Divide and Conquer: A Compositional Approach to Game-Theoretic Security

IVANA BOCEVSKA, TU Wien, Austria
ANJA PETKOVIĆ KOMEL, Argot Collective, Switzerland
LAURA KOVÁCS, TU Wien, Austria
SOPHIE RAIN, Argot Collective, Switzerland
MICHAEL RAWSON, University of Southampton, United Kingdom

We propose a compositional approach to combine and scale automated reasoning in the static analysis of decentralized system security, such as blockchains. Our focus lies in the game-theoretic security analysis of such systems, allowing us to examine economic incentives behind user actions. In this context, it is particularly important to certify that deviating from the intended, *honest* behavior of the decentralized protocol is not beneficial: as long as users follow the protocol, they cannot be financially harmed, regardless of how others behave. Such an economic analysis of blockchain protocols can be encoded as an automated reasoning problem in the first-order theory of real arithmetic, reducing game-theoretic reasoning to satisfiability modulo theories (SMT). However, analyzing an entire game-theoretic model (called a game) as a single SMT instance does not scale to protocols with millions of interactions. We address this challenge and propose a *divide-and-conquer security analysis* based on compositional reasoning over games. Our compositional analysis is incremental: we divide games into subgames such that changes to one subgame do not necessitate re-analyzing the entire game, but only the ancestor nodes. Our approach is sound, complete, and effective: combining the security properties of subgames yields security of the entire game. Experimental results show that compositional reasoning discovers intra-game properties and errors while scaling to games with millions of nodes, enabling security analysis of large protocols.

CCS Concepts: • **Theory of computation → Automated reasoning**; **Algorithmic game theory**; • **Security and privacy → Logic and verification**; **Distributed systems security**.

Additional Key Words and Phrases: Game Theory, Security, SMT Solving, Automated Reasoning

ACM Reference Format:
Ivana Bocevska, Anja Petković Komel, Laura Kovács, Sophie Rain, and Michael Rawson. 2025. Divide and Conquer: A Compositional Approach to Game-Theoretic Security. *Proc. ACM Program. Lang.* 9, OOPSLA2, Article 342 (October 2025), 26 pages. https://doi.org/10.1145/3763120

1 Introduction

Decentralized systems based on blockchain technology, such as cryptocurrencies [Nakamoto 2009] and decentralized finance [Buterin 2014], are in need of security guarantees. Establishing such guarantees is usually first approached by the formal static analysis of the underlying cryptographic protocols [Blanchet 2014; Kobeissi et al. 2020; Meier et al. 2013; Wang et al. 2020]. Then, game-theoretic security analysis [Rain et al. 2023; Zappalà et al. 2020] is employed to ensure economic incentives in a protocol align with intended functional outcomes. The latter ensures malicious

Authors' Contact Information: Ivana Bocevska, TU Wien, Vienna, Austria, ivana.bocevska@tuwien.ac.at; Anja Petković Komel, Argot Collective, Zug, Switzerland, anja@argot.org; Laura Kovács, TU Wien, Vienna, Austria, laura.kovacs@tuwien.ac.at; Sophie Rain, Argot Collective, Zug, Switzerland, sophie.rain@argot.org; Michael Rawson, University of Southampton, Southampton, United Kingdom, michael@rawsons.uk.

ACM 2475-1421/2025/10-ART342
https://doi.org/10.1145/3763120

actions and security attacks can be prevented via punishment mechanisms. This paper scales and improves game-theoretic security with satisfiability modulo theory (SMT) solving over blockchain security. We were motivated by the compositional static analysis in [Blackshear et al. 2018].

Recent work shows that automatic analysis of game-theoretic models (called games) is tractable via SMT solving in first-order real arithmetic. In particular, game-theoretic analysis is reduced to solving a single large SMT instance [Brugger et al. 2023]. However, this style of automated analysis has inherent limitations. A major problem is *scalability*: game models of large protocols are huge, yielding enormous SMT instances that cannot be solved in reasonable time. Another challenge is *game-theoretic modeling*: it is much more convenient to reason about protocol parts/phases in a modular, independent manner and compose their results into results over the entire protocol. Such convenience becomes even more pronounced in the presence of repeated phases.

This paper addresses the aforementioned challenges and introduces a *compositional approach to game-theoretic security* (Section 5). Given a protocol, we study its parts independently (as subgames), analyze the subgames' security, and combine their results to enforce security of the game that models the entire protocol. In other words, we perform a *divide-and-conquer* approach for game-theoretic security analysis, whose automation is feasible via SMT solving (Section 6). Game modeling is not in the scope of this paper, but interleaving the modeling and the analysis of a game, as shown in Section 7, makes it feasible to verify even complex real-world models with over 100 million nodes.

Compositional reasoning is, however, not trivial, as illustrated by Example 4.1, which shows that SMT queries may not be naïvely split into subqueries, as constraints in one may interact with constraints in another (Section 4). Further, a security result of a subgame cannot be simply propagated upwards, as in our experiments, we have encountered all four possible scenarios: a subgame is not secure, but the entire game is, and vice-versa; both subgame and entire game are secure; both are not secure. Our divide-and-conquer approach provides a theoretically *sound and complete* way to decompose reasoning into fine-grained SMT queries over subgames (Theorem 6.4).

To the best of our knowledge, our approach is the first compositional method for game-theoretic security. We implement our work as an extension of the CheckMate tool [Rain et al. 2024], resulting in our CheckMate2.0 framework[1]. While our approach in this paper is illustrated via a simple example, our experiments demonstrate that divide-and-conquer reasoning in CheckMate2.0 enables game-theoretic analysis even for complex real-world protocols with millions of nodes. Compositional reasoning allows us, thus, to scale game-theoretic security analysis to large code bases, improving the efficiency of automated security analysis in general.

1.1 Setting and Wider Application

Game theory provides a rich formalism for reasoning about systems and their economics. Here we work with *extensive-form* games, which are essentially fixed finite trees where each branch has an associated player. Playing these games involves a traversal of the tree from root to leaf, with players deciding which branch they wish to take. At the end of a game, each player receives a *utility*: this could be a number or a symbolic expression like $2c + d$. *Game-theoretic security* considers a subset of possible traversals *honest* and aims to ensure that honest behavior is compatible with economic incentives present in a game. It has been shown that if a game enjoys a handful of properties, it amounts to game-theoretic security [Rain et al. 2023]. Automated reasoning tools for game-theoretic security ingest a game and attempt to (dis)prove its security.

While the main motivation of our work comes from blockchain verification, our framework can be used for arbitrary (extensive-form) game-theoretic models of any kind of turn-based interaction, such as computer security in a wider sense, correctness of concurrent and distributed systems,

[1] https://github.com/apre-group/checkmate/releases/tag/OOPSLA25

argumentation, negotiation, or even the design and analysis of board games. To showcase this claim we provide the following example.

Example 1.1. Suppose that a bank concurrently executes two procedures concerning the same customer. (1) A credit check requests the total amount held by the customer, who has both a savings and a current account. This requires two reads: (1a) the present value of the current account, (1b) the present value of the savings account. (2) The customer has requested regular automated transfers of money from the current account to a savings account. This requires two writes: (2a) debit the current account, (2b) credit the savings account. There are two hazards: the sequence 1a-2a-2b-1b overreports the customer's total amount (higher utility), while the sequence 2a-1a-1b-2b underreports it (lower utility). All other sequences report a correct total ("honest" (actual) utility). This can be modeled as a game, where analyzing game-theoretic security includes checking that no interleaving reports an incorrect total (utility). Here, without some kind of transaction control, this is violated and our framework can enumerate both possible serializations where this occurs.

1.2 Our Contributions.

We bring the following contributions[2].

- We introduce a *compositional framework for game-theoretic security analysis* (Section 5). Our framework defines player-dependent notions of security properties, which in turn enables divide-and-conquer reasoning over games. We divide games into subgames while ensuring that the resulting reasoning is both sound and complete.
- We advocate divide-and-conquer algorithmic reasoning to automate compositional modeling and security analysis (Section 6). We interleave subgame and supergame (parent game) reasoning, by using the security result of a subgame within leaves of their respective supergames.
- Our compositional framework naturally supports the *generation of counterexamples* if security properties are violated. Moreover, we *revise game preconditions* in order to strengthen and enforce security. When security is established, we *extract a game strategy* as a proven security certificate (Sections 6.2 and 6.3).
- We implement compositional game reasoning in the CheckMate2.0 tool. Our experiments show that compositionality significantly improves runtime and supports efficient case-splitting over symbolic game utilities (Section 7), enabling verification of real-world protocols.

Outline. This paper is structured as follows. Section 2 introduces common game-theoretic concepts relevant to our work. Section 3 summarizes the notion of game-theoretic security, including security properties, quantification over utilities, and counterexamples, as largely defined by previous work [Brugger et al. 2023; Rain et al. 2023; Zappalà et al. 2020]. Section 4 briefly shows the complexity of our work before our contributions are presented in Sections 5 to 7 as listed above.

2 Preliminaries

We assume familiarity with standard first-order logic [Smullyan 1995] and real arithmetic in the context of SMT solving [Barrett and Tinelli 2018; Bjørner and Nachmanson 2024]. We next introduce common game-theoretic concepts relevant to our work.

A *game* is a finite object with finitely many *players*. Players choose from a finite set of *actions* until the game ends, whereupon they receive a *utility*. The focus is on perfect information *Extensive Form Games* (EFGs) [Osborne and Rubinstein 1994] in which the actions are chosen sequentially with full knowledge of all previous actions. Games may yield collective benefit or loss, i.e. they are not necessarily *zero-sum*.

[2]Formal proofs of all our claims can be found in [Bocevska et al. 2025a].

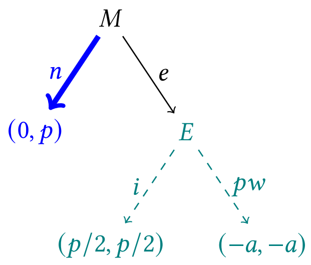

Fig. 1. Market Entry Game Γ_{me}, with $a, p > 0$. Utility tuples state M's utility first, E's second.

Definition 2.1 (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 $h = (e_1, ..., e_n)$, with $e_i \in E$, that starts from the root of G is called a *history*. We denote the set of histories $\mathcal{H}$. 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* are those histories that are not terminal. Non-terminal histories h have assigned a next player denoted as $P(h) \in N$. Player $P(h)$ chooses from the set $A(h)$ of possible actions following h.

Definition 2.2 (Honest History). In an EFG Γ, we call a terminal history *honest*, denoted by h^*, if it represents expected behavior in Γ. An EFG Γ can have many honest histories; security analysis over Γ is always performed relative to a chosen and fixed honest history (see Section 3.1).

Example 2.3 (Market Entry Game). Consider the Market Entry game Γ_{me} of Figure 1. This game has been chosen for its simplicity to ease readability; our approach can, however, be applied to real-world examples, as shown in Section 7. The Market Entry game Γ_{me} has two players: M representing a new company and E an established company. At the root, it is the turn of player $P(\emptyset) = M$ to choose from actions $A(\emptyset) = \{n, e\}$. Action n represents *not* entering the market, producing a terminal history (n) where M gets 0 utility and E gets all of the profits $p > 0$. Action e represents entering the market, in which case E can respond by either ignoring this move and thus splitting profits equally, or by entering a price war that damages both players.

Utilities in game theory are usually numeric constants. We generalize utilities to *symbolic* terms in real arithmetic and thus encode all possible values within given constraints. Variables and numeric constants are evaluated over the real numbers extended by a finite set of *infinitesimals*, closer to zero than any real number. Infinitesimals model subjective (in)conveniences that do not relate directly to funds, such as opportunity cost. We model infinitesimals with 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, using $a, b, c \ldots$ for real variables, and $\alpha, \beta, \gamma, \ldots$ for infinitesimals. The utility term $a + \alpha - \varepsilon$ is therefore represented as $(a, 0) + (0, \alpha) - (0, \varepsilon) = (a, \alpha - \varepsilon)$.

Example 2.4. We could modify the Market Entry game from Figure 1 by adding an infinitesimal $\alpha > 0$ to the utility of player M at (e, i). Player M's new utility $\frac{p}{2} + \alpha$ at leaf (e, i), then represents half of the profit p and the additional benefit of entering the market α, as M is motivated to establish a new entity.

To formulate game-theoretic security properties, we need the following definitions for EFGs.

Definition 2.5 (EFG Properties). Let $\Gamma = (N, G)$ be an EFG.

Strategy A *strategy* σ for a group of players $S \subseteq N$ is a function mapping non-terminal histories $h \in \mathcal{H} \setminus \mathcal{T}$, where one of the players in group S has a turn $P(h) \in S$, to the possible actions $A(h)$. We write $\mathcal{S}_S$ for the set of strategies for group S, and $\mathcal{S}$ for $\mathcal{S}_N$ which we call *joint strategies*. We refer to the union of strategies with disjoint domains as a *combined strategy* and denote it as a tuple. To combine e.g. $\sigma_S \in \mathcal{S}_S$ and $\tau_{N-S} \in \mathcal{S}_{N-S}$, we write $(\sigma_S, \tau_{N-S}) \in \mathcal{S}$.

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

Following Honest History A strategy for a player p *follows the honest history* h^* if, at every node along the honest history, where p is making a choice, the strategy chooses the action in h^*. For every other node, there is no constraint.

Utility Function The *utility function* $u_p(\sigma)$ assigns to player $p \in N$ their utility at the resulting terminal history of the joint strategy $\sigma \in \mathcal{S}$, that is $u_p(\sigma) := u_p(H(\sigma))$. We sometimes write all player utilities for a joint strategy $\sigma \in \mathcal{S}$ as $u(\sigma)$, denoting a tuple of size $|N|$.

Subgame *Subgames* $\Gamma_{|h}$ of Γ are formed from the same set N of players and a subtree of G, and are therefore identified by the history h leading to the subtree $G_{|h}$. A history $t \in \mathcal{H}_{|h}$ of $\Gamma_{|h}$ is a suffix of a history $(h, t) \in \mathcal{H}$, a strategy $\sigma \in \mathcal{S}_{|h}$ of $\Gamma_{|h}$ is a restriction of a strategy $\tau \in \mathcal{S}$ to the nodes in $\mathrm{G}_{|h}$, that is $\tau_{|h} = \sigma$, and the utility function $u_{|h}$ of $\Gamma_{|h}$ assigns each joint strategy $\sigma \in \mathcal{S}_{|h}$ the utility of the yielded leaf $u_{|h}(\sigma) := u(h, H(\sigma))$. This includes trivial subgames: leaves or the entire tree Γ at the empty history.

Supergame If h' is a prefix of h, $\Gamma_{|h'}$ is a supergame of $\Gamma_{|h}$.

Subgame along/off Honest History Let h^* be the honest history. A subgame $\Gamma_{|h}$ is *along* the honest history iff h is a prefix of h^*; that is, there is a history $g \in \mathcal{H}_{|h}$ in the subtree such that $(h, g) = h^*$. Otherwise, $\Gamma_{|h}$ is *off* the honest history.

Intuitively, a *subgame* is the part of the game that is still to be played after some actions have been taken already. A *supergame* of a subgame is any game tree that embeds the subgame as the subtree. For the sake of readability, we use the following simplifications. From now on we use *subgame*/*subtree* and *supergame*/*supertree* interchangeably. We write $u(\sigma_S, \tau_{N-S}) := u((\sigma_S, \tau_{N-S}))$ for the combined strategy $(\sigma_S, \tau_{N-S}) \in \mathcal{S}$. For history $k \in \mathcal{H}$, we write $k_{|h}$ to express the suffix of k after h, that is $(h, k_{|h}) = k$.

Example 2.6. Consider again the Market Entry game Γ_{me} in Figure 1. A joint strategy $\tau \in \mathcal{S}$ could have M taking action n initially, and player E taking i after (e). M's (single) strategy $\tau_M \in \mathcal{S}_M$ takes action n at the root. Joint strategy τ yields utility $u_E(\tau) = p$ for player E. The history resulting from τ is (n).

The subgame $\Gamma_{me|(e)}$ after history (e) is indicated by dashed lines in Figure 1. It has players $\{M, E\}$ and a tree where E must choose between action i with utility $(\frac{p}{2}, \frac{p}{2})$ and action pw with utility $(-a, -a)$. Considering honest history (n) the subtree $\Gamma_{me|(e)}$ is *off* the honest history, whereas the trivial subtree $\Gamma_{me|(n)}$ after action n is *along* the honest history.

The Market Entry game has $2 \times 2 = 4$ joint strategies as M chooses from two possible actions, and independently also E picks one action out of two.

3 Game-Theoretic Security Properties

Our work models real-life protocols as extensive form games (EFGs). Subsequently, we reduce the security analysis of a protocol to the game-theoretic security analysis of its corresponding EFG. According to [Zappalà et al. 2020] an adversary could execute an attack in a protocol for personal gain or harming somebody. Therefore, we consider a protocol to be *game-theoretically secure* if the following properties hold:

(P1) **Byzantine Fault-Tolerance.** Even in the presence of adversaries, honest players do not suffer loss. That is, 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 it yields the best payoff. Hence, in a secure protocol, a rational "attacker" is behaving honestly and no adversary gets personal gain by deviation.

3.1 Security Properties for Subgames

As elaborated in [Rain et al. 2023], the high-level properties (P1) and (P2) can be ensured through the game-theoretic concepts *weak(er) immunity*, *collusion resilience*, and *practicality*. Property (P1) is ensured by weak(er) immunity and (P2) by collusion resilience and practicality. We take the definitions of these security properties as posed in [Brugger et al. 2023], and adapt them for any subtree of Γ, to accommodate a compositional game-theoretic approach (Section 5). Note that the definitions coincide when the entire game Γ is taken as the subtree.

Since we allow symbolic utilities, we do not necessarily know their ordering, respectively relation to zero. This knowledge is however needed to evaluate the security properties. This is why, in this subsection, we assume a total order on the symbolic utility terms. We will lift this assumption in Section 3.2.

Definition 3.1 (Weak Immunity). A subtree $\Gamma_{|h}$ of game Γ with honest history h^* is *weak immune*, if a strategy $\sigma \in \mathcal{S}_{|h}$ exists such that all players p following σ always receive non-negative utility:

$$\exists \sigma \in \mathcal{S}_{|h}. \forall p \in N\ \forall \tau \in \mathcal{S}_{|h}.\ u_p(\sigma_p, \tau_{N-p}) \geq 0\ . \qquad (\mathrm{wi}(\Gamma_{|h}))$$

If h is along h^*, additionally $H_{|h}(\sigma) = h^*_{|h}$ has to hold.

Example 3.2. The Market Entry game from Figure 1 with honest history (n) is weak immune: according to Definition 3.1 we look at the trivial subtree Γ_{me}, after the empty history $h = \emptyset$. Since the empty history is always along the honest history, following the last sentence of Definition 3.1, the desired strategy σ has to yield the honest history (n). If M behaves honestly both players get a nonnegative utility; if M deviates via e, player E can choose action i and obtains a positive utility $\frac{p}{2}$. Hence, the witness strategy σ assigns action n to the empty history: $\sigma(\emptyset) = n$, and action i to history (n): $\sigma((n)) = i$.

Sometimes, weak immunity is too restrictive and we take *weaker immunity* to ensure (P1).

Definition 3.3 (Weaker Immunity). A subtree $\Gamma_{|h}$ of game Γ with honest history h^* is *weaker immune*, if there exists a strategy $\sigma \in \mathcal{S}_{|h}$, such that all players p that follow σ always receive at least a negative infinitesimal:

$$\exists \sigma \in \mathcal{S}_{|h}. \forall p \in N\ \forall \tau \in \mathcal{S}_{|h}.\ \mathsf{real}(u_p(\sigma_p, \tau_{N-p})) \geq 0\ . \qquad (\mathrm{weri}(\Gamma_{|h}))$$

If h is along h^*, additionally $H_{|h}(\sigma) = h^*_{|h}$.

Next, the property of collusion resilience requires the honest behavior to yield the best payoff, even in the presence of collusion.

Definition 3.4 (Collusion Resilience). A subtree $\Gamma_{|h}$ of the game Γ with honest history h^* is *collusion resilient* if there exists a strategy $\sigma \in \mathcal{S}_{|h}$ such that no strict subgroup of players can deviate to receive a joint utility greater than their joint honest utility:

$$\exists \sigma \in \mathcal{S}_{|h}. \forall S \subset N \; \forall \tau \in \mathcal{S}_{|h}. \; \sum_{p \in S} u_p(h^*) \geq \sum_{p \in S} u_p(\sigma_{N-S}, \tau_S) \; . \qquad (\mathrm{cr}(\Gamma_{|h}))$$

If h is along h^*, also $H_{|h}(\sigma) = h^*_{|h}$ has to hold.

Note that the collusion resilience of a subtree according to the above definition depends on the *honest utility*, the utility resulting from the honest history in the entire game Γ. The node containing the honest utility is not necessarily part of the considered subtree. We also note that we only check for strict subgroups $S \subset N$, since if the entire group of players is colluding, no player gets harmed[3].

Example 3.5. Consider again the Market Entry game from Figure 1 with the honest history (n). This is collusion resilient: we can take actions n for player M and pw for player E. Since it is a two-player game, the colluding group of players can only be a singleton. If M deviates from the honest behavior, they get utility $-a$, which is less than 0 in the honest case. If E deviates, the history is not affected, since M chooses action n. Thus, the game is collusion resilient.

Note that this is not what one would usually do; it is just a toy example to show what collusion resilience can protect from: It guarantees that the honest participants have a way to keep others from maliciously increasing their profit. The rationality of the honest choices is ensured through the next property *practicality*.

The next property of practicality ensures that, for all player decisions, the honest behavior is also "greedy": if all players act selfishly, that is they maximize their own utilities, the honest choice yields the best utility.

Definition 3.6 (Practicality). A subtree $\Gamma_{|h}$ of the game Γ with honest history h^* is *practical*, if there exists a strategy $\sigma \in \mathcal{S}_{|h}$ such that no player can deviate in any subtree to receive a strictly greater utility in the subtree:

$$\exists \sigma \in \mathcal{S}_{|h} \; \forall g \in \mathcal{H}_{|h} \; \forall p \in N \; \forall \tau \in \mathcal{S}_{|(h,g)}. \;\; u_{|g,p}(\sigma_{|g}) \geq u_{|g,p}(\tau_p, \sigma_{|g,N-p}) \; . \qquad (\mathrm{pr}(\Gamma_{|h}))$$

If h is along h^*, also $H_{|h}(\sigma) = h^*_{|h}$ has to hold.

Example 3.7. The Market Entry game from Figure 1 with the honest history (n) is not practical. Player E should choose i, as it yields a better utility. It is then not practical for M to choose n, as it yields utility 0, whereas action e yields the better utility $\frac{p}{2}$.

We finally note that every subtree $\Gamma_{|h}$ of a game Γ that is off the honest history is always practical.[4]

Example 3.8. Consider the Market Entry subgame after the non-terminal history (e), marked by teal dashed lines in Figure 1. We can always choose the action that yields the best utility for the current player E. The only way we can violate practicality is by having the best choice conflicting with the honest choice, which cannot happen when the subtree is off honest history.

The definitions weak(er) immunity, collusion resilience, and practicality can be extended to strategies and terminal histories in the following way.

[3]If the entire group of players N is colluding and getting a better utility, this could happen for several reasons. There could be lower subjective values, like less opportunity costs, smaller waiting times, etc.; or the players could harm the system, for example by conjuring coins out of thin air. Since our work is checking harm to other players, this kind of a check is currently out of scope, but could be a potentially useful extension.
[4]This is also formally proven in [Bocevska et al. 2025a].

Definition 3.9 (Security Properties of Strategies and Histories). A *strategy* $\sigma \in \mathcal{S}$ of a game Γ is weak(er) immune, collusion resilient, or practical, iff it can serve as the witness strategy in (wi($\Gamma_{|h}$)), (weri($\Gamma_{|h}$)), (cr($\Gamma_{|h}$)), or (pr($\Gamma_{|h}$)), respectively.

A *terminal history* $t \in \mathcal{T}$ of the game Γ is weak(er) immune, collusion resilient, or practical, iff there exists a strategy $\sigma \in \mathcal{S}$ that has the respective property and the resulting history of σ is t, that is $H(\sigma) = t$.

Together, these properties define game-theoretic security:

Definition 3.10 (Game-Theoretic Security). A game Γ with honest history h^* is *game-theoretically secure* if it is weak immune, collusion resilient, and practical.

In some applications, minor inconveniences can be neglected, and the notion of game-theoretic security can be weakened to *weaker immunity*, collusion resilience, and practicality.

Remark. Definition 3.10 implies that the listed security properties are not just any properties. They are *the* security properties that are sufficient to ensure game-theoretic security of game models as introduced and justified in [Rain et al. 2023].

In a protocol there can be several intended behaviors, corresponding to different honest histories. Those can be iteratively checked for security. Further, for an honest history we may get different strategies for different security properties. If a player deviated from the honest choice, the other players can choose among those strategies, depending on which attack they are defending against.

3.2 Total Orders

Similarly to [Brugger et al. 2023], in order to lift the assumption that we know how all utility terms relate, we make the security analysis relative to a finite set C of *initial constraints* on the symbolic variables appearing in the utility terms and explicitly universally quantify over the variables, as follows

$$\forall \vec{x}. \left(\bigwedge_{c \in C} c[\vec{x}] \right) \rightarrow \exists \sigma \in \mathcal{S}. H(\sigma) = h^* \wedge sp(\sigma)[\vec{x}] \ , \tag{1}$$

where $\vec{x} = (x_1, \dots, x_\ell)$ are the real/infinitesimal variables occurring in the utility terms T_u and $sp(\sigma)$ is the formula of a security property $sp \in \{wi, weri, cr, pr\}$ after existential quantification of the strategy: E.g. for weak immunity $wi(\sigma) = \forall p \in N \ \forall \tau \in \mathcal{S}_{|h}. \ u_p(\sigma_p, \tau_{N-p}) \geq 0$, and similarly for the other properties.

Furthermore, to efficiently handle the comparison of symbolic utilities in an SMT solver, we implement an equivalent version of the above formula by considering all consistent *total orders* $\preceq$ over the set T_u of utility terms appearing in the game Γ.

Theorem 3.11 (Game-Theoretic Security with Total Orders). *For an EFG Γ with honest history h^* and a finite set of initial constraints C, property (1) is equivalent to*

$$\forall (\preceq, T_u) \ \exists \sigma \in \mathcal{S}. H(\sigma) = h^* \wedge \ \ \forall \vec{x}. \left(\bigwedge_{c \in C \cup \preceq} c[\vec{x}] \right) \rightarrow sp(\sigma)[\vec{x}] \, . \tag{2}$$

3.3 Counterexamples

If a game violates a security property – that means if there is no joint strategy satisfying the security property – we can investigate why: Counterexamples serve the important purpose of providing attack vectors and thus pinpointing weaknesses of a protocol underlying the game model.

Counterexamples to Weak(er) Immunity. For the weak(er) immunity property, a counterexample is a harmed honest player p and a partial strategy of the other players $N - p$ such that no matter what honest actions player p chooses, p receives negative utility.

Definition 3.12 (Counterexamples to Weak(er) Immunity). Let Γ 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} of the other players $N - p$ such that s_{N-p} combined with any strategy σ_p of player p that follows the honest history h^*, yields a terminal history $H(s_{N-p}, \sigma_p) = t_{\sigma_p}$ with $u_p(t_{\sigma_p}) < 0$ (resp. for weaker immunity $\text{real}(u_p(t_{\sigma_p})) < 0$) and s_{N-p} is *minimal* with that property.

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 σ_p of player p such that (σ_p, s'_{N-p}) does not yield a terminal history, where s'_{N-p} is s_{N-p} without assigning action a to history h. That is, when following only actions of (σ_p, s'_{N-p}), we get stuck at an internal node of the tree.

Example 3.13. A counterexample to the weak immunity of the Market Entry game of Figure 1 with the (for this example considered) *honest* history (e, i) would be player M and a partial strategy for E, where they choose action pw. If M behaves honestly and chooses action e, they end up with the negative utility of $-a$ after the terminal history (e, pw).

Counterexample to Collusion Resilience. A counterexample to collusion resilience consists of a group of deviating players S and their partial strategy $s_S \in \mathcal{S}$, such that the joint utility of S is better than their joint honest utility, no matter how the other players $N - S$ react, while still following the honest history.

Definition 3.14 (Counterexamples to Collusion Resilience). Let Γ be an EFG and h^* the considered honest history. A *counterexample to h^* being collusion resilient* is a set of deviating players S together with a partial strategy s_S of players S such that s_S extended by any strategy σ_{N-S} of players $N - S$, which follows the honest history h^*, yields a terminal history $H(\sigma_{N-S}, s_S) = t_{\sigma_{N-S}}$ with

$$\sum_{p \in S} u_p(t_{\sigma_{N-S}}) > \sum_{p \in S} u_p(h^*)$$

and s_S is minimal with that property. The minimality of s_S is similar to the minimality of the partial strategy for weak(er) immunity.

Example 3.15. In the Market Entry game of Figure 1, a counterexample to the (for this example considered) *honest* history (e, pw) being collusion resilient is the deviating group $\{E\}$ with the partial strategy that takes action i after history (e). Since the honest player M can only take action e (while being honest), the deviating utility for E is $\frac{p}{2}$, which is greater than the honest one $-a$.

Counterexamples to Practicality. Intuitively, a counterexample to practicality of the honest history h^* has to provide a reason why a rational player would not follow h^*. At some point along h^* after a prefix h, there is an action a promising the current player $P(h)$ a strictly better utility than h^*. That means in the subgame $\Gamma_{|(h,a)}$ after (h, a) all practical terminal histories yield a utility for player $P(h)$ that is better than their honest one. Otherwise, other rational players would choose actions in $\Gamma_{|(h,a)}$ which disincentivize $P(h)$ to deviate from h^*.

Definition 3.16 (Counterexamples to Practicality). For an EFG Γ and honest history h^*, a *counterexample to practicality of h^** is a prefix h of h^* together with an action $a \in A(h)$, such that for all practical terminal histories t in the subgame $\Gamma_{|(h,a)}$ it holds that $u_{P(h)}(h^*) < u_{P(h)}((h, a, t))$.

Example 3.17. Recall that the Market Entry game Γ_{me} from Figure 1 with the honest history (n) is not practical. A counterexample to practicality is the empty prefix $h = \emptyset$ and the action e, as the

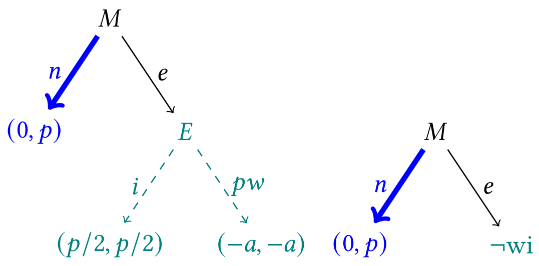

Fig. 2. Naive Compositionality of Weak Immunity for Market Entry Game Γ_{me}, $a, p > 0$.

only practical terminal history in $\Gamma_{me|(e)}$ is (i) which yields $\frac{p}{2}$ for player M, which is strictly better than the 0 in the honest case.

4 Unsound Naïve Approach to Compositionality

For a divide-and-conquer style of compositional game-theoretic security analysis, we would like to analyze a game tree by propagating security results of subtrees upwards to the parent/ancestor nodes of the supertree. However, naïvely propagating the yes/no security result of the subtree does not suffice, as shown in Example 4.1.

Example 4.1. Consider the Market Entry game Γ_{me}(Example 2.3) reproduced on the left-hand side of Figure 2, with honest history (n). Example 3.2 shows this game is weak immune. Now consider a naïve compositional approach looking at the subgame after non-terminal history (e), marked by teal dashed lines. Since player E can take action pw — leading to negative utility for M — this subtree is not weak immune. To mimic a naïve compositionality approach, we replace the subtree after (e) by ¬wi, shown on the right. Asked whether this *supertree* is weak immune, one would say no, as M could deviate from the honest history via e, which leads to a subtree that is not weak immune. This is, however, an incorrect conclusion since the Market Entry game is weak immune for the honest history (n), as shown in Example 3.2.

The main reason why the naïve approach above fails is that we need more information to be able to propagate a result from a subtree to its parent, namely that the subtree is not weak immune *only for player M*. In the parent, player M can achieve weak immunity by behaving honestly and choosing action n, ensuring weak immunity of the entire game tree. Similar additional information (see Theorem 5.7) is needed for the other security properties: collusion resilience requires which colluding groups the subtree is secure against; practicality requires the utilities resulting from terminal histories that are practical in the subtree (*practical utilities*). We now show that propagating this information yields a sound and complete compositional approach to game-theoretic security.

5 Compositional Game-Theoretic Security

Our compositional framework for game-theoretic security analysis is materialized via two crucial components:

(1) Stratified analysis of security properties over players, capturing player-wise security properties (Section 5.1).
(2) Splitting player-wise security properties into subgames, enabling us to propagate subgame reasoning for deriving supergame security (Section 5.2).

For simplicity, we assume a total order $\preceq$ on the occurring utility terms T_u in order to relate symbolic game utilities. As before, this assumption is relaxed in Section 6, generalizing our approach.

5.1 Security Properties Stratified over Players

We start with the following observation. While Example 4.1 shows that there are no implications of subtree and supertree results in general, subtrees *along the honest history* can in fact soundly pass negative (not secure) results up to their parents.

Theorem 5.1 (Equivalence of Non-Secure Games). *A game Γ with honest history h^* violates one of the security properties of weak(er) immunity, collusion resilience, or practicality iff there exists a history h along the honest history h^* such that $\Gamma_{|h}$ violates the respective security property.*

Intuitively, the honest history h^* "enforces" a path down the tree Γ: when a non-secure subtree $\Gamma_{|h}$ is encountered along this path, there is no way to compensate for it. Theorem 5.1, however, only propagates non-secure properties along the honest history. To allow for all analysis results to propagate from subgames to supergames, we *stratify game-theoretic security analysis over individual players.* This means we can analyze the security properties for a player (weak immunity: Definition 5.2, practicality: Definition 5.5), or player group (collusion resilience: Definition 5.4) at a time, without interfering with results of other players or groups (Theorem 5.6).

Definition 5.2 (Weak Immunity for a Player). A subgame $\Gamma_{|h}$ with honest history h^* is *weak immune for player $p \in N$*, if there exists a strategy $\sigma \in \mathcal{S}_{|h}$ such that no matter to which strategy $\tau \in \mathcal{S}_{|h}$ other players deviate, p's utility will be non-negative and, if h is along h^*, then also $H_{|h}(\sigma) = h^*_{|h}$:

$$\exists \sigma \in \mathcal{S}_{|h}.\ (h \text{ along } h^* \rightarrow H_{|h}(\sigma) = h^*_{|h}) \land \qquad (\mathrm{wi}_p(\Gamma_{|h}))$$
$$\forall \tau \in \mathcal{S}_{|h}.\ u_{|h,p}(\sigma_p, \tau_{N-p}) \geq 0 \ .$$

An analogous definition applies to *weaker immunity.*

Example 5.3 (Player-Wise Weak Immunity). Let us revisit the Market Entry game Γ_{me} with honest history (n) from Figure 1, considering one player at a time. The first player is M. The subgame $\Gamma_{me|(e)}$ after history (e) is not weak immune for M, since E could take action pw. Towards compositional reasoning, we try propagating this result to the supertree Γ_{me}. It leads to correctly reporting weak immunity for M: as M will honestly take action n, we avoid $\Gamma_{me|(e)}$.

For E, $\Gamma_{me|(e)}$ is weak immune as action i can always be chosen, yielding positive utility. We again try propagating this result, and also here conclude correctly that Γ_{me} is weak immune for E: all choices of M (whom we do not assume to be honest in the analysis of E), lead to either non-negative utility for E or to a subtree which is weak immune for E.

The definition for collusion resilience *against* a given player group is similar to Definition 5.2, by lifting the quantifier over the player subgroups $S \subset N$ to the front of the formula.

Definition 5.4 (Collusion Resilience against a Player Group). A subgame $\Gamma_{|h}$ of game Γ with honest history h^* is *collusion resilient against a group of players $S \subset N$*, if there exists a strategy $\sigma \in \mathcal{S}_{|h}$ such that no matter to which strategy $\tau \in \mathcal{S}_{|h}$ the players in S deviate, their joint utility will be not greater than their honest joint utility and, if h is along h^*, then also $H_{|h}(\sigma) = h^*_{|h}$:

$$\exists \sigma \in \mathcal{S}_{|h}.\ (h \text{ along } h^* \rightarrow H_{|h}(\sigma) = h^*_{|h}) \land \qquad (\mathrm{cr}_S(\Gamma_{|h}))$$
$$\forall \tau \in \mathcal{S}_{|h}.\ \sum_{p \in S} u_{|h,p}(\sigma) \geq u_{|h,p}(\tau_S, \sigma_{N-S}) \ .$$

Defining practicality for a single player, though, requires slight changes: instead of considering an arbitrary player p, we define practicality for that player whose turn it is in the considered subtree.

Definition 5.5 (Practicality for the Current Player). A subgame $\Gamma_{|h}$ of a game Γ with honest history h^* is *practical for the current player*, if there exists a strategy $\sigma \in \mathcal{S}_{|h}$ such that in each further subtree $\Gamma_{|(h,g)}$ no matter to which strategy $\tau \in \mathcal{S}_{|(h,g)}$ the current player $P(h,g)$ deviates, the utility of $P(h,g)$ in the subtree will not increase strictly and, if h is along h^*, then also $H_{|h}(\sigma) = h^*_{|h}$:

$$\exists \sigma \in \mathcal{S}_{|h}. \quad (h \text{ along } h^* \rightarrow H_{|h}(\sigma) = h^*_{|h}) \ \wedge \forall g \in \mathcal{H}_{|h}\ \forall \tau \in \mathcal{S}_{|(h,g)}. \qquad (\mathrm{pr}_P(\Gamma_{|h}))$$
$$u_{|(h,g),P(h,g)}(\sigma_{|g}) \geq u_{|(h,g),P(h,g)}(\tau_{P(h,g)}, \sigma_{|g,N-P(h,g)}) \ .$$

We now state our first crucial result towards compositionality: stratification of security analysis over players.

Theorem 5.6 (Player-Wise Security Properties). *A game Γ satisfies a security property iff it satisfies the respective security property player-wise. That is, the following equivalences hold:*

(1) *Γ weak immune $\Leftrightarrow \forall p \in N.\ \Gamma$ weak immune for p.*
(2) *Γ weaker immune $\Leftrightarrow \forall p \in N.\ \Gamma$ weaker immune for p.*
(3) *Γ collusion resilient $\Leftrightarrow \forall S \subset N.\ \Gamma$ collusion resilient against S.*
(4) *Γ practical $\Leftrightarrow \Gamma$ practical for the current player.*

5.2 Splitting and Combining Player-Wise Security Properties

Theorem 5.6 proves that the security analysis of a game can be carried out player-wise, instead of analyzing interactions between all (groups of) players. We now show that not only players can be treated individually, but *(super)game security can also be split into subgame security.* That is, the security of a supergame can be proven by proving player-wise security over subgames. This implies compositional game-theoretic security is sound and complete.

Theorem 5.7 (Compositional Game-Theoretic Security). *The game-theoretic security of an EFG Γ with honest history h^* can be computed compositionally. That is, the only information needed of a subtree $\Gamma_{|h}$, to decide whether Γ satisfies security property is*

- *for weak(er) immunity: for which players $p \in N$ the subtree $\Gamma_{|h}$ is weak(er) immune;*
- *for collusion resilience: against which player groups $S \subset N$ the subtree $\Gamma_{|h}$ is collusion resilient;*
- *for practicality:*
  - *if h is along h^*: whether $h^*_{|h}$ is practical in $\Gamma_{|h}$;*
  - *if h is not along h^*: the set $\mathbb{U}(h)$ containing all practical utilities of $\Gamma_{|h}$[5]. A utility $u(t)$ after terminal history $t \in \mathcal{T}$ is practical in subgame $\Gamma_{|h}$ iff t is practical in $\Gamma_{|h}$.*

Theorems 5.8, 5.10 and 5.12 establish how to compositionally compute player-wise security for each security property, yielding a constructive proof of Theorem 5.7.

Theorem 5.8 (Compositional Weak Immunity). *Let Γ be an EFG with honest history h^* and $p \in N$ a player. The following hold.*

(1) *A leaf of Γ is weak immune for p iff p's utility is non-negative:*

$$\forall t \in \mathcal{T}.\ wi_p(\Gamma_{|t}) \quad \Leftrightarrow \quad u_p(t) \geq 0\ .$$

(2) *A branch of Γ is weak immune for p, where p is not the current player, iff all children are weak immune for p:*

$$\forall h \in \mathcal{H} \setminus \mathcal{T}.\, p \neq P(h) \ \Rightarrow\ \big(wi_p(\Gamma_{|h}) \ \Leftrightarrow\ \forall a \in A(h).\, wi_p(\Gamma_{|(h,a)})\big)\ .$$

[5]The set $\mathbb{U}(h)$ is introduced properly in the paragraph before Theorem 5.12.

(3) *A branch of Γ along the honest history h^* is weak immune for the current player p, iff the child following h^* is weak immune for p. Let $a^* \in A(h)$ be the honest choice, i.e. (h, a^*) along h^*, then:*

$$\forall h \in \mathcal{H} \setminus \mathcal{T} .\ p = P(h) \wedge h \text{ along } h^* \quad \Rightarrow \quad \big(wi_p(\Gamma_{|h}) \ \Leftrightarrow \ wi_p(\Gamma_{|(h,a^*)})\big) \ .$$

(4) *A branch of Γ off the honest history h^* is weak immune for the current player p, iff there exists a child that is weak immune for p:*

$$\forall h \in \mathcal{H} \setminus \mathcal{T} .\ p = P(h) \wedge h \text{ off } h^* \quad \Rightarrow \quad \big(wi_p(\Gamma_{|h}) \ \Leftrightarrow \ \exists a \in A(h).\ wi_p(\Gamma_{|(h,a)})\big) \ .$$

Similar results to Theorem 5.8 hold for weaker immunity.

Example 5.9 (Compositional Weak Immunity). We revisit the Market Entry game Γ_{me} of Figure 1, with honest history (n). We compute that Γ_{me} is weak immune using our compositional approach, where we stratify over players first and then split Γ_{me} into subtrees.

We start with player M. Theorem 5.8 implies that Γ_{me} is weak immune for M iff $\Gamma_{me|(n)}$ is weak immune for M; since $\Gamma_{me|(n)}$ is a leaf, we must check that the utility of M is non-negative, i.e. $0 \geq 0$. As this is true, game Γ_{me} is weak immune for M.

Next, E. According to Theorem 5.8, the game Γ_{me} is weak immune iff $\Gamma_{me|(n)}$ and $\Gamma_{me|(e)}$ are weak immune for E. The subgame $\Gamma_{me|(n)}$ is weak immune for E if their utility is non-negative, i.e. $p \geq 0$, true by assumption. The subtree $\Gamma_{me|(e)}$ is now weak immune for E iff either $\Gamma_{me|(e,i)}$ or $\Gamma_{me|(e,pw)}$ is. E's utility at $\Gamma_{me|(e,i)}$ is $p/2 \geq 0$. Therefore Γ_{me} is weak immune for E, and from Theorem 5.6 it follows that Γ_{me} is weak immune.

Theorem 5.10 (Compositional Collusion Resilience). *Let Γ be an EFG with honest history h^* and honest utility $u^* = u(h^*)$. The following equivalences hold.*

(1) *A leaf of Γ is collusion resilient against $S \subset N$ iff the honest joint utility of the deviating players $p \in S$ is greater than or equal to their joint utility at that leaf:*

$$\forall t \in \mathcal{T} .\ cr_S(\Gamma_{|t}) \quad \Leftrightarrow \quad \sum_{p \in S} u_p^* \geq \sum_{p \in S} u_p(t) \ .$$

(2) *A branch of Γ, where the current player is in the deviating group $S \subset N$, is collusion resilient against S iff all children are collusion resilient against S:*

$$\forall h \in \mathcal{H} \setminus \mathcal{T} .\ P(h) \in S \quad \Rightarrow \quad \big(cr_S(\Gamma_{|h}) \ \Leftrightarrow \ \forall a \in A(h).\ cr_S(\Gamma_{|(h,a)})\big) \ .$$

(3) *A branch of Γ along the honest history h^*, where the current player is not in the deviating group $S \subset N$, is collusion resilient against S iff the child following h^* is collusion resilient against S. Let $a^* \in A(h)$ be the honest action, i.e. (h, a^*) along h^*, then:*

$$\forall h \in \mathcal{H} \setminus \mathcal{T} .\ P(h) \notin S \wedge h \text{ along } h^* \quad \Rightarrow \quad \big(cr_S(\Gamma_{|h}) \ \Leftrightarrow \ cr_S(\Gamma_{|(h,a^*)})\big) \ .$$

(4) *A branch of Γ off the honest history h^*, where the current player is not in the deviating group $S \subset N$, is collusion resilient against S iff there exists a child that is collusion resilient against S:*

$$\forall h \in \mathcal{H} \setminus \mathcal{T} .\ P(h) \notin S \wedge h \text{ off } h^* \quad \Rightarrow \quad \big(cr_S(\Gamma_{|h}) \ \Leftrightarrow \ \exists a \in A(h).\ cr_S(\Gamma_{|(h,a)})\big) \ .$$

Note that, if a player is in a deviating group, all child subtrees need to be collusion resilient even if we are along honest history, as the deviator might choose any action and potentially harm honest players. In contrast, for an honest player and a node off honest history, there needs to merely exist one collusion resilient child that the player can choose to defend against the deviating group.

Example 5.11 (Compositional Collusion Resilience). We compositionally compute the collusion resilience of the Market Entry game Γ_{me} (Figure 1) with honest history (n). We have two possible colluding groups, both singletons $\{M\}$ and $\{E\}$.

Consider $\{M\}$. At the root of Γ_{me}, since the player M is in the colluding group, all subtrees must be collusion resilient against $\{M\}$. Along the honest history we reach a leaf $\Gamma_{me|(n)}$, which is collusion resilient (it is the honest leaf). For subtree $\Gamma_{me|(e)}$ there needs to exist a collusion resilient child, which is the case in the leaf after (e, pw): utility $-a$ is strictly smaller than the honest utility 0.

Next, $\{E\}$. At the root, M is not in the deviating group. Hence, only the honest child $\Gamma_{me|(n)}$ need be collusion resilient against $\{E\}$, which it is, as it is the honest leaf; so the utility is equal to the honest one in part (1) of Theorem 5.10. This suffices to establish collusion resilience against $\{E\}$; checking $\Gamma_{me|(e)}$ is unnecessary.

Using Theorem 5.6, it follows that Γ_{me} is collusion resilient.

While we only have to remember boolean values of subtrees (i.e. whether it satisfies the property) to compositionally compute weak(er) immunity and collusion resilience per player(group), we need more information when reasoning about practicality: To correctly assess whether a tree is practical, one has to know the utility tuples of *all* practical terminal histories of its subtrees. For a given subtree $\Gamma_{|h}$ with practical terminal histories $\{t_1, \dots, t_n\}$, we define the set of practical utilities $\mathbb{U}(h)$ as $\mathbb{U}(h) := \{u_{|h}(t_i) : i = 1, \dots, n\}$.

Theorem 5.12 (Compositional Practicality). *Let Γ be an EFG with honest history h^* and $\mathbb{U}(h)$ be the set of practical utilities of subtree $\Gamma_{|h}$. Let u^* be the honest utility $u^* = u(h^*)$. Then the following identities and equivalences hold.*

(1) *In a leaf of Γ the only practical utility is that of the leaf.*

$$\forall t \in \mathscr{T}.\ \mathbb{U}(t) = \{u(t)\}\ .$$

(2) *The honest utility u^* is practical in a branch of Γ along h^* iff it is practical in the child following h^* and if for every other child at least one practical utility is not greater than u^* for the current player. Let $a^* \in A(h)$ be the honest action after h, then:*

$$\forall h \in \mathscr{H} \setminus \mathscr{T}.\ h \text{ along } h^* \quad \Rightarrow \quad \big(pr(\Gamma_{|h}) \quad \Leftrightarrow \\ pr(\Gamma_{|(h,a^*)})\ \wedge\ \forall a \in A(h) \setminus \{a^*\}\ \exists u \in \mathbb{U}((h,a)).\ u^*_{P(h)} \geq u_{P(h)}\big)\ .$$

(3) *A utility is practical in a branch of Γ off the honest history h^* iff it is practical in a child and if, for every other child, at least one practical utility is not greater for the current player.*

$$\forall h \in \mathscr{H} \setminus \mathscr{T}.\ h \text{ off } h^* \quad \Rightarrow \quad \big(\forall t \in \mathscr{T}_{|h}.\ u(t) \in \mathbb{U}(h) \quad \Leftrightarrow \\ \exists a \in A(h).\ u(t) \in \mathbb{U}((h,a)) \wedge \\ \forall a' \in A(h) \setminus \{a\}\ \exists u' \in \mathbb{U}((h,a')).\ u_{P(h)}(t) \geq u'_{P(h)}\big)\ .$$

Example 5.13 (Compositional Practicality). To compositionally compute the practicality of the Market Entry game Γ_{me} of Figure 1 with honest history (n), we start with the leaves of the tree, where the practical utilities are the utilities of the leaves. Moving upwards in the tree, we look at the subtree $\Gamma_{me|(e)}$, which is off the honest history, so we take the better utility for player E, setting $\mathbb{U}(e) = \{(\frac{p}{2}, \frac{p}{2})\}$. At the root of the tree, which is along the honest history, the practical utility of the honest subtree $(0, p)$ should be practical. Since all practical utilities of the non-honest child (there is just one) are better for player M (as $\frac{p}{2} > 0$), the honest utility is not practical. Theorems 5.6 and 5.12 then imply that Γ_{me} is not practical.

6 Automating Compositional Security Analysis

Section 5 assumed a total order $\preceq$ on game utility terms T_u. This section lifts this fixed ordering constraint and interprets the game variables in the utility terms T_u as real-valued variables $\vec{x}$, as explained in Section 3.2. Theorem 3.11 showed that quantification of the variables $\vec{x}$ can be done equivalently by grouping values of $\vec{x}$ that satisfy the same total order $(\preceq, T_u)$. In this section we combine this result with Theorem 5.6 and further pull the security property quantifications (over players, subgames and strategies) out of the variable $\vec{x}$ quantification, as they are independent. For weak immunity, for example, the formula (1) (with $sp = wi$) becomes equivalent to:

$$\forall(\preceq, T_u)\ \forall p \in N\ \exists \sigma \in \mathcal{S}.\, H(\sigma) = h^* \wedge \forall \tau \in \mathcal{S}\ .$$
$$\forall \vec{x}. \left(\bigwedge_{c \in C \cup \preceq} c[\vec{x}] \right) \to u_p(\sigma_p, \tau_{N-p})[\vec{x}] \geq 0\,. \tag{3}$$

This mapping of game-theoretic security from Theorem 3.11 to the player-wise security of Theorem 5.6 is crucial for automating compositional security: we only forward relatively small first-order expressions of the form

$$\forall \vec{x}. \left(\bigwedge_{c \in C \cup \preceq} c[\vec{x}] \right) \to ut_1[\vec{x}] \geq ut_2[\vec{x}]\ , \tag{4}$$

to an SMT solver, where ut_1 and ut_2 are term expressions over $\vec{x}$; checking such formulas is very feasible for SMT solvers.

As usual, to check whether (4) is a theorem, the property is first negated, and then an SMT solver is used to check satisfiability. This is where the simplified quantified structure of (4) becomes especially *friendly for automation*: The SMT solving of (4) happens in a purely existential fragment, for which efficient decision procedures exist [Barrett and Tinelli 2018; Bjørner and Nachmanson 2024]. The remaining reasoning in (3), about the players and the existence of strategies σ witnessing player-wise security, is performed using the compositional security results of Theorems 5.8, 5.10 and 5.12, without burdening the SMT solver. Such an interplay between SMT solving and compositional security eases automation, as illustrated below and detailed further in Section 6.1.

Example 6.1 (SMT Reasoning for Compositional Security). Revisiting the Market Entry game Γ_{me} of Figure 1, we study the SMT formulae resulting from (4), given initial constraints $C = \{a > 0, p > 0\}$. All symbolic utility terms occurring in the security properties of Γ_{me} are already totally ordered by the constraints in C. Hence, the only relevant total order $\preceq$ here that is consistent with C is $-a \prec 0 \prec p/2 \prec p$.

To analyze, for example, the weak immunity of the Market Entry game for player E compositionally, we follow the algorithm induced by Theorem 5.8. Starting at the root, we observe that we are *not* at a leaf and E is *not* the current player. Thus, by Theorem 5.8 it follows that we need to make sure that the subgames after history (n) and (e) are both weak immune for player E. The SMT reasoning is only needed when we reach a leaf, such as the one after history (n). The resulting SMT query is

$$\forall a, p.\ a > 0 \wedge p > 0 \to p \geq 0\ ,$$

which is trivially valid. Hence, the subtree after history (n) is weak immune for E for all allowed utility values. The analysis for the subgame after history (e) can be performed in an analogous way. Finally, we can combine the results for both subtrees to obtain the result for the supertree. In this manner, the inductive approach allows us to reduce reasoning about secure strategies to reasoning about utility terms in the leaves of the game tree.

6.1 Divide-and-Conquer Algorithms for Compositional Security

Our compositionality results from Theorem 5.6 and Theorems 5.8, 5.10 and 5.12, extended by a lazy total-order approach, induce a *divide-and-conquer approach for splitting and combining reasoning* *over game subtrees and supertrees.* Our overall divide-and-conquer framework for automating compositional game-theoretic reasoning is summarized in Algorithm 1, which in turn relies upon Algorithm 2 as well as upon Algorithms 3 to 5 from [Bocevska et al. 2025a]. We compositionally compute the game-theoretic security of a protocol, analyzing the (protocol model) game for all real-valued variables $\vec{x}$ of utitilty terms T_u, considering all total orders $\preceq$ at once. If we fail, we split the total orders into multiple cases, unless we can conclude that the respective security property cannot be satisfied even if we restrict the values of $\vec{x}$ to one total order $\preceq$. The case split we consider is induced by an SMT query as in property (4) when some but not all $\vec{x}$ satisfy the implication. We then split into total orders that enforce $ut_1[\vec{x}] \geq ut_2[\vec{x}]$, respectively $ut_1[\vec{x}] < ut_2[\vec{x}]$, in (4).

Algorithm 1: Function `SatisfiesProperty` for Compositional Game-Theoretic Security Reasoning.

```
input  : input instance $\Pi = (\Gamma, inf, C)$, honest history $h^*$, the name of a security property
         $sp \in \{wi, weri, cr, pr\}$, and the currently analyzed case (as set of SMT constraints) case.
output : true if $\Pi$ satisfies $sp$ in case case, false otherwise
S ← $\emptyset$
AddConstraints (S, $C \cup$ case)
result ← true
split ← null
for pg ∈ RelevantGroups($\Pi$, $sp$) do
    ($\text{result}_{pg}$, $\text{split}_{pg}$) ← ComputeSP ($\Pi$,$h^*$,S,$sp$,pg)
    if $\text{result}_{pg}$ = false then
        result ← $\text{result}_{pg}$
        split ← $\text{split}_{pg}$
        break
    end
end
if result = true then
    return true
end
if split = null then
    return false
end
for constr ∈ {split, ¬split} do
    if ¬SatisfiesProperty($\Pi$, $h^*$, $sp$, case ∪ {constr}) then
        return false
    end
end
return true
```

Algorithm 1: Function `SatisfiesProperty`. In Algorithm 1 an instance Π, which contains the game tree Γ, the set of infinitesimal variables inf (as introduced in Section 2), and the set of initial constraints C, is given as input. The input to Algorithm 1 also contains the honest history h^*, the security property to be analyzed, and the currently considered case case.

The function SatisfiesProperty in Algorithm 1 is called initially with the empty case to analyze all total orders. This case can be refined throughout Algorithm 1, using case splits. Hence, in the first call of the function, the set S, representing the constraints handed to an SMT solver, contains only the initial constraints C. The relevant player groups `RelevantGroups` of security property sp are set according to the stratified definitions of sp from Section 5.1: N for $w(er)i$, as we stratify over players; $2^N \setminus \{\emptyset, N\}$ for cr, as we stratify over deviating subgroups; and {"none"} for pr.

The function `ComputeSP` in line 6 of Algorithm 1 stands for `ComputeWI`, `ComputeCR` or `ComputePR`, depending on the security property sp, as summarized in [Bocevska et al. 2025a]. The result of `ComputeSP` depends on whether Γ with honest history h^* satisfies property sp for/against pg, given the constraints in S. Here, we also keep track of utility comparisons we cannot decide. Importantly, the constraint $ut_1[\vec{x}] \geq ut_2[\vec{x}]$ to whether Γ satisfies sp in `case` is returned as split_{pg}, if it exists.

The loop in lines 5–12 of Algorithm 1 incorporates player-wise security from Theorem 5.6. It additionally provides a necessary case split if the security property is violated for a player group. Subsequently, the respective results are returned: `true` for all groups yields `true`; `false` but nothing to split on for at least one group yields `false`; and `false` together with a split leads to further case splits (lines 19–24). If we split the total orders into multiple cases, all the cases have to return `true` for the property to be satisfied.

Example 6.2. Let us revisit the Market Entry game from Example 2.3, but this time let us assume only $p > 0$ is the initial constraint and $a \in \mathbb{R}$ can take any value. We check whether the honest history (n) is collusion resilient. Algorithm 1 will in line 5 consider each singleton player group individually, suppose we start with $\{M\}$. The function `ComputeCR`, specified in Algorithm 4 in [Bocevska et al. 2025a], returns $(\texttt{false}, 0 \geq -a)$, as the comparison between 0 and $-a$ is missing to determine collusion resilience. So the result is set to `false` and split to $0 \geq -a$. For the colluding group $\{E\}$ the function `ComputeCR` returns $(\texttt{true}, \text{null})$, as the honest player M can choose a collusion resilient honest action. Algorithm 1 then proceeds with line 19, refining the constraints by first adding $0 \geq -a$ to the case. The function `SafisfiesProperty` will return `true` (and empty split); and then adding the negated constraint $0 < -a$ to the case, at which point `SafisfiesProperty` will return `false`, since M can profit by deviating from the honest action (n), as both $\frac{p}{2}$ and $-a$ are better utilities than 0. Algorithm 1 thus terminates by returning `false`.

The security-property-specific function variants of `ComputeSP` recursively apply the compositional results of Theorem 5.7. To illustrate case splitting of total orders, we only describe function `ComputeWI` of Algorithm 2 below.

Algorithm 2: Function `ComputeWI`. The function `ComputeWI` of Algorithm 2 is initially called with the entire game tree Γ from function `SatisfiesProperty` of Algorithm 1. We then proceed recursively, according to Theorem 5.8. Note that the player group pg is just one player.

In a leaf, `GetUtility` in Algorithm 2 returns $ut(\vec{x})$ as the utility of player pg. We then – in line 2 of Algorithm 2 – check whether the constraints in S together with $ut(\vec{x}) < 0$ are `unsat`. This is equivalent to the constraints in S implying $ut(\vec{x}) \geq 0$, which is an instance of property (4), except that we do not (necessarily) have one total order $\preceq$ at hand, only some constraints from `case`. If the implication holds, we return `true`. Otherwise, we check the opposite condition, by asking in line 5 of Algorithm 2 whether

$$\forall \vec{x}. \bigwedge_{c \in C \cup \texttt{case}} c[\vec{x}] \rightarrow ut[\vec{x}] < 0 \tag{5}$$

holds. If it does (line 6), the leaf is not weak immune. Otherwise (line 8), the leaf's weak immunity depends on the total order, which induces a case split on $ut[\vec{x}] \geq 0$.

At a branch (lines 10–32 of Algorithm 2), we check in which of the cases of Theorem 5.8 we are. We then call the function `ComputeWI` recursively on immediate subgames $\Gamma_{|(a)}$ and propagate the

Algorithm 2: Function ComputeWI for Weak Immunity.

```
input  : game tree $\Gamma$, honest history $h^*$, set S containing initial constraints and current case, player
         group pg.
output : (result, split), where result states whether $\Gamma$ is weak immune for pg, given S, and split a
         crucial utility comparison we cannot decide.
1  if isLeaf($\Gamma$) then
2      if Check(S, GetUtility($\Gamma$, pg) $< 0$) = unsat then
3          return (true, null)
4      end
5      if Check(S, GetUtility($\Gamma$, pg) $\geq 0$) = unsat then
6          return (false, null)
7      end
8      return (false, GetUtility($\Gamma$, pg) $\geq 0$)
9  end
10 if CurrentPlayer($\Gamma$) $\neq$ pg then
11     for $a \in$ Actions($\Gamma$) do
12         (result, split) $\leftarrow$ ComputeWI($\Gamma_{|(a)}$, $h^*$, S, pg)
13         if result = false then
14             return (result, split)
15         end
16     end
17     return (true, null)
18 end
19 if AlongHonest($\Gamma$, $h^*$) then
20     $a^* \leftarrow$ HonestAction($\Gamma$, $h^*$)
21     return ComputeWI($\Gamma_{|(a^*)}$, $h^*$, S, pg)
22 end
23 newsplit $\leftarrow$ null
24 for $a \in$ Actions($\Gamma$) do
25     (result, split) $\leftarrow$ ComputeWI($\Gamma_{|(a)}$, $h^*$, S, pg)
26     if result = true then
27         return (true, null)
28     else if split $\neq$ null then
29         newsplit $\leftarrow$ split
30     end
31 end
32 return (false, newsplit)
```

result accordingly. Note that, for simplicity, in line 13 of Algorithm 2 we do not wait for a null split that would immediately return `false`, but rather proceed with a split. However, as there are only finitely many possible case splits, we will eventually see the null split for a `false` subtree if it exists and return it to reach the correct result.

Example 6.3. We mimic the execution of the function ComputeWI from Algorithm 2 on the Market Entry game from Example 2.3, but this time only assume $a > 0$ is the initial constraint, and $p \in \mathbb{R}$ can take any value. Suppose we enter Algorithm 2 with the entire tree Γ_{me}, honest history (e, i) and player $pg = M$. Since the root of the tree is along the honest history, the function will jump to line 19, and recursively call ComputeWI for the honest subtree $\Gamma_{me|(e)}$. Then the current player E is not pg, so we proceed with line 10, and iterate through the actions pw and i. Suppose we first look at the action pw and from line 12 recursively compute the weak immunity for the leaf after (e, pw). Algorithm 2 will execute lines 1 and 2, and since the utility of player M is 0, which is a non-negative number, the check in line 2 will be unsat, so the function returns (true, null). For the other action i, we recursively compute (line 12 of the algorithm) the weak immunity for the leaf after (e, i). The function GetUtility($\Gamma_{me|(e,i)}$, M) will return $\frac{p}{2}$, for which we cannot decide whether it is non-negative (there are no initial constraints on p). Both conditions from lines 2 and 5 are thus false and we return the pair (false, $\frac{p}{2} \geq 0$) in line 8. Proceeding from the supertree $\Gamma_{me|(e)}$ in line 12, with the result being false, we return in line 14 the pair (false, $\frac{p}{2} \geq 0$).

Theorem 6.4 (Correctness of Algorithm 1). *The compositional approach to compute the game-theoretic security of an input instance Π for honest history h^* described in Algorithm 1 is sound and complete. That is,* SatisfiesProperty($\Pi, h^*, sp, \emptyset$) = true *iff Π with honest history h^* satisfies the property sp. Otherwise, it returns* false.

In addition to compositional security via Algorithm 1, our work supports additional features to debug a protocol and better understand its structure. Those include (i) strategy extraction in case the considered security property was satisfied (Section 6.2), (ii) finding counterexamples (Section 6.3), and (iii) providing weakest preconditions to make the game secure otherwise. Computing preconditions in our compositional setting can be done via collecting all cases, where the security property is violated, and then conjoin and negate them afterwards.

6.2 Extracting Compositional Strategies

The way compositional security analysis in Algorithm 1 works, unfortunately, does not immediately provide witness strategies. However, Algorithm 1 can still carry around enough information to compute witnesses.

Theorem 6.5 (Weak(er) Immune Strategies). *For a weak(er) immune game Γ, with honest history h^* and total order $\preceq$, strategy σ is honest and weak(er) immune for all $\vec{x}$ satisfying $\preceq$, where*

$$\sigma := (\sigma^{p_1}, \dots, \sigma^{p_{|N|}}) \,,$$

and $\sigma^{p_i} \in \mathcal{S}_{p_i}$ is a strategy for player p_i. Strategy σ^{p_i} picks the honest choice along the honest history, whereas at other nodes, where it is p_i's turn, it picks an arbitrary action a that yields a weak(er) immune for p_i subtree after action a.

Theorem 6.5 is constructive in nature, yielding thus an algorithmic approach for extracting a weak(er) immune strategy. For each player pg, function ComputeWI (and ComputeWERI) proceeds as follows. If it is their turn after history h, h off h^*, and we found a weak(er) immune choice, we store this action as the choice of a possible weak(er) immune and honest strategy σ. If the game is weak(er) immune for all players, we can simply compute σ by collecting all the stored choices throughout the tree.

Example 6.6. We compute the weak immune strategy of the Market Entry game from Example 2.3 with honest history (n), which was analyzed as in Example 5.9. The strategy σ^M for player M has to choose the honest action n at the root, which is the only choice point for M. The strategy σ^E for player E needs to choose one weaker immune subtree after history (e). Since the subtree after history (e, i) is the only candidate, we set $\sigma^E(e) = i$. The strategy $\sigma = (\sigma^M, \sigma^E)$ is the desired weak immune strategy.

Collusion resilience and practicality also admit elegant methods for deriving compositional strategies, as detailed in [Bocevska et al. 2025a].

6.3 Finding Compositional Counterexamples

Counterexamples to the security properties, as defined in Definitions 3.12, 3.14 and 3.16, serve the important purpose of providing attack vectors and thus pinpointing the weaknesses of a protocol underlying the considered game model. We use the following *pseudo-algorithms* to compute counterexamples compositionally.

Compositional Counterexamples to Weak(er) Immunity. We first store information during Algorithm 2: When analyzing the weak(er) immunity for a player p, whenever it is not p's turn and there exists an action leading to a not weak(er) immune subtree (line 14 with split = null in Algorithm 2), we store the action, the current history and the player p.

Secondly, after the analysis terminated and the result was not weak(er) immune, we generate a counterexample to the weak(er) immunity of player p by walking through the tree again. Assume the current history is h and we proceed from the root as follows.

- If p is the current player and h is along the honest history, we follow the honest action to a subtree. This is sufficient since an honest p follows the honest history.
- If it is p's turn but h is not along the honest history, all choices had to lead to not weak(er) immune for p subtrees for the current tree to be not weak(er) immune for p. We, therefore, have to follow all choices to compute a counterexample.
- Otherwise, if it is not p's turn, we check our stored data for a not weak(er) immune for p choice a. By construction and using Theorem 5.8, it has to exist. We add it to our partial strategy s_{N-p}, i.e. $s_{N-p}(h) = a$. Then, we continue at history (h, a).
- At a leaf nothing has to be considered. A not weak(er) immune for p leaf leads to a negative (real) utility for p.

According to Theorem 5.8, the steps outlined above provide a player p and a partial strategy s_{N-p} for the other players $N-p$, no matter how the honest p behaves off the honest history. It also yields only negative utilities for p and it thus provides a counterexample to the weak(er) immunity of p and, therefore, a counterexample to the weak immunity of the game with the considered honest history.

It is also possible to compute *all* counterexamples to weak(er) immunity. This can be done by simply storing *all* actions that lead to not weak(er) immune subtrees.

Example 6.7. Let us adapt the Market Entry game from Example 2.3 by changing the initial constraint on the variable p to $p < 0$. The honest history (n) is not weak immune for player E, as they get a negative utility $p < 0$ in the honest leaf. We can thus construct the counterexample as follows: starting from the root, it is not E's turn and the not weak immune choice is (n), so we add the action n to the partial strategy for player M. We then continue at history (n), which is a leaf, so we are done.

Counterexamples to collusion resilience and practicality can also be computed, which is detailed in [Bocevska et al. 2025a].

7 Experimental Evaluation

We implemented the compositional security approach of Section 5 by exploiting its divide-and-conquer reasoning nature from Section 6. Our implementation is available online in the CheckMate2.0 tool [6].

[6] https://github.com/apre-group/checkmate/releases/tag/OOPSLA25

Experimental Setup. We evaluated our tool using a machine with 2 AMD EPYC 7502 CPUs clocked at 2.5 GHz with 32 cores and 1 TB RAM using 16 game-theoretic security benchmarks. Our dataset contains the 15 benchmarks from [Rain et al. 2024], listed in Table 3 in [Bocevska et al. 2025a], which include realistic models of real-world blockchain protocols along with game scenarios of various sizes, such as models of an auction (*Auction*) or tic-tac-toe (*Tic Tac Toe*, which is modeled infeasibly on purpose). The games *Closing*, *3-Player Routing*, *Unlocking Routing* are detailed models with up to 36,000 tree nodes of different phases of Bitcoin's Lightning protocol [Poon and Dryja 2016]. Additionally, we detail later in this section one large example (over 100 million nodes), named *4-Player Routing*, in order to showcase the impact of interleaved sub- and supertree reasoning. 4-Player Routing is yet another realistic model of a phase of the Lightning protocol.

To the best of our knowledge, the only other automated approach for game-theoretic security is the CheckMate framework [Rain et al. 2024]. Our experiments also compare CheckMate2.0 to CheckMate.

Experimental Results. Tables 1 and 2 summarize our experiments, with further details in [Bocevska et al. 2025a]. We report both on the results of CheckMate2.0 and CheckMate; the respective columns on times, nodes, and calls in Tables 1 and 2 detail these comparisons. In particular, the columns "Nodes evaluated" and "Nodes evaluated (reps)" of Table 1 indicate the number of game tree nodes visited during the security analysis without and, respectively, with repetitions. The "Calls" column of Table 1 shows the number of calls made to the SMT solver while proving the security property listed in column 4.

Table 1. Selected experimental results of game-theoretic security, using the compositional CheckMate2.0 approach and the non-compositional CheckMate setting of [Rain et al. 2024]. A full summary of our experiments is in [Bocevska et al. 2025a]. Runtimes are given in seconds, with timeout (TO) after 8 hours. For each game, columns 2–3 list the size (tree nodes and game players) of the game from column 1. Column 4 shows the game-theoretic security property we analyzed and (dis)proved, as indicated in column 5. Columns 6–9 present the results of CheckMate2.0 compared to CheckMate, using the slash / sign.

	Game	Nodes	Players	Property	Secure yes/no	Time	Nodes evaluated	Nodes evaluated (reps)	Calls
						CheckMate2.0/CheckMate			
medium-sized games	Pirate	79	4	wi	n	0.010 / 0.015	10 / 79	10 / 316	5 / 1
	$(y, n, n$			cr	n	0.041 / 0.029	79 / 79	622 / 1,106	368 / 4
	$n, y, y)$			pr	n	0.036 / 0.049	79 / 79	482 / 79	554 / 8
	Auction	92	4	wi	n	0.012 / 0.033	16 / 92	16 / 368	9 / 1
	(E, E, I, I)			cr	n	0.018 / 0.030	66 / 92	128 / 1,288	103 / 1
real-world models	Closing	221	2	wi	y	0.011 / 0.024	20 / 221	22 / 442	16 / 1
	(H)			weri	y	0.010 / 0.021	20 / 221	22 / 442	16 / 1
				cr	y	0.012 / 0.023	44 / 221	46 / 442	36 / 1
				pr	n	0.097 / 0.346	221 / 221	568 / 221	1454 / 1
	(C_h, S)			wi	y	0.011 / 0.024	33 / 221	36 / 442	25 / 1
				weri	y	0.011 / 0.020	33 / 221	36 / 442	25 / 1
				cr	y	0.013 / 0.023	60 / 221	63 / 442	48 / 1
				pr	y	2.144 / 0.345	221 / 221	14353 / 221	38220 / 1
	3-Player	21,688	3	wi	n	0.248 / 0.984	16 / 21,688	16 / 65,064	9 / 1
	Routing			weri	y	0.514 / 1.008	7,084 / 21,688	7,570 / 65,064	5,441 / 1
	$(S_H, L, L,$			cr	n	0.272 / 1.886	430 / 21,688	474 / 130,128	299 / 1
	$U, U)$			pr	n	33.162 / 34.717	21,688 / 21,688	416,156 / 21,688	569,418 / 13
	Tic Tac Toe	549,946	2	wi	y	5.276 / 255.368	18,026 / 549,946	18,036 / 1,099,892	10,694 / 1
	$(CM, RU, LU,$			weri	y	5.256 / 255.600	18,026 / 549,946	18,036 / 1,099,892	10,694 / 1
	$RD, RM, LM,$			cr	y	5.302 / 286.574	18,026 / 549,946	18,036 / 1,099,892	10,694 / 1
	$CU, CD, LD)$			pr	y	36.530 / TO	549946 / TO	549946 / TO	527198 / TO

Experimental Analysis. Table 1 demonstrates that the compositional approach of CheckMate2.0 significantly outperforms the non-compositional CheckMate setting in execution time across nearly all benchmarks. The scalability of CheckMate2.0 is especially evident in the Tic Tac Toe benchmark, which involves a substantial 548,946 nodes. In this example, for the properties weak immunity (`wi`), weaker immunity (`weri`), and collusion resilience (`cr`), CheckMate2.0 completes the security analysis in approximately 5 seconds, whereas CheckMate requires between 255 and 287 seconds. When proving practicality (`pr`) of Tic Tac Toe, the conventional CheckMate fails to terminate within 8 hours while CheckMate2.0 succeeds in less than 37 seconds.

In some benchmarks, where a security property is not satisfied, CheckMate2.0 explores significantly fewer nodes, see 3-Player Routing for weak immunity and collusion resilience, the Pirate game for weak immunity, and Auction for weak immunity and collusion resilience.

We note that CheckMate2.0 requires considerably more SMT-solving calls. Notable examples include the Closing Game (38,220 CheckMate2.0 calls vs. 1 CheckMate call for practicality), 3-Player Routing (546,418 vs. 13 calls for practicality), and Tic Tac Toe (10,694 vs. 1 call for weak(er) immunity and collusion resilience). Despite the higher number of SMT calls in CheckMate2.0, the SMT queries generated by CheckMate2.0 are considerably smaller than the ones of CheckMate; moreover, CheckMate2.0 calls inhabit a quantifier-free fragment, easing reasoning significantly as reflected in the improved execution times.

In general, CheckMate2.0 analysis may occasionally result also in suboptimal splits, leading to longer execution times. This issue is exemplified in the Closing game when analyzing practicality of the honest history (C_h, S). Additionally, analyzing collusion resilience can sometimes take longer, particularly when more players are involved, for example in the Pirate game. This might be explained by the very large number of colluding groups combined with a small game resulting in many trivial SMT calls compared to CheckMate.

Table 2. Selected experiments on counterexample (CE) generation using our CheckMate2.0 approach and the non-compositional CheckMate tool of [Rain et al. 2024]. Full details and experiments are given [Bocevska et al. 2025a]. Runtimes are given in seconds.

Game	Property	Time (one CE) CheckMate2.0/CheckMate	Time (all CEs) CheckMate2.0/CheckMate
Pirate (y, n, n, n, y, y)	cr	0.041 / 0.039	3.232 / 79.839
Auction (E, E, I, I)	wi	0.012 / 0.048	0.025 / 4.172
	cr	0.018 / 0.066	0.036 / 15.106
3-Player Routing (S_H, L, L, U, U)	wi	0.251 / 1.925	5.909 / 110.716
	cr	0.279 / 5.619	1.657 / 7.815
	pr	33.561 / 46.480	291.236 / 3 033.784

Counterexamples and Strategies. Table 2 presents the CheckMate2.0 runtimes to generate counterexamples compared to CheckMate, for selected benchmarks. It reports the execution time required to find one counterexample (for one case split) as well as finding all counterexamples in all cases for violated security properties. The former is useful for quickly identifying scenarios where the property is not met, while the latter proves particularly helpful when revising and refining a protocol. Comprehensive data for all benchmarks can be found in [Bocevska et al. 2025a].

The use of compositionality in CheckMate2.0 demonstrates notable improvements in execution time, particularly when retrieving all counterexamples. Additionally, the execution times for compositional analysis with and without counterexample extraction are quite similar, indicating that CheckMate2.0 enables counterexample extraction with minimal overhead. The counterexamples to collusion resilience for the Pirate game show this clearly. While CheckMate2.0 requires slightly more time for property analysis compared to CheckMate, we note that the new CheckMate2.0 identifies all counterexamples across all cases in approximately 3 seconds, whereas CheckMate takes almost 80 seconds. Similarly, in the case of the 3-Player Routing game, CheckMate2.0 retrieves all counterexamples for all cases within 291 seconds, while it takes CheckMate over 3,000 seconds (50 minutes).

Similar benefits of CheckMate2.0 can also be observed for strategy extraction, with details in [Bocevska et al. 2025a].

Sub- and Supertree Reasoning. One of the most significant contributions of the compositional reasoning is that CheckMate2.0 enables analyzing subtrees independently and integrating only their security results in the supertree. This feature of CheckMate2.0 is particularly beneficial in larger models. For instance, the 3-Player Routing and Routing Unlocking benchmarks based on the routing protocol [Poon and Dryja 2016] are generated using a script, as it is not feasible to model protocols of this size manually. Modeling the routing protocol for 3 players results in a game with 21,688 nodes (3-Player Routing), taking 20 MB on disk.

We next detail a more challenging routing example with 4 players, called *4-Player Routing*, which has 144,342,306 nodes. This example exceeds our 200 GB of allocated disk space, and thus could not even be created fully. However, by leveraging compositionality, we intertwine model generation and analysis, making it possible to discard generated subtrees after the results of security analysis have been obtained. Specifically, during the game generation process, each subtree corresponding to a specific phase of the protocol called unlocking phase (a total of 1440 subtrees) is analyzed on the fly, with only the results kept. The final outcome, the *4-Player Routing* game, is a supertree with 396 regular nodes and 1440 nodes representing subtrees, or 1,836 nodes in total. The supertree has a size of about 60 MB and in it all subtrees for the unlocking phase have already been solved. This allows us to directly apply CheckMate2.0 to the supertree. Using CheckMate2.0 compositionally, we conclude that 4-Player Routing is weaker immune, but not weak immune, nor collusion resilient, nor practical.

We note that game modeling is not in the scope of this paper, but interleaving the modeling and the analysis of a game, as described in the paragraph above, makes it feasible to verify even complex real-world models with over 100 million nodes.

8 Related Work and Conclusions

We present the first approach to compositionally analyze the security properties of game-theoretic protocol models. By mapping our work to SMT-based reasoning in combination, we introduce a divide-and-conquer framework to automate compositional reasoning in a sound and complete manner. Our experiments clearly showcase scalability improvements, especially for real-world protocols with millions of nodes/actions.

We believe that our compositional reasoning framework generalizes to other tree search properties that are (in-)equations of utility terms quantified over strategies, histories, and player groups, provided the property can be analyzed player(group)-wise. By memorizing some subtree data, it should be possible to decide such a property given all subtree results. Other common game-theoretic properties, such as Nash Equilibria for example, are in fact covered by our security properties and are thus guaranteed to be of compositional nature.

Our compositional approach is a strong enhancement over the non-compositional setting of [Brugger et al. 2023]. While the definitions of the security properties are the same, their encoding is different: we not only improve practical usage but also provide a sound and complete way to split and combine game-theoretic properties of subgames/supergames. Compared to [Brugger et al. 2023], we minimize the use of SMT solving by applying it only over game leaves.

Compositional game theory, without considering game-theoretic security, is also addressed in [Bolt et al. 2023; Ghani et al. 2018, 2020]. Here, so-called open games are introduced to represent games played relative to a given environment. Open games are, however, restricted to constant numeric utilities and assuming rational behavior of players. Unlike these works, we work with symbolic utilities and capture honest/rational behavior, and thus security, in games.

Divide-and-conquer approaches to parallelize SMT queries have also been studied [Wilson et al. 2023], proposing partitioning strategies to decompose an SMT problem into subproblems. Our approach avoids generating large SMT queries altogether, utilizing instead local tree reasoning.

Related to compositional verification, [Wesley et al. 2021] presents compositional analysis of smart contracts. Instead of verifying a smart contract relative to all users, a few representative users are chosen, thereby avoiding intractability due to state explosion. While game-theoretic security is not addressed in [Wesley et al. 2021], program verification and synthesis are worthy approaches to be further considered in our future work.

Importantly, (automatically) synthesizing game models from the protocol's definition, respectively source code in the case of smart contracts, is a challenge we aim to address in the future. Allowing infinite games and modeling game actions impacted by external factors are other tasks for future work, allowing us to model uncontrollable protocol effects, such as price changes.

9 Data-Availability Statement

The software that implements the techniques described in Section 5 and Section 6 and supports the evaluation results reported in Section 7 is available on Zenodo [Bocevska et al. 2025b].

Acknowledgments

This research was funded in whole or in part by the ERC Consolidator Grant ARTIST 101002685, the Austrian Science Fund (FWF) SPyCoDe Grant 10.55776/F85, the WWTF Grant ForSmart 10.47379/ICT22007, the TU Wien Doctoral College SecInt, the Amazon Research Award 2023 QuAT, and a Netidee Fellowship 2022.

References

Clark Barrett and Cesare Tinelli. 2018. Satisfiability Modulo Theories. In *Handbook of Model Checking*, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). Springer International Publishing, Cham, 305–343. https://doi.org/10.1007/978-3-319-10575-8_11

Nikolaj Bjørner and Lev Nachmanson. 2024. Arithmetic Solving in Z3. In *Computer Aided Verification: 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24–27, 2024, Proceedings, Part I* (Montreal, QC, Canada). Springer-Verlag, Berlin, Heidelberg, 26–41. https://doi.org/10.1007/978-3-031-65627-9_2

Sam Blackshear, Nikos Gorogiannis, Peter W. O'Hearn, and Ilya Sergey. 2018. RacerD: compositional static race detection. *Proc. ACM Program. Lang.* 2, OOPSLA (Oct. 2018), 28 pages. https://doi.org/10.1145/3276514

Bruno Blanchet. 2014. Automatic Verification of Security Protocols in the Symbolic Model: The Verifier ProVerif. In *Foundations of Security Analysis and Design VII: FOSAD 2012/2013 Tutorial Lectures*, Alessandro Aldini, Javier Lopez, and Fabio Martinelli (Eds.). Springer International Publishing, Cham, 54–87. https://doi.org/10.1007/978-3-319-10082-1_3

Ivana Bocevska, Anja Petković Komel, Laura Kovács, Sophie Rain, and Michael Rawson. 2025a. Divide and Conquer: a Compositional Approach to Game-Theoretic Security. EasyChair Preprint 15785, https://easychair.org/publications/preprint/kxKK.

Ivana Bocevska, Anja Petković Komel, Laura Kovács, Sophie Rain, and Michael Rawson. 2025b. Artifact Evaluation CheckMate. https://doi.org/10.5281/zenodo.15725152

Joe Bolt, Jules Hedges, and Philipp Zahn. 2023. Bayesian open games. *Compositionality* 5 (Oct. 2023), 9. https://doi.org/10.32408/compositionality-5-9

Lea Salome Brugger, Laura Kovács, Anja Petkovic 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 (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 1407–1421. https://doi.org/10.1145/3576915.3623183

Vitalik Buterin. 2014. *A Next Generation Smart Contract & Decentralized Application Platform*. Technical Report. Ethereum Foundation. Issue 37. https://ethereum.org/en/whitepaper/

Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. 2018. Compositional Game Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science* (Oxford, United Kingdom) *(LICS '18)*. Association for Computing Machinery, New York, NY, USA, 472–481. https://doi.org/10.1145/3209108.3209165

Neil Ghani, Clemens Kupke, Alasdair Lambert, and Fredrik Nordvall Forsberg. 2020. Compositional Game Theory with Mixed Strategies: Probabilistic Open Games Using a Distributive Law. *Electronic Proceedings in Theoretical Computer Science* 323 (Sept. 2020), 95–105. https://doi.org/10.4204/eptcs.323.7

Nadim Kobeissi, Georgio Nicolas, and Mukesh Tiwari. 2020. Verifpal: Cryptographic Protocol Analysis for the Real World. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop* (Virtual Event, USA) *(CCSW'20)*. Association for Computing Machinery, New York, NY, USA, 159. https://doi.org/10.1145/3411495.3421365

Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Computer Aided Verification*. Springer Berlin Heidelberg, Berlin, Heidelberg, 696–701. https://doi.org/10.1007/978-3-642-39799-8_48

Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/en/bitcoin-paper

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

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

Sophie Rain, Georgia Avarikioti, Laura Kovács, and Matteo Maffei. 2023. Towards a Game-Theoretic Security Analysis of Off-Chain Protocols. In *2023 IEEE 36th Computer Security Foundations Symposium (CSF)*. IEEE Computer Society, Los Alamitos, CA, USA, 107–122. https://doi.org/10.1109/CSF57540.2023.00003

Sophie Rain, Lea Salome Brugger, Anja Petković Komel, Laura Kovács, and Michael Rawson. 2024. Scaling CheckMate for Game-Theoretic Security. In *Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning (EPiC Series in Computing, Vol. 100)*, Nikolaj Bjørner, Marijn Heule, and Andrei Voronkov (Eds.). EasyChair, Stockport, UK, 222–231. https://doi.org/10.29007/llnq

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

Yuepeng Wang, Shuvendu K. Lahiri, Shuo Chen, Rong Pan, Isil Dillig, Cody Born, Immad Naseer, and Kostas Ferles. 2020. Formal Verification of Workflow Policies for Smart Contracts in Azure Blockchain. In *Verified Software. Theories, Tools, and Experiments*, Supratik Chakraborty and Jorge A. Navas (Eds.). Springer International Publishing, Cham, 87–106. https://doi.org/10.1007/978-3-030-41600-3_7

Scott Wesley, Maria Christakis, Jorge A. Navas, Richard Trefler, Valentin Wüstholz, and Arie Gurfinkel. 2021. Compositional Verification of Smart Contracts Through Communication Abstraction. In *Static Analysis: 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17–19, 2021, Proceedings* (Chicago, IL, USA). Springer-Verlag, Berlin, Heidelberg, 429–452. https://doi.org/10.1007/978-3-030-88806-0_21

Amalee Wilson, Andres Nötzli, Andrew Reynolds, Byron Cook, Cesare Tinelli, and Clark W. Barrett. 2023. Partitioning Strategies for Distributed SMT Solving. In *Proceedings of the 23rd Conference on Formal Methods in Computer-Aided Design*, Vol. 4. TU Wien Academic Press, Vienna, Austria, 199–208. https://api.semanticscholar.org/CorpusID:259129858

Paolo Zappalà, Marianna Belotti, Maria Gradinariu Potop-Butucaru, and Stefano Secci. 2020. Game theoretical framework for analyzing Blockchains Robustness. https://api.semanticscholar.org/CorpusID:219616790

Received 2025-03-26; accepted 2025-08-12