. This contradicts the assumption of *Case 1*.
*Case 2:* $(a, t)$ is practical in $\Gamma_{|h}$ in $\leq_s$, yielding two further cases.
*Case 2.1:* $c = (h, a, t)$ is in the first group $C$, that is, there is no counterexample with a shorter prefix. In this case, $c = (h, a, t)$ has to be practical in $\Gamma$ in $\leq_s$. By assuming the contrary, there exists a counterexample $c' = (h', a', t')$, where $h'$ is a prefix of $c$, $t'$ is practical in $\Gamma_{|(h',a')}$ in $\leq_s$, and $u_{P(h')}(c) <_s u_{P(h')}(c')$. Then, $h'$ cannot be a prefix of $h$, as this contradicts $c$ being in the first group of $C$. Further, $h$ can not be a prefix of $h'$, since this contradicts the fact that $(a, t)$ is practical in $\Gamma_{|h}$ in $\leq_s$. Therefore, $c$ is practical in $\Gamma$ in $\leq_s$, implying however that $c$ has to occur in the first iteration of the loop in prStrat (being practical in subcase). Another case is not possible (similar to Case 1) as a further case split of subcase did not happen.
*Case 2.2:* $c = (h, a, t)$ is not in the first group in $C$. Let then $h'$ be the maximal common prefix with $h^*$ in the group directly before $c$. All the counterexamples $c'$ of this group have $u_{P(h')}(c') > u_{P(h')}(h^*)$. Similar to *Case 1*, as they all appear in CE, we reach the following tree to be considered: $\Gamma_{|h'}$ minus all the immediate dishonest branches that contain counterexamples. In the subsequent iteration of the algorithm, prStrat has to be empty as there are no more branches that make the condition $u_{P(h')}(c') > u_{P(h')}(h^*)$ true. Therefore, we go to the if-branch in line 9 and consider $\Gamma_{|(h',a^*)}$ in the following iteration. The action $a^*$ is the next action in $h^*$ after $h'$. Note that for the tree $\Gamma_{|(h',a^*)}$, $c$ is in the first group of counterexamples by construction. Therefore, *Case 2.1* applies for the game $\Gamma_{|h',a^*}$, thus counterexample $c$ has to be practical in $\Gamma_{|(h',a^*)}$ and hence has to occur in prStrat in this iteration. This contradicts the assumption that $c \notin \mathrm{CE}$ and overall concludes direction (ii) of the proof. □

[3]only the respective suffix of $c$ occurs since we add the subgame prefix only later to CE.

## 5.3 Inferring Preconditions for Security

In addition to identifying concrete counterexamples (Section 5.2) in case a security property $\phi_{\mathrm{sp}}$ is not satisfied in Algorithm 1, our work can also determine additional assumptions necessary to ensure that property $\phi_{\mathrm{sp}}$ holds. We support such an extended security analysis by iteratively computing preconditions for the input EFG.

*Definition 5.12 (Preconditions to Security).* Given a game $\Gamma$, an honest history $h^*$, a security property sp and initial conditions $C$ and $C_{\mathrm{sp}}$ such that $h^*$ violates sp, we say $\pi$ is a *precondition* if

$$\forall\ (\preceq, T_u) \text{ total order. } \exists\ \mathrm{I} \in \mathcal{I}.$$
$$\mathrm{I}\left(\forall \vec{x}. \bigwedge_{c \in \preceq \cup C \cup C_{\mathrm{sp}} \cup \{\pi\}} c[\vec{x}] \rightarrow \left(\phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}]\right)\right) = \top.$$

We note that in Definition 5.12, the precondition $\pi$ strengthens the initial constraints by making the left-hand side of the above implication satisfiable for less total orders, thus ensuring validity of the above implication for more of them.

In Algorithm 3, we show our adjustment of Algorithm 1, allowing us to generate the weakest precondition under which a security property sp is satisfied. If precondition generation is enabled in our work, the solving routine at line 16 in Algorithm 1 does not terminate; instead, it adds the negation of the ordering in the unsolved case $O$ to preconditions in $C$ (the variable pre in line 20 of Algorithm 3) and restarts the solving routine with the new set of initial constraints. With such an adjustment of Algorithm 1, Algorithm 3 allows us to generate *weakest preconditions* to sp, in the following sense: if $\pi$ and $\psi$ are preconditions, then $\pi$ is *weaker* than $\psi$ if

$$\bigwedge_{c \in \preceq \cup C \cup C_{\mathrm{sp}} \cup \{\pi\}} c[\vec{x}]$$

is satisfiable for more total orders $\preceq$ than

$$\bigwedge_{c \in \preceq \cup C \cup C_{\mathrm{sp}} \cup \{\psi\}} c[\vec{x}].$$

For proving correctness of Algorithm 3, we first prove the following helping lemma.

Lemma 5.13. *[Unique Weakest Precondition] Given a game $\Gamma$, an honest history $h^*$, a security property* sp *and finite sets of initial constraints $C$ and $C_{\mathrm{sp}}$, there exists a unique (modulo equivalence) weakest precondition $\pi$ to make history $h^*$ satisfy* sp.

Proof. Let $\pi$ be a precondition, by Definition 5.12, it holds that

$$\forall\ (\preceq, T_u) \text{ total order. } \exists\ \mathrm{I} \in \mathcal{I}.$$
$$\mathrm{I}\left(\forall \vec{x}. \bigwedge_{c \in \preceq \cup C \cup C_{\mathrm{sp}} \cup \{\pi\}} c[\vec{x}] \rightarrow \left(\phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}]\right)\right) = \top.$$

As noted after Corollary 5.3, the satisfiability of the formula depends only on the finitely many total orders of terms in $T_u$. Thus, any precondition can be weakened to a list of term orderings to be avoided, which are finitely many. The weakest of these is the one

**Algorithm 3:** Generating Weakest Preconditions

```
input : an input instance
        $\Pi = (\Gamma, \mathcal{O}, inf, C, C_{\text{wi}}, C_{\text{weri}}, C_{\text{cr}}, C_{\text{pr}})$, an honest
        history $h_o \in \mathcal{O}$ and the name of a security
        property sp $\in$ {wi, weri, cr, pr}
output: weakest precondition under which $\phi_{\text{sp}}$ is
        satisifiable in $\Pi$
1  S ⟵ Solver()
2  A ⟵ Solver()
3  pre ⟵ $\top$ // the precondition to be constructed
4  $\varphi \longleftarrow \top$ // the conflict clauses to reach the next case
5  AddConstraints (S, ComputeStrategyConstraints ($\Gamma$,
     $h_o$))
6  AddConstraints (A, $C \cup C_{sp}$)
7  $T \longleftarrow \emptyset$
8  while Check ($A, \varphi$) = sat do
9      $M \longleftarrow$ GetModel (A, $\varphi$)
10     $O \longleftarrow$ {EvaluateModel ($M, x, y$) : $(x, y) \in$
         Combinations ($T$)}
11     if Check (S, $sp$, $C \cup C_{sp} \cup$ {pre} $\cup O$) = sat then
12         // found strategy for current ordering, add conflict
             clause
13         $\varphi \longleftarrow \varphi \wedge (\bigvee_{c\in O} \neg c)$
14     end
15     else
16         $T' \longleftarrow \emptyset$
17         foreach $\ell_{x,y} \in$ GetUnsatCore(S) do
18             $T' \longleftarrow T' \cup \{t : t \in \{x, y\}, t \notin T\}$
19         if $T' = \emptyset$ then // no new expressions, case is unsat
20             pre ⟵ pre $\wedge (\bigvee_{c\in O} \neg c)$
21             $\varphi \longleftarrow \varphi \wedge (\bigvee_{c\in O} \neg c)$
22         end
23         $T \longleftarrow T \cup T'$
24     end
25 end
26 return pre
```

that allows precisely all total orderings of terms in $T_u$ that are satisfiable and forbids all others, and is thus unique up to equivalence (as a quantifier-free first-order formula can be expressed in many equivalent ways). □

We next proceed with proving correctness of Algorithm 3.

Theorem 5.14 (Weakest Preconditions – Correctness of Algorithm 3). *Given a game* $\Gamma$*, an honest history* $h^*$ *and a security property* sp *that* $h^*$ *violates, Algorithm 3 generates the weakest precondition* $\pi$ *which makes* $h^*$ *satisfy* sp.

Proof. As Algorithm 3 adjusts Algorithm 1, we only prove that whenever Algorithm 3 terminates (which it does, because we only need to consider finitely many total orders of $T_u$), the generated precondition $\pi$ (i) is indeed a precondition and (ii) is the weakest.

(i) By construction, $\pi$ is a conjunction of negated term orderings:

$$\pi = \left(\bigvee_{c\in \mathrm{O}_1} \neg c\right) \wedge \cdots \wedge \left(\bigvee_{c\in \mathrm{O}_n} \neg c\right).$$

Let us write $\neg O_i$ for $\left(\bigvee_{c\in \mathrm{O}_i} \neg c\right)$. By construction, we leave out only the term orderings for which the security property holds (line 11 in the Algorithm 3), and hence $\pi$ is a precondition.

(ii) Let $\psi$ be the weakest precondition given by Lemma 5.13. Towards a contradiction, assume that $\psi$ is strictly weaker than $\pi$; that is, there exists a total order $\preceq$ such that

$$\bigwedge_{c\in \preceq \cup C \cup C_{\text{sp}} \cup \{\psi\}} c[\vec{x}] \tag{10}$$

is satisfiable and

$$\bigwedge_{c\in \preceq \cup C \cup C_{\text{sp}} \cup \{\pi\}} c[\vec{x}] \tag{11}$$

is unsatisfiable. Let $M$ be a model for $\vec{x}$ that satisfies (10). Since $M$ does not satisfy (11), we know that $M(\pi) = \bot$. Let $O_i$ be the first ordering in the conjunction $\pi$ such that $M(\neg O_i) = \bot$. Consider line 20 of Algorithm 3 at the point when $\neg O_i$ was added to the precondition $\pi$: the else case of the condition of line 11 implies that the security property is not satisfied in the case $O_i$; further, line 19 implies that there are no more comparisons to be added. As such, for every total order $\preceq_i$ that implies the case $O_i$, we have

$$\forall\, \mathsf{I} \in \mathcal{I}.\ \mathsf{I}\left(\forall \vec{x}.\ \bigwedge_{c\in \preceq_i \cup C \cup C_{\text{sp}}} c[\vec{x}] \rightarrow \left(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]\right)\right) = \bot,$$

which is equivalent to

$$\forall\, \mathsf{I} \in \mathcal{I}.\ \exists \vec{x}.\ \bigwedge_{c\in \preceq_i \cup C \cup C_{\text{sp}}} c[\vec{x}] \wedge \neg\, \mathsf{I}\left(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]\right). \tag{12}$$

The inequalities in $\phi_{\text{sp}}[\vec{x}]$ only depend on the total order $\preceq_i$ and not on the actual values of $\vec{x}$. Thus,

$$\exists \vec{x}.\ \bigwedge_{c\in \preceq_i \cup C \cup C_{\text{sp}}} c[\vec{x}] \wedge \neg\, \mathsf{I}\left(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]\right)$$

implies

$$\forall \vec{x}.\ \bigwedge_{c\in \preceq_i \cup C \cup C_{\text{sp}}} c[\vec{x}] \rightarrow \neg\, \mathsf{I}\left(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]\right).$$

Using (12), we get

$$\forall\, \mathsf{I} \in \mathcal{I}.\ \forall \vec{x}.\ \bigwedge_{c\in \preceq_i \cup C \cup C_{\text{sp}}} c[\vec{x}] \rightarrow \neg\, \mathsf{I}\left(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]\right). \tag{13}$$

Consider now $\preceq_M$ the total order on $T_u$ induced by the model $M$. Since $M(\neg O_i) = \bot$, we know that $M$ satisfies $O_i$, so $\preceq_M$ implies $O_i$ as well and (13) holds for $\preceq_M$ too. As $\psi$ is a precondition, by using Definition 5.12 with $\preceq_M$, we obtain $\mathsf{I}' \in \mathcal{I}$ such that

$$\forall \vec{x}.\ \bigwedge_{c\in \preceq_M \cup C \cup C_{\text{sp}} \cup \{\psi\}} c[\vec{x}] \rightarrow \mathsf{I}'\left(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]\right) \tag{14}$$

holds. Using (13) with total order $\preceq_M$ and $\mathsf{I}' \in \mathcal{I}$, we obtain

$$\forall \vec{x}.\ \bigwedge_{c\in \preceq_M \cup C \cup C_{\text{sp}}} c[\vec{x}] \rightarrow \neg\, \mathsf{I}'\left(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]\right). \tag{15}$$

With the model $M$ for $\vec{x}$, the antecedent of the implication in (14) holds as $M$ satisfies (10) (and thus satisfies the constraints in $C$, $C_{\text{sp}}$ and $\psi$). Thus, $\mathsf{I}'\ (\phi_{\text{strat}} \land \phi_{\text{hist}} \land \phi_{\text{sp}}[M(\vec{x})])$ holds. With the model $M$ for $\vec{x}$, the antecedent of the implication in (15) is also valid, yielding $\neg\,\mathsf{I}'\ (\phi_{\text{strat}} \land \phi_{\text{hist}} \land \phi_{\text{sp}}[M(\vec{x})])$. We thus obtained a contradiction, concluding the proof. □

*Example 5.15.* The weak immunity property $\phi_{\text{wi}}$ of the game from Example 5.1 for honest history $(l, r)$ is not satisfied if $b < 0$. Our work identifies this case and reports that there is no weak immune joint strategy given that $b < 0$. The solving routine of Algorithm 1 is restarted with $b \geq 0$ added to $C$ in Algorithm 3, triggering the satisfiability of the such revised weak immunity property $\phi_{\text{wi}}$. In this case, Algorithm 3 returns one additional precondition, namely $b \geq 0$.

We conclude this section by noting that finding sufficient (non-trivial) preconditions to satisfy a security property $\phi_{\text{sp}}$ is not always possible with Algorithm 3: if the existing preconditions $C$ imply the ordering $O$ for which there is no adequate joint strategy, the security property $\phi_{\text{sp}}$ cannot be satisfied by adding more assumptions about the EFG utilities. In this case, the weakest precondition returned by Algorithm 3 is $\perp$.

## 6 IMPLEMENTATION AND EXPERIMENTAL EVALUATION

***Implementation.*** We implemented our game-theoretic security analysis in the new CheckMate tool[4], written in Python and using Z3 [7] as the underlying SMT solver of Algorithm 1. The input for CheckMate is a JSON [14] encoding of an input instance $\Pi$ of Algorithm 1, which is also the current representation language for the EFG. The instance $\Pi$ is parsed into an internal representation, which is used to construct SMT constraints for (any subset of) the security properties presented in Section 4. CheckMate then executes Algorithm 1 for each property, logging intermediate results such as case splits. CheckMate outputs a joint strategy if a property is satisfied, or a list of counterexamples otherwise; in the latter case, CheckMate also produces preconditions (Section 5.3). In addition, CheckMate supports a *verification* mode to check whether a joint strategy satisfies a security property.

CheckMate only supports EFGs with finite game trees. However, EFGs are known to have the capacity to model many protocols. Further, every step in the game can model an arbitrary long time in the protocol (e.g. the "ignore" actions model unlimited time). An advantage of the trees being finite is that the game-theoretic properties will only quantify over finite sets, implying that the respective security properties can be translated to first-order linear real arithmetic (as shown in Section 4).

***Benchmarks.*** Among other benchmarks, we evaluated CheckMate using the examples of Section 2 as well as real-world blockchain protocols from Bitcoin's Lightning Network [30]. These examples are listed as the last four entries of Table 1. Namely, we used the complete model of the Closing Game of [31] and the full model of the routing phase corresponding to the Routing Game of [31] with three players. The honest behavior of the full routing game is similar to the Simplified Routing Game, but players have many more actions available, such as sharing the secrets or varying the way they lock funds. The full Routing Game for five (and more) players is a task for future work.

As the modeling process of real-world protocols as EFGs is an intricate and time-consuming process, representing a challenge on its own, we so far have a restricted but representative benchmark set, showcasing the logical expressivity provided by CheckMate to efficiently (dis)prove blockchain security. We are not aware of other efforts providing practical challenges for evaluating formal methods in support of blockchain security. We believe our EFG examples provide an initial set of examples to be further used in verifying blockchain security.

***Experimental Results.*** Table 1 summarizes our experimental results, using an Apple M1 Pro CPU with 10 cores and 32 GB of RAM. The sizes of our EFG examples in terms of nodes and players are listed in columns 2–3. For each honest history (column 4), Table 1 displays CheckMate's results for game-theoretic security. We also report execution times for complete analysis of all properties, without counterexample generation. In column 6 (Calls), the number of SMT calls within CheckMate is listed. Both protocol benchmarks (the Closing Game and the 3-Player Routing Game) hint that even for bigger game trees, the number of case splits is relatively low.

Table 2 displays the number of counterexamples found for each violated property, as well as a precondition strong enough to make the respective property true. We write $\perp$ to indicate that no precondition (except $\perp$) is strong enough to satisfy the property. Thus, the problem is inherent to the game and is not a matter of the variable values. Due to its size, we do not state the nontrivial precondition $\pi$ of the Pirate Game, generated by CheckMate in 37 seconds.

***Experimental Analysis.*** CheckMate successfully analyzes the games of Section 2 and correctly identifies the Wormhole attack as one of the 16 counterexamples to collusion resilience in the Simplified Routing Game. Applied to the Closing Game, CheckMate terminates after about 17 seconds for each honest history and correctly identifies required case splits for history $(C_h, S)$. The Routing Game is not weak immune, but *weaker* immune: honest players will not lose "real" resources, but may suffer opportunity cost. The 3-player Routing Game has 21,688 nodes, compared to 221 nodes for the Closing Game.

Our experiments demonstrate the usability and scalability of CheckMate. For example, for analyzing real-world applications (closing and routing phases in the last two lines of Table 1), CheckMate enabled the automation of reasoning about trillions of game strategies and thousands of game nodes. We are not aware of other automated reasoning approaches handling such and similar EFGs.

The mostly trivial preconditions in Table 2 are not surprising, since we asserted constraints we were aware of as initial conditions. For the Closing Game and history $(C_h, S)$, for example, we inherited initial constraints $a, b \geq f$ for weak immunity and $c \neq p_A$ for practicality from [31]. When removing those constraints, CheckMate finds a precondition equivalent to $a, b \geq f$ for weak immunity and one equivalent to $c \neq p_A \lor b - p_A + d_B = f$ for practicality. Thus, CheckMate provides a less restrictive but still sufficient precondition than [31].

[4]Our tool is available at https://github.com/apre-group/checkmate.

| Game | Nodes | Players | History | Security | Calls | Time |
|---|---|---|---|---|---|---|
| Splits$_{\text{wi}}$ | 5 | 2 | $(q)$ | ✓ | 8 | 0.38 |
| Splits$_{\text{cr}}$* | 5 | 2 | $(n)$ | ✓ | 10 | 0.76 |
| Market Entry* | 5 | 2 | $(e, i)$ | ✗(wi,weri) | 6 | 0.32 |
| Pirate* | 52 | 4 | $(y, n, n, n, y, y)$ | ✗(wi,weri,cr,pr) | 11 | 1.88 |
| Simplified Closing | 8 | 2 | $(H)$ | ✗(pr) | 5 | 0.36 |
| | | | $(C_h, S)$ | ✗(wi, weri) | 6 | 0.36 |
| Simplified Routing | 17 | 5 | $(S_H, L, L, L, L, U, U, U, U)$ | ✗(wi,cr) | 6 | 0.57 |
| Closing | 221 | 2 | $(H)$ | ✗(pr) | 5 | 16.4 |
| | | | $(C_h, S)$ | ✓ | 6 | 17.8 |
| 3-player Routing | 21,688 | 3 | $(S_H, L, L, U, U)$ | ✗(wi,cr,pr) | 149 | 1222 |

**Table 1: CheckMate results, time in seconds. ✓means that all security properties are satisfied. ✗(sp) indicates that the history is not secure as property "sp" failed. The games marked with (*) are introduced in [4].**

| Game | Property | CE | PC |
|---|---|---|---|
| Market Entry | wi | 1 | $\perp$ |
| $(e, i)$ | weri | 1 | $\perp$ |
| Pirate | wi | 141 | $\perp$ |
| $(y, n, n, n, y, y)$ | weri | 141 | $\perp$ |
| | cr | >80 | $\perp$ |
| | pr | 1 | $\pi$ |
| Simplified Closing $(H)$ | pr | 1 | $\perp$ |
| Simplified Closing $(C_h, S)$ | wi | 1 | $\perp$ |
| | weri | 1 | $\perp$ |
| Simplified Routing | wi | 9 | $\perp$ |
| $(S_H, L, L, L, L, U, U, U, U)$ | cr | 16 | $\perp$ |
| Closing $(H)$ | pr | 5 | $\perp$ |
| 3-player Routing | wi | 41 | $\perp$ |
| $(S_H, L, L, U, U)$ | cr | 2 | $\perp$ |
| | pr | >0* | TO |

**Table 2: Counterexample (CE) and Precondition (PC) analysis provided by CheckMate. >$N$ indicates that we found $N$ counterexamples in 10 minutes, but generation has not terminated yet. The practicality result (*) for 3-player Routing is computationally demanding, but begins to produce results after the time limit. At around 60 minutes, we have over 100 counterexamples. TO indicates a timeout after 2 hours.**

## 7 RELATED WORK

Game theory opens up new venues in security and privacy analysis [8], particularly within blockchain technologies [21]. Game-theoretic modeling approaches in support of blockchain security have recently emerged [31, 36], complemented by specific analysis of several attack vectors, such as the griefing attack [24]. Our work complements these efforts as we express game-theoretic security in first-order linear real arithmetic and turn protocol security into an SMT-solving problem. Extended with game-theoretic case splitting and counterexample generation, we scale game-theoretic security to large protocols and fully automate it. Our work is thus orthogonal to [31].

The work of [5] focuses on game-theoretic security specific to the Ethereum blockchain [35], but does not reason about punishment mechanisms. While these works provide rigorous game-theoretic models, they lack support for automated reasoning, hindering their scalability and computer-aided certification.

Focusing on formal verification for security, the Tamarin [26], Verifpal [17], and Proverif [3] frameworks study general protocols, while the Verisol approach targets the Ethereum blockchain [34]. These methods operate on cryptographic security properties, proving whether certain actions are cryptographically possible. Our work complements these techniques by analyzing and establishing whether punishment mechanisms work as intended.

A similar line of research is analysis of cryptographic security properties in rational cryptography [1, 12, 13, 15, 22]. Rational cryptography does not assume players to be inherently honest or malicious, but instead *rational*. However, rational cryptography verifies cryptographic alignment of rationality and honesty, whereas our work in CheckMate focuses on incentive/punishment mechanisms modeled via games.

Another game-theoretic security reasoning framework is introduced by [18, 19]. Here, concurrent stochastic game structures (CGSs) are modeled within probabilistic asynchronous-time temporal logic with rewards (rPATL). The respective CGS are verified via model-checking of PATL formulas. Unlike this framework, our work does not reason with uncertainty. Instead, we provide a decidable first-order logic fragment for proving game-theoretic security. Thanks to our logical precision, we can avoid the computational burden of handling probability (reward) operators. Our EFGs also avoid concurrent game strategies and provide a deterministic game structure.

When it comes to automated game-theoretic analysis, it is worth noting the modeling and verification support offered by Gambit [25], PRISM-games [18] and Open Games [11]. These frameworks, however, are restricted to constant numeric utilities and cannot process symbolic utilities, limiting their applicability to reason about all game actions/strategies. We are not aware of other game-theoretic frameworks that natively support symbolic utilities, which is a key feature of CheckMate. Compositional reasoning in Open Games [11] does enable modular analysis, which we aim to further investigate in order to split large EFGs (such as routing games) into subgames.

## 8 CONCLUSION AND DISCUSSION

Game-theoretic security analysis provides new ways to derive security guarantees and identify attack vectors. Yet, automation in this area was so far limited, if at all, hindering the applicability and scalability of game-theoretic security analysis. Our work addresses these obstacles and implements novel methods for deciding security properties in games with symbolic utilities. We reduce security analysis to satisfiability solving over game strategies expressed in first-order theory of linear arithmetic. We provide full automation of within the new CheckMate framework, scaling security analysis to very large game trees, by automatically identifying necessary case splits for efficient reasoning. In addition, CheckMate supports the generation of counterexamples and necessary preconditions to security, enabling model repair and synthesis of game properties.

***Limitations and Challenges for Future Work.*** Modeling protocols requires considerable expertise, and the precision of our analysis depends on the accuracy of the underlying model. To maximize application of the CheckMate framework, we must provide ways to accurately and efficiently model protocols as EFGs. Partial automation of the EFG modeling process is therefore a valuable task to be addressed further.

Currently, CheckMate only supports linear utilities. However, protocols exist that naturally involve nonlinear terms over utilities. For example, ratios in market settings produce terms containing multiplication of two variables. While CheckMate internally already supports nonlinear arithmetic formulas over utilities, it comes with the caveat that the resulting SMT queries are also nonlinear and therefore significantly harder. Improving the performance of SMT solving over nonlinear arithmetic in the setting of CheckMate is its own challenge, which we aim to address in the future.

Finally, extending CheckMate with compositional game analysis is another direction we are already investigating. We believe compositionality may enable us to support randomized game aspects, for example model behavior that is not controllable by any player. In addition, compositional game analysis might ease modeling further real-world protocols as EFGs.

## ACKNOWLEDGMENTS

This research was supported by the ERC Consolidator Grant ARTIST 101002685, the Austrian FWF SFB project SpyCoDe F8504, and the SecInt Doctoral College funded by TU Wien.

## REFERENCES

[1] Gilad Asharov and Yehuda Lindell. 2009. Utility Dependence in Correct and Fair Rational Secret Sharing. Cryptology ePrint Archive. https://eprint.iacr.org/2009/373

[2] Clark Barrett and Cesare Tinelli. 2018. Satisfiability Modulo Theories. In *Handbook of Model Checking*. Springer, UC Berkeley, 305–343.

[3] Bruno Blanchet. 2014. Automatic Verification of Security Protocols in the Symbolic Model: The Verifier ProVerif. In *Foundations of Security Analysis and Design.* Springer, Cham, 54–87. https://doi.org/10.1007/978-3-319-10082-1_3

[4] Lea Salome Brugger, Laura Kovács, Anja Petković Komel, Sophie Rain, and Michael Rawson. EasyChair, 2023. CheckMate: Automated Game-Theoretic Security Reasoning. EasyChair Preprint no. 10853.

[5] Krishnendu Chatterjee, Amir Goharshady, and Arash Pourdamghani. 2019. Probabilistic Smart Contracts: Secure Randomness on the Blockchain. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, Seoul, South Korea, 403–412.

[6] George E Collins. 1975. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decompostion. In *Automata Theory and Formal Languages*. Springer, Berlin, Heidelberg, 134–183.

[7] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS*. Springer, Berlin, Heidelberg, 337–340.

[8] Cuong T. Do, Nguyen H. Tran, Choongseon Hong, Charles A. Kamhoua, Kevin A. Kwiat, Erik Blasch, Shaolei Ren, Niki Pissinou, and Sundaraja Sitharama Iyengar. 2017. Game Theory for Cyber Security and Privacy. *ACM Comput. Surv.* 50, 2, Article 30 (2017), 37 pages.

[9] Bruno Dutertre and Leonardo De Moura. 2006. A Fast Linear-Arithmetic Solver for DPLL (T). In *CAV*. Springer, Berlin, Heidelberg, 81–94.

[10] Yeting Ge and Leonardo De Moura. 2009. Complete Instantiation for Quantified Formulas in Satisfiability Modulo Theories. In *CAV*. Springer, Berlin, Heidelberg, 306–320.

[11] Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. 2016. Compositional Game Theory. arxiv.org/abs/1603.04641

[12] Ronen Gradwohl. 2010. Rationality in the full-information model. In *Theory of Cryptography*. Springer, Berlin, Heidelberg, 401–418.

[13] Ronen Gradwohl, Noam Livne, and Alon Rosen. 2013. Sequential Rationality in Cryptographic Protocols. *ACM Trans. Econ. Comput.* 1, 1, Article 2 (jan 2013), 38 pages. https://doi.org/10.1145/2399187.2399189

[14] ISO. 2017. *The JSON data interchange syntax.* ISO 21778:2017. International Organization for Standardization, Geneva, Switzerland.

[15] Sergei Izmalkov, Silvio Micali, and Matt Lepinski. 2005. Rational Secure Computation and Ideal Mechanism Design. In *FOCS*. IEEE, Pittsburgh, 585–594.

[16] Dejan Jovanović and Leonardo De Moura. 2012. Solving Non-Linear Arithmetic. In *IJCAR*. Springer, Manchester, 339–354.

[17] Nadim Kobeissi, Georgio Nicolas, and Mukesh Tiwari. 2020. Verifpal: Cryptographic Protocol Analysis for the Real World. In *Progress in Cryptology*. Springer International Publishing, Cham, 151–202.

[18] Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. 2020. PRISM-games 3.0: Stochastic Game Verification with Concurrency, Equilibria and Time. In *CAV*. Springer International Publishing, Cham, 475–487.

[19] Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. 2021. Automatic Verification of Concurrent Stochastic Systems. *Formal Methods Syst. Des.* 58, 1-2 (2021), 188–250. https://doi.org/10.1007/s10703-020-00356-y

[20] Mark H. Liffiton, Alessandro Previti, Ammar Malik, and Joao Marques-Silva. 2016. Fast, Flexible MUS Enumeration. *Constraints* 21, 2 (2016), 223–250.

[21] Ziyao Liu, Cong Nguyen, Wenbo Wang, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. 2019. A Survey on Blockchain: A Game Theoretical Perspective. *IEEE Access* 7 (2019), 47615–47643.

[22] Anna Lysyanskaya and Nikos Triandopoulos. 2006. Rationality and Adversarial Behavior in Multi-Party Computation. In *Annual International Cryptology Conference*. Springer, Berlin, Heidelberg, 180–197.

[23] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In *Network and Distributed System Security Symposium.* The Internet Society, San Diego, CA, USA.

[24] Subhra Mazumdar, Prabal Banerjee, Abhinandan Sinha, Sushmita Ruj, and Bimal Roy. 2022. Strategic Analysis to Defend against Griefing Attack in Lightning Network. arxiv.org/abs/2203.10533

[25] Richard Mckelvey, Andrew McLennan, and Theodore Turocy. 2005. Gambit: Software Tools for Game Theory.

[26] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *CAV*. Springer, Berlin, Heidelberg, 696–701.

[27] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System.

[28] Van-Hau Nguyen and Son T Mai. 2015. A New Method to Encode the At-Most-One Constraint into SAT. In *Information and Communication Technology.* ACM, New York, NY, USA, 46–53.

[29] Martin J. Osborne and Ariel Rubinstein. 1994. *A Course in Game Theory.* The MIT Press, Cambridge, USA.

[30] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf.

[31] Sophie Rain, Georgia Avarikioti, Laura Kovács, and Matteo Maffei. 2023. Towards a Game-Theoretic Security Analysis of Off-Chain Protocols. In *CSF*. IEEE Computer Society, Los Alamitos, CA, USA, 31–46.

[32] Alexander Schrijver. 1999. *Theory of Linear and Integer Programming.* Wiley, Hoboken, New Jersey, USA.

[33] Raymond M Smullyan. 1995. *First-Order Logic.* Dover Publications, New York.

[34] Yuepeng Wang, Shuvendu K. Lahiri, Shuo Chen, Rong Pan, Isil Dillig, Cody Born, Immad Naseer, and Kostas Ferles. 2019. Formal Verification of Workflow Policies for Smart Contracts in Azure Blockchain. In *VSTTE*. Springer International Publishing, Cham, 87–106.

[35] Gavin Wood. 2014. Ethereum: A Secure Decentralised Generalised Transaction Ledger. https://gavwood.com/paper.pdf

[36] Paolo Zappalà, Marianna Belotti, Maria Potop-Butucaru, and Stefano Secci. 2021. Game Theoretical Framework for Analyzing Blockchains Robustness. , 18 pages.