# Game Modeling of Blockchain Protocols

Sophie Rain[1], Anja Petković Komel[1], Michael Rawson[2], and Laura Kovács[3]

[1] Argot Collective, Switzerland
[2] University of Southampton, UK
[3] TU Wien, Austria

**Abstract.** Reasoning about incentives in a blockchain protocol can be captured by game-theoretic modeling. We present modeling principles sufficient to create a faithful representation of a blockchain protocol as an extensive form game. Such games are then suitable for automatically establishing game-theoretic security. We showcase the semi-automated generation of the game models for two parts of Bitcoin's Lightning protocol: the closing of a channel and the routing of a payment along channels. Additionally, we provide a domain-specific language, which eases the implementation of the games. We believe our modeling principles and guidelines strengthen machine-supported modeling practices.

**Keywords:** Game Theory · Formal Models · Modeling Template · Protocol Modeling.

## 1 Introduction

Users of decentralized economic systems, such as cryptocurrencies, demand security guarantees of the system as a whole. This includes *cryptographic* security [1,18,13], which ensures that the cryptographic premises of the protocol are secure, *implementation* security [12,6,4], which ensures the implementation of the protocol does not allow any unexpected behavior, and *game-theoretic* security [25,2], which ensures that cryptographically and implementationally possible but undesired behavior, such as collusion, is economically disincentivized. This paper focuses on the rigorous modeling of protocols as games, in order to automatically assess their game-theoretic security.

To say that a protocol is *game-theoretically secure* means that it has various desirable properties. Notably, we are interested in capturing that protocol users/participants cannot be economically harmed, and that a group of players cannot collaborate to gain an advantage. These properties are called *Byzantine-fault tolerance* and *incentive compatibility* [20] and revised in [2]: These properties are implemented through the game-theoretic concepts *weak immunity*, respectively *weaker immunity*, for Byzantine-fault tolerance, and *collusion resilience* and *practicality* for incentive compatibility. Manually checking that a game has a particular game-theoretic property is, however, tedious and often not viable. Protocols may allow many possible options at each step, producing game trees with millions of paths, as showcased in [2]. To overcome the burden of manual and notoriously error-prone protocol analysis, automated approaches arose to check the game-theoretic properties for a given protocol [2], which work – in essence – by an exhaustive symbolic enumeration of all paths through the game tree. In this setting, game-theoretic security derived by an automated formal approach increases user trust in the protocol under analysis.

Tacit in game-theoretic security is the expertise and work required to model a protocol as a game rather than only as a list of specifications. *Extensive form games* (EFGs) turn out to offer suitable expressivity for modeling protocols using a tree-like data structure capturing actions between protocol users. Ensuring, though, that protocol requirements are best represented by EFGs is, however, challenging: it is reminiscent of the formalization of mathematics in that hidden details must be recovered, but with the added challenge of managing very large game trees. To the best of our knowledge, game-theoretic modeling of protocols has remained mostly manual to date, partially due to the fact that – until recently – automated methods did not scale to very large trees found in real-world protocols.

Our paper addresses this key missing bit in game-theoretic security: *provide (semi-)automated methods to represent protocols as games*, upon which game-theoretic analysis can be performed. We present a network of techniques (Section 3) to partially automate and accelerate the development of game models, while increasing trust in their construction. Using these techniques, we have successfully encoded very large games, such as those required to model a phase of Bitcoin's Lightning protocol [19], and the large proprietary protocol FAsset [7] on the *Flare* blockchain [8] for digital currency exchange. Our resulting models were then handed to the automated game-theoretic security tool CheckMate [21], as this framework allows automatically analyzing the Byzantine-fault tolerance and incentive compatibility of game models of very large protocols, and further supports models containing *conditional actions* (Section 7).

In summary, the main contributions of our paper come with (i) providing general principles guiding the automated synthesis of games from protocols (Section 3); (ii) introducing a domain specific language for game modeling (Section 4); and (iii) illustrating the generation of games from protocols on various examples (Section 5).

**Related Work.** The modeling principles described in Section 3 rely on the definition of an EFG as a suitable model for the game-theoretic security analysis as introduced in [20,25], that can be automatically processed by CheckMate [2,21]. While [20,25] introduce manually made models, our semi-automated approach enables exhausting all parameter options systematically in a machine supported manner, and thus refines these models, making fewer assumptions on the way.

Extensive form games can be translated to Open games with agency [3,10,11]. The utilities in Open games are restricted to constant numeric utilities, and rational behavior of players is assumed. We, however, work with symbolic utilities and capture honest/rational behavior, and hence game-theoretic security. PRISM-games [14], offer a modeling language for concurrent stochastic multi-player games (CSGs), thus capturing probabilistic behavior. They are also limited to constant numeric utilities, as opposed to the symbolic ones.

The manual game-theoretic models as static noncooperative games, such as a model for shard-based permissionless blockchains [16], the griefing attack in blockchain mining [5,17], the mining strategy for Bitcoin-NG blockchain protocol [24], the reward distribution in Algorand [9], use many of the modeling principles presented in Section 3. The authors of these models detail the assumptions and design decisions in forming a static game model. They are also able to use symbolic utilities in their (manual) game-theoretic security analysis. As opposed to the approach described in the present paper, these games were modeled manually and are not amenable to automatic processing.

## 2 Preliminaries

This section introduces relevant aspects for game-theoretic *modeling* of (blockchain) security, complementing the automated *verification* of game-theoretic security, and in particular the CheckMate framework [2].

### 2.1 Protocols

Blockchain protocols are a natural use for game-theoretic security, and in particular for the CheckMate framework [2], as they formalize economic incentives within strict formal rules. We, therefore, introduce an interesting Bitcoin protocol called Lightning [19] as a running example. It enables users to safely send numerous transactions while only having to publish two of them on the blockchain. This saves transaction fees and time. The protocol is based on so-called channels, each connecting two users. The part of the protocol we focus on is called *routing*: it enables Lightning users to *route* money from a user $A$ to another user $B$ (who do not share a channel) along a path of channels.

$$\begin{tikzcd}
A \arrow[r, "{(m+3f,y,t_1)}", "(2)"'] & I_1 \arrow[l, bend left=60, "x", "(9)"'] \arrow[r, "{(m+2f,y,t_2)}", "(3)"'] & I_2 \arrow[l, bend left=60, "x", "(8)"'] \arrow[r, "{(m+f,y,t_3)}", "(4)"'] & I_3 \arrow[l, bend left=60, "x", "(7)"'] \arrow[r, "{(m,y,t_4)}", "(5)"'] & B \arrow[l, bend left=60, "x", "(6)"'] \arrow[llll, bend right=30, "y"', "(1)"]
\end{tikzcd}$$

**Fig. 1.** Routing in Lightning, where hash$(x) = y$.

*Example 1 (Routing in Lightning).* Figure 1 illustrates the routing of transactions from a user $A$ to a user $B$ with the three intermediaries $I_1$, $I_2$, and $I_3$, where all *neighbors* (i.e. $A$ and $I_1$, $I_1$ and $I_2$, etc.) have a *Lightning* channel. To route money from $A$ to $B$ using Lightning, $B$ has to (1) define a secret $x$ and send its hash value $y$ to $A$. Next, (2) $A$ locks some amount in the channel with $I_1$, which can only be unlocked with the secret $x$ before a timeout $t_1$. The locked amount should be the amount $m$ to be sent to $B$ plus a fee $f$ for each intermediary. The intermediaries proceed accordingly in steps (3–5), each reducing the amount by $f$. Then (6), as $B$ knows the secret, they can unlock the money in their channel with $I_3$, thereby revealing the secret to $I_3$. Knowing the secret $x$, the intermediaries continue to unlock the money in their respective channels (7–9). For the sake of simplicity, in this paper we focus mostly on the *unlocking phase*, i.e., steps $(6-9)$. Steps 1–5 are referred to as the *locking phase*.[4].

*Closing a Lightning Channel.* As mentioned above, the Lightning protocol is based on channels that each connect two users. Such a channel can be closed at any time by the users, releasing the money that was locked inside. Either user can unilaterally close the channel by posting the latest distribution state they agreed upon on the blockchain; we refer to this as *honest unilateral closing.* However, a user can also take an outdated distribution state and publish it on the blockchain to close the channel; we call this *dishonest unilateral closing.* After dishonest unilateral closing, the other channel user has the option to prove that the posted distribution state was outdated and evoke a punishment mechanism that keeps the other one from getting their assets. This is done through a so-called *revocation transaction.*

Further, the channel users can also collaboratively close their channel: this can be achieved through both signing and publishing the same transaction that distributes their assets. The distribution transaction proposed by one of them can be both honest (using the values of the latest agreed distribution state) or dishonest (using any other distribution values). The other user has to decide whether or not they agree to the distribution by signing or not signing the transaction. If none of the users ever close the channel in any way, their money will stay locked forever.

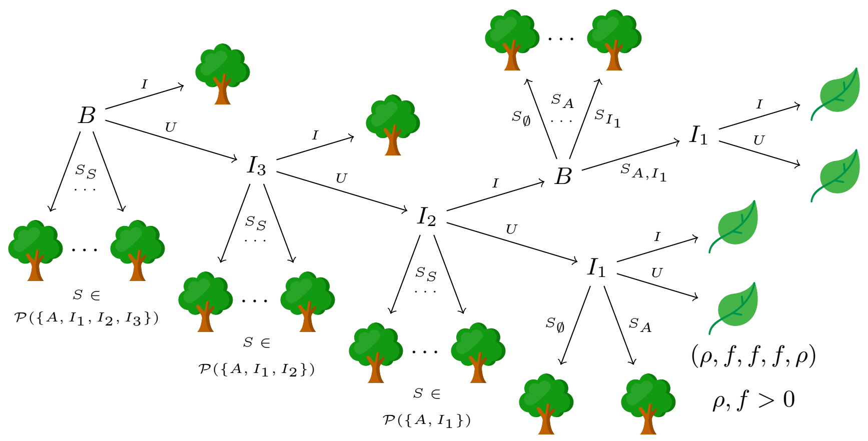
### 2.2 Game-Theoretic Concepts and CheckMate Input Structure

As detailed in [21], game-theoretic security verification relies on inputs given as *extensive form games* (EFGs) with *symbolic utilities.*

**Definition 1 (Extensive Form Game –EFG).** *An* extensive form game (EFG) *is a finite tree $G$ together with a finite set of players $N$, where*

- *each path in $G$ that starts from the root is called* history*;*
- *each internal node has a* player *– the one whose turn it is – assigned;*
- *the set of edges at each internal node is called* actions *and are the options the assigned player can choose from;*

[4] The model of the full routing protocol and its generation code are available at [22].



$B$
$I$
$U$
$s_S$
$s \in \mathcal{P}(\{A, I_1, I_2, I_3\})$
$I_3$
$s \in \mathcal{P}(\{A, I_1, I_2\})$
$I_2$
$s \in \mathcal{P}(\{A, I_1\})$
$I_1$
$s_\emptyset$
$s_A$
$s_{I_1}$
$s_{A,I_1}$
$(\rho, f, f, f, \rho)$
$\rho, f > 0$

**Fig. 2.** Sketch of Lightning's Routing Unlocking phase. Tree icons by Freepik - Flaticon.

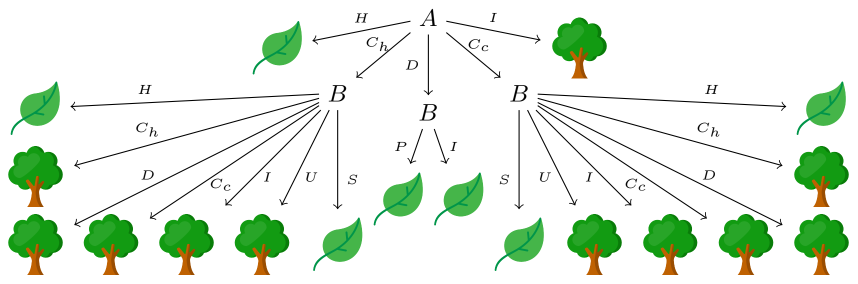


**Fig. 3.** Sketch of Lightning's Closing phase. Tree icons by Freepik - Flaticon.

– *each leaf has a* utility *assigned and represents a possible end of the game. The utility specifies the pay-off for each player in $N$ after this history (root to this leaf);*
– *there is at least one history to a leaf that represents the expected behavior of the underlying protocol, called* honest history.

*Example 2 (Routing EFG).* Figure 2 illustrates an EFG modelling *routing unlocking* in Figure 1. At the root, it is player $B$'s turn to choose between the actions $U$ (unlocking), $I$ (ignoring), or one of the $S_S$ choices, representing $B$'s option to share their secret $x$. We assume history $(U, U, U, U)$ to be the honest history leading to utility $\rho$ for $A$ and $B$ and $f$ for the intermediaries $I_1$, $I_2$, $I_3$.

*Example 3 (Closing EFG).* Figure 3 shows an EFG model of closing a Lightning channel. We call the two users of a channel player $A$ and player $B$. Without loss of generality, we assume player $A$ has the first turn. Players can choose between action $H$ (closing unilaterally and honestly), action $C_h$ (closing the channel honestly and collaboratively, which requires player $B$ to react), action $D$ (dishonest unilateral closing, allowing $B$ to post a revocation transaction – action $P$ – or to ignore it – action $I$), action $C_c$ (dishonest collaborative closing, again requiring $B$ to react), or action $I$ (not closing the channel).

If player $A$ chose action $C_h$ or $C_c$, player $B$ gets to react. They have the same choices as player $A$ before, but they additionally can pick action $U$ (propose an update to the channel) or action $S$ (signing player $A$'s closing proposal).

We fix both ways of closing honestly: history $(H)$ and history $(C_h, S)$ to be honest histories.

The *symbolic utilities* are terms with two kinds of variables: *constants* and *infinitesimals*. Both types are interpreted over the reals, but the *infinitesimals* are assumed to be closer to 0 than any of the *constants* and are supposed to represent subjective motivation. This is achieved by interpreting utilities as constant-infinitesimal pairs, which are ordered lexicographically, thereby achieving infinitesimality. Further, to allow for more realistic models, the values of the constants and infinitesimals can be restricted through *initial constraints*.

*Example 4 (Symbolic Utilities).* The utility after the honest history in Figure 2 contains one constant $f$ and one infinitesimal $\rho$. For each variable, its type has to be specified. For example, $A$'s utility $\rho$ will be interpreted as the pair $(0, \rho)$ since it does not contain any *constant* variable. Further, it is assumed in Figure 2 that both $\rho$ and $f$ are positive, hence the initial constraints $\rho > 0$, $f > 0$.

## 3 Modeling Principles for Game-Theoretic Security

Usually, several EFGs can serve as game-theoretic models for a blockchain protocol, so that they faithfully represent the protocol itself and adequately capture its security properties. While modeling protocols, we are therefore making some assumptions and design choices (see Section 3.3) that allow us to construct one such finite model of a possibly infinite protocol. These design choices and assumptions have to be thoroughly documented to ensure transparency of the model's limitations with regard to how accurately it corresponds to the protocol (and its potential implementation).

In this section, we present our guidelines towards faithful game-theoretic modeling of (blockchain) protocols. Our modeling principles yield a possible approach that was feasible to construct sufficient models for the CheckMate framework. To ensure faithful representations of protocols that are also aligned with the presuppositions on the CheckMate inputs, our models have the following three properties:

1. **Relative Utilities.** The utilities awarded to each player are relative to what they were assumed to have initially.
2. **Ghosting.** At every internal node of the game tree, it is possible to do nothing, i.e., to not respond in any way. It is crucial to account for such behavior, as it can easily happen in the decentralized and pseudonymous setting.

3. **Actual Choice.** At every internal node, there have to be at least two actions available. It is not possible to have just one, as this player does not have an actual choice.

*Example 5 (Model Properties).* In Example 2, *relative utility* means that player $A$ in *Routing Unlocking* does not receive utility $-m$ in the honest case, even though $A$'s balance is decreased by a value of $m$; instead, they receive an infinitesimal but positive utility $\rho$. This is due to our assumption that there is a fair trade of an asset in some form for Bitcoin, giving $A$ some form of benefit. Otherwise, $A$ would not route the money to the player $B$.

Next, *ghosting* means that at every internal node in the game tree, there has to be an "ignore" action that corresponds to inaction. For readability, it is useful to specify what task the current player is ignoring. In *Routing Unlocking*, players can "ignore to unlock" their HTLC (action $I$) or choose not to share their secrets with others, which we call "ignore the option to share a secret" (action $S_\emptyset$). In the *Closing game* players also have an "ignore" action, signifying doing nothing.

At every point in time in *Routing Unlocking* and in *Closing game*, there are always at least two available choices, doing something or ignoring to do something, ensuring *actual choice.*

## 3.1 Game Modeling – Setup

We next summarize the setup we propose and use for modeling (blockchain) protocols as games. Our modeling principles listed below form instructions on how to model a decentralized protocol as an EFG, which we further translate into a modeling template as discussed next; for concrete design/implementation details on our template we refer to [22].

*Define Players.* Fix the number and names of players. Each agent who can make active choices in the protocol or impact it through their choices should be represented as a player.

*Example 6 (Modeling Players).* In Example 2 of Lightning's routing, we considered the player who wants to route money to another one, called $A$; the one who is supposed to receive the routed value, called $B$; and we chose to study the case where we have 3 intermediaries, named $I_1$, $I_2$ and $I_3$. In our modeling implementation, this is defined as

```
PLAYERS = players('A', 'I1', 'I2', 'I3', 'B')
```

In the Closing game of Example 3, we have two players, $A$ and $B$, who share a channel.

We note that, within a protocol, there is potentially an unspecified number of players. It is then possible to create a model for each number instance of players. For example, the routing protocol of Example 2 has three intermediaries, but could have also just had one, two, or even more players (four, five, etc.). A good representation of players, therefore, depends on the protocol itself, and the number of players should be documented as an explicit assumption on the model.

*Exhaust Parameter Options.* One has to pay attention to the parameters in a protocol and vary them in all combinations exhaustively. Within CheckMate, all possible real values in players' utilities can be considered by construction, as CheckMate supports symbolic utilities. In addition to symbolic utilities, other parameters also need to be varied. For example, in our routing protocol from Example 2, relevant parameters include time (and time-outs), secret sharing, wrong addresses, wrong amounts, and all other possible ways of acting, including locking funds using a wrong secret hash.

Further, one also has to separate which model parameters can be tackled through symbolic utilities and which have to be studied as distinct actions that players can take. For instance, locking the wrong amount will introduce a new symbol for the utilities to the model (the amount to lock), but also a different action from locking the correct amount. In the closing protocol of Example 3, closing dishonestly also introduces a new symbol: the proposed cheating factor. Note that some parameters require additional constraints and assumptions, such as the symbol representing the wrong amount locked being different from the correct one, see Section 3.3.

*Define a State.* It is necessary to identify the values, storage slots, and facts about previous choices that suffice to compute utility in each EFG leaf. Constructing a state to keep track of these values/facts throughout the game tree generation is equally important. A state, then, usually includes at least all the varied parameters mentioned before. We define the initial values in the state and define a deep copy function. Note that during the design of EFG tree generation, the state might need to be refined with more information to ease the tree generation process.

*Example 7 (Modeling States).* For Routing unlocking in Example 2, our state contains the following information for each player: whether the contract they can possibly unlock is `locked`, `unlocked`, or `expired`, the amount locked in this contract `amountToUnlock`, whether they know the secret to unlock the contract `secret`, and with whom they have decided not to share the secret with (`ignoreshare`). Additionally, we keep track of whether player $B$ has ever shared the secret (`BShared`), as any participation from player $B$ requires them to send the goods to $A$. In our modeling template, we store the Routing unlocking state as a Python dictionary as follows:

```
initial_state = {"B_shared": False}
for player in PLAYERS:
  initial_state[player] = {}
  initial_state[player]["contract"] = "locked"
  initial_state[player]["amount_to_unlock"] =
                     m + (len(PLAYERS)-PLAYERS.index(player)-1)*f
  initial_state[player]["secret"] = False
  initial_state[player]["ignoreshare"] = {p: False for p in PLAYERS}
initial_state[PLAYERS[-1]]["secret"] = True
```

```
initial_state[PLAYERS[0]]["contract"] = "null"
initial_state[PLAYERS[0]]["amount_to_unlock"] = None
```

For the Closing game of Example 3, the model state contains for each player whether they closed unilateraly (if yes, then value by which they tried to enrich themselves, 0 if honest), published a revocation transaction, made a collaborative attempt (if yes, then value by which they tried to enrich themselves, 0 if honest), signed a collaborative closing, proposed an update (if yes, then value by which current balance of player changes), or agreed to an update. We also keep their current `balance`.

## 3.2 Generate the Game Tree of the Model

*Define Final States.* We fix the criteria under which an EFG is over, that is, when a final state is reached. In the routing phase of Example 2, this is when no contract is locked anymore, i.e., when all contracts are either expired or unlocked. Our model template uses a `isFinal` function to decide this, as follows

```
def is_final(state):
for p in PLAYERS:
    if state[p]["contract"] == "locked":
        return False
return True
```

*Define the Utility Function.* For every final state, that is, at each EFG leaf, the utility for each player has to be defined. The information in the state should suffice for this step. In the Routing unlocking protocol of Example 2, an intermediate player (I1 - I3) gets the utility of the amount in the contract they unlocked minus what they locked in their contract and the previous player unlocked. Players $A$ and $B$ are expected to make a fair trade of money and goods, which, in case it goes through (when $B$ participates in the unlocking through either unlocking $I_3$'s contract or sharing the secret), leads to a small positive utility $\rho$ for both $A$ and $B$. Our modeling template suggests a function `compute_utility(state)`.

*Collect Actions.* We collect all possible actions in the protocol and decide in which scenarios they are possible. Note that often variations of parameters can be grouped into a small set of actions depending on what impact choosing a specific value for a variable has. In our model template, the actions are collected in `ACTIONS` and the tree is produced by a `generate_tree` function.

*Example 8 (Model Actions).* In the Routing unlocking protocol of Example 2, besides the always-present action of ignoring ($I$), there is also unlocking ($U$) when the player knows the hashed secret, and sharing said secret with other players. For simplicity, we model secret sharing by each player choosing a subset of other players, who do not know the secret yet, and with whom they share it. An empty subset thus corresponds to ignoring sharing the secret.

For the Closing protocol of Example 3, the actions are listed in Example 3.

*Fix Player Precedence.* Next, an ordering of players has to be fixed, such that we can decide which player's turn it is at every point in the game. That means their precedence has to be established. In Example 2, we decided to give precedence according to the following three criteria:

– **Priority 1**: The next player is the one with the next time-out (= right-most in Figure 1) who knows the secret and whose current state of the contract is locked.
– **Priority 2**: The next player is the one with the earliest time out who can share the secret.
– **Priority 3**: If there is no such player, all locked contracts expire.

Our modeling template suggests defining the next player in a dedicated function `NextPlayer`, given below.

```
def next_player(state):
    # prio1:
    for p in PLAYERS[::-1]:
        if state[p]["secret"] and state[p]["contract"] == "locked":
            return p, state

    # prio2:
    for p in PLAYERS[::-1]:
        if state[p]["secret"]:
            for share_with in PLAYERS:
                if (not state[p]["ignoreshare"][share_with] and
                not state[share_with]["secret"]):
                    return p, state

    # prio3:
    state1 = copy_state(state)
    for p in PLAYERS:
        if state1[p]["contract"] == "locked":
            state1[p]["contract"] = "expired"
    return None, state1
```

## 3.3 Declare Assumptions and Design Choices

Some of the modeling steps from Sections 3.1–3.2 require making assumptions. One has to pay special attention to such assumptions and document them thoroughly. Some assumptions only need to be documented, such as choosing to require player $B$ in the Routing game of Example 2 to send the goods in case they share the secret. Others, like assumptions on utility variables (e.g. $f > 0$), have to be explicitly listed as an `INITIAL CONSTRAINT`. Design choices usually also influence the shape and the size of the game tree: in the Routing unlocking of Example 2 the choice of how we set priorities for the player precedence enables us to model a protocol where in practice simultaneous actions are possible (sharing secrets could in principle happen at the same time), but the EFG does not allow for it.

## 4 Domain Specific Language for Game Modeling

Section 3 summarizes our guidelines to model protocols as EFGs. Our modeling template is designed such that the resulting (EFG) model is parsable by our CheckMate tool, which takes a '.json' file as input. To ease the process of model generation in a suitable format, we designed a simple domain-specific language (DSL), written in Python, that enables game tree generation. The usage of this DSL is discussed next and exemplified in the modeling template available [22].

To handle real-valued expressions in the utilities of the EFG players, appearing in the leaves of the game trees, and the constraints imposed on the symbols, our DSL introduces a class of expressions `Expr`, which includes a `NameExpr` to handle the symbols. The class overloads the usual arithmetic operators (+, -, *, /) and establishes their precedence. Further, the DSL defines a class of constraints, with subclasses for disequation constraints, conjunctions, and disjunctions, and defines their string representations. For convenience, the comparison operators ($<$, $>$, $=$) are also overloaded. Dedicated functions define `constants` and `infinitesimals`, and turn them into expressions. Similarly, `players` and `actions` turn them into a list of members of classes `Player` and `Action`, respectively. The class of trees is defined, with subclasses `Leaf` (containing a dictionary of utilities) and `Branch` (containing the player and a dictionary mapping actions to trees); and a dedicated `json` method, which produces a tree representation suitable to serve as CheckMate input. Finally, a function `finish` can be called to output the data into an appropriate '.json' file that can be piped directly to CheckMate.

## 5 Examples of Generated Game Models

Using the modeling principles specified in Section 3, the DSL of Section 4, and our modeling template available in [22], we generated game-theoretic models in the form of an EFG for the protocols listed in Table 1.

The Closing and Routing protocols listed in Table 1 are described in Examples 2–3 and explained in Section 2.1. A short description of the fAsset protocol can be found in Section 7. The other game-theoretic models of Table 1 are not modeling blockchain protocols, but rather simpler games, illustrating the variety to which our modeling principles apply. Here we give a short explanation of those models and refer to [22] for the full implementation.

The Auction model represents a simple auction with 4 players involved - an auctioneer and three bidders. The auctioneer sets an initial price for the item

[5] The model of the fAsset protocol uses conditional actions defined in Section 7. The model implementation is currently not public, as the security analysis is still in process, but it will be made available once the analysis is terminated.

| Game | Nodes | Players | Honest histories | LOC |
|---|---|---|---|---|
| Closing | 2131 | 2 | $(H)$, $(C_h, S)$ | 265 |
| Routing | 21688 | 3 | $(S_H, L, L, U, U)$ | 487 |
|  | 144342306 | 4 | $(S_H, L, L, L, U, U, U)$ | 487 |
| Routing Unlocking | 18707 | 5 | $(U, U, U, U)$ | 271 |
| fAsset[5] | 1805409 | 6 | (CRTc+pp) | 4300 |
| Auction | 81 | 4 | $(L, E, I, I)$ | 131 |
| EBOS | 31 | 4 | (Mine, Mine, Mine, Mine) | 123 |
| Tic-Tac-Toe | 549946 | 2 | $(CM, RU, LU, RD, RM, LM, CU, CD, LD)$ | 126 |
| Tic-Tac-Toe (concise) | 58748 | 2 | $(CM, LU, RU, LD, LM, RM, CU, CD, RD)$ | 182 |
| Pirate | 161 | 4 | $(y, y), (y, n, y)$ | 193 |

**Table 1.** Examples of game-theoretic models. The full implementation of the models can be found at [22]. Columns 2–3 list respectively the number of EFG nodes and players of the games listed in Column 1. Column 4 specifies the intended honest history of the EFG, whereas Column 5 gives the lines of code (LOC) in our model templates.

to be sold. This can be less than what the item is worth in the auctioneer's opinion, exactly what they think it is worth, or more than what they think it is worth. As usual, the auctioneer also has the choice not to put it up for auction. Then the three bidders have the option to – one after the other – bid a price that is higher than the previous one, which can be lower than what they think the item is worth; exactly what the item is worth; or higher than what it is worth. In this model, each player can bid at most once. Their utility depends on whether they received the item or not. If the item was sold, the bidder who buys it receives what they think it is worth minus what they paid; the other bidders receive a very small (infinitesimal) negative utility for not getting an item they wanted; the auctioneer receives what they sold it for minus what they think it was worth. If the item was not bought, everyone receives a small (infinitesimal) negative utility, because it is assumed they wanted to sell/buy the item. Finally, if the auctioneer never put it up for auction, everyone receives utility 0.

The EBOS game is an extended version of the game-theoretic problem known as the battle of the sexes. In the Extended Battle Of the Sexes (EBOS), there are 4 players, which model two couples. Each player can choose between two activities: one they like but their partner does not, and one they do not like but their partner does. Their utility depends on whether they do the activity they prefer and with whom they do it.

The Tic-Tac-Toe game is enhanced with an end utility for each player: the player that wins receives a reward $w > 0$, reduced by the penalty $s$ for every move required. The constraint $w > 9s$ is an initial constraint of the model. The player who loses the game is compensated by the utility $k \cdot s - w$, where $k$ is the number of moves played. In the case of a draw, both players get utility 0. The concise version identifies equivalent actions and thus reduces the size of the model by breaking the symmetry. For example, the first player making a move can choose to pick a corner, but it does not matter whether it is the top left, top right, bottom left, or bottom right. These 4 actions are equivalent. Both versions of the model are faithful representations of a Tic-Tac-Toe game and hence also bear the same game-theoretic properties.

The Pirate game is an adapted version of the "puzzle for pirates" introduced in [23]. It follows a voting scenario: there are 4 players ($A$, $B$, $C$, and $D$) and each player proposes a distribution of a joint utility $g$. First, the players vote whether to accept $A$'s proposed distribution, in alphabetical order. If the majority of players are in favor of the proposal (indicated by taking action $y$ when it is their turn), the game ends with the proposed utilities. Otherwise, i.e., if the majority votes no (action $n$), player $A$ is eliminated from the game, which results in the utility $-d$ for player $A$ when the game ends, with $d > 0$. The process repeats with the joint utility proposed by player $B$, and then $C$. In case of a tie, the decision of the proposing player is the casting vote.

## 6 Evaluation of Modeling Principles

*Comparison to Manual Games.* The Closing and the Routing protocol, as explained in Section 2.1, have previously been modeled manually [25,20]. We now compare the game models generated in Section 5 to their manually modeled counterparts. We note that in the related approach of [25], the Closing protocol is modeled as a Normal form game (NFG) with two players rather than an EFG. That means that in [25], both players only get to choose an action once and simultaneously, which drastically simplifies the protocol. The manual model of the Closing protocol in [20], which is modeled as an EFG with 221 nodes, provides a more thorough view when compared to [25]. However, compared to our generated model with 2131 nodes, the approach of [20] still misses several possible sequences of choices, such as player $B$ proposing their own collaborative closing (actions $C_c$ and $C_h$ after player $A$ chose $C_c$ or $C_h$ in Figure 3). While these additional choices arose naturally when following the modeling template, since player $B$ as well has the option to close the channel at any point in time, they were overlooked in [20]. Nonetheless, the manual model of [20] and the one generated using our modeling principles yield the same security result when analyzed by CheckMate.

Similar observations and improvements are also true for the Routing protocol (see Section 2.1). This protocol is modeled as an EFG with 3 players in [25], where each player only ever has 2 options: to lock the money or to not lock the money in the *locking phase* of the protocol (steps 1–5 in Figure 1); and to unlock the money or to not unlock the money in the *unlocking phase* (steps 6–9 in Figure 1), thereby ignoring all possible deviations such as sharing secrets or locking the wrong amount of money. The work of [20] aimed to consider these deviations in an EFG model of the Routing protocol with 5 players; however, due to the sheer size of the resulting game tree, the method of [20] only presents a partial manual model of the Routing protocol.

*Assumptions.* The fact that we need to make assumptions is not specific to the introduced modeling principles, but a natural consequence of abstraction. We work with two kinds of assumptions:

1. Assumptions on the *occurring variables*, listed as `INITIAL CONSTRAINTS`. These assumptions are typically driven either by (i) the protocol itself: excluding impossible values for variables, such as routing a negative amount in the routing protocol, Section 2.1; or by (ii) the modeling as an EFG: restrictions on values may be necessary to ensure all the given choices are possible for all allowed values of the variables.
2. Assumptions that are just *documented in words, but are not part of the generated EFG*. These assumptions usually stem from two considerations: (i) to decrease the size of the game tree, without violating the faithfulness of the generated model; or (are necessary) (ii) to accommodate the community's choice to use EFGs for game-theoretic security analyses, as discussed in Section 7.

*Example 9 (Game Modeling Assumption).* As an example of an assumption of the kind of (2i) discussed above, we study *secret sharing* in our model of the routing protocol (Figure 2). We allow each player $P$ to share the secret (actions $S_S$ in Figure 2) once with a subset of players $S$. Player $P$ can never, in the future, share the secret with further players. Sharing the secret at most once and with a set of players, rather than with one player at a time, significantly reduces the size of the game tree, while arguably not impacting its faithfulness: a player should, in principle, not share the secret. However, if another one does so while refusing to unlock (e.g., to perform a Wormhole attack [15], where an intermediary's participation fee $f$ is stolen), our approach allows the others to rectify the situation by sharing the secret with the deceived intermediary.

*Required Domain Knowledge.* Even when following our modeling principles, constructing game models still requires extensive domain knowledge. For example, the choice of parameters and the encoding of constraints need to be designed in a faithful way, and expertise is required for this. However, the manual effort is now semi-automated, reducing the errors during modeling and scaling. It further allows the user to focus on the technicalities of the protocol to be modeled rather than on defining their own modeling strategy.

## 7 Limitations of EFGs as Game-Theoretic Models

When considering EFGs as game-theoretic models of blockchain protocols, there are certain limitations, which also imply limitations on the modeling principles from Section 3. First of all, we consider deterministic models, rather than probabilistic ones - for security analysis, we ask whether the honest players could lose money, which depends on all possibilities of the other players' behavior and is fundamentally deterministic, thereby circumventing probabilistic effects. We assume that at every given point, full information about which previous actions were taken is known to the current player. Further, no simultaneous actions or choices of different players are permitted in the model. While in the protocol, it is often the case that any player could take action, an EFG model has to pose some assumptions on the order in which players act. Thus, the modeling principles require the user to define the player precedence (also in the modeling template), which affects the assumptions and the tree generation. Similarly, EFGs model finite games, and the modeling principles reflect that by checking whether a final state has been reached or the tree generation continues.

In an EFG with symbolic utilities, the actions available to the players cannot depend on the actual values of symbolic expressions and parameters. Further, there could be outside – possibly stochastic – effects from the environment that influence players' choices, but have no agenda on their own. Examples of such an effect would be time or price changes of the currencies, assuming players in the protocol are not moving the market. Modeling such effects as another player would compromise the definition of game-theoretic security as defined in [20]. In the modeling principles, this limitation is reflected in exhausting parameter options, where more actions are necessary to distinguish available choices. To enable more leeway in modeling, we propose the following extension to EFGs.

## 7.1 Extending EFGs to Conditional Actions

In this section, we introduce a generalization to EFGs to allow for actions that can only be taken if some (uncontrollable by the players, possibly probabilistic) condition is met. We call such actions *conditional actions*.

**Definition 2 (EFGs with Conditional Actions).** *An* extensive form game with conditional actions *is an EFG with the following adaptations:*

- *every non-terminal history $h$ is assigned a set of constraints $CA(h)$ called* conditions*;*
- *every action $a$ from an internal node belonging to history $h$ is assigned one condition $c(a)$ from $CA(h)$;*
- *let $h = (a_1, a_2, \dots, a_k)$ be a non-terminal history. Then the following holds:*
  1. *the conditions in $CA(h)$ are mutually exclusive:*

$$\big(\bigvee_{c \in CA(h)} c \wedge (\forall c, c' \in CA(h). c \neq c'\big) \implies \neg(c \wedge c');$$

  2. *the conditions are collectively exhaustive (i.e. span the whole subspace):*

$$(c(a_1) \wedge c(a_2) \wedge \dots \wedge c(a_k)) \iff \big(\bigvee_{c \in CA(h)} c \wedge c(a_1) \wedge c(a_2) \wedge \dots \wedge c(a_k)\big);$$

  3. *conditions along $h$ are non-contradictory: $(c(a_1) \wedge c(a_2) \wedge \dots \wedge c(a_k)) \neq \bot$.*
- *honest behavior is captured by an* honest subtree*: At the root node, one action is chosen for every condition. Then, recursively, at every node that belongs to a chosen action, for every condition, one action is chosen.*

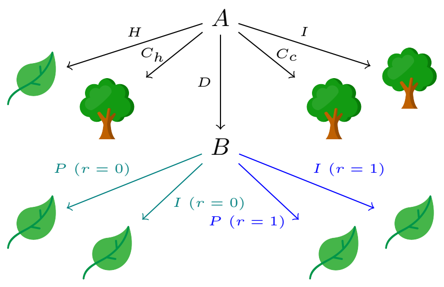


**Fig. 4.** Sketch of Lightning's Closing phase with conditional actions. Tree icons by Freepik - Flaticon.

*Example 10 (Closing Game as an EFG with Conditional Actions).* Let us revisit the Closing game from Example 3, but this time suppose that proving dishonest behavior is additionally rewarded by the system. If a player behaves dishonestly (chooses action $D$), and the other player proves it on chain (action $P$), the system will not require the transaction fee $f$ for this proof for every other such player. The players have no control over whether the system will pardon the fee or not. We model this behavior by introducing a new symbolic value $r$, requiring in the initial constraints that $r = 0 \vee r = 1$ and in the utility multiplying the transaction fee by this factor, so $r \cdot f$. After a dishonest action $D$, we encounter conditional actions, as depicted in Figure 4. There are two conditions ($r = 0$ depicted with teal actions and $r = 1$ depicted with blue actions), each having two actions possible (proving dishonest behavior $P$, or ignoring to prove $I$). These conditions are mutually exclusive and exhaustive. They are also non-contradictory when paired with the additional initial constraint. Note that in the nodes that do not require a conditional split of actions, we can assign the trivial condition $\top$.

The requirements on the conditions (mutually exclusive, exhaustive, non-contradictory) are, in practice, not difficult to meet and are relevant for subsequent security analysis: the definitions of the security properties need to be adapted for these extended models.

Our main motivation for defining EFGs with conditional actions is modeling the fAsset protocol on the Flare network. This decentralized finance platform enables communication of different blockchains through wrapped tokens: the assets (like Bitcoin, XRP, Eth, Dodgecoin, etc.) can be represented as fAssets on the Flare ecosystem and can be redeemed to reclaim the original assets. Such representation is sensitive to the price changes on the market, so the fAssets are collateralized during minting to ensure the redemption can always be performed for the original assets or for collateral. If the collateral drops below a certain threshold due to market fluctuations or misbehavior of the agents, a so-called liquidation phase is entered, where users are encouraged to redeem the fAssets in exchange for the collateral. The conditional actions in an EFG model play a natural role here, representing the changes in the market that can either trigger the liquidation of fAssets or not.

## 8 Conclusion

The modeling principles introduced in this paper rely on the definition of an EFG as a suitable model for the game-theoretic security analysis and can be automatically processed by game-theoretic security checks in CheckMate [2,21]. While previous work [20,25] introduced manually made models, to the best of our knowledge, our work provides the first semi-automated approach in generating games modeling protocols. Our approach enables exhausting all parameter options systematically in a machine-supported manner, thereby refining existing models, while also making fewer assumptions along the modeling process.

The modeling support presented in this paper makes automated game-theoretic security analyses much more accessible, as it eases the modeling process and automates the mechanical part of error-prone game tree generation. While the semi-automatic approach described here is already quite beneficial, (automatically) synthesizing game models from the protocol's specification, respectively source code in the case of smart contracts, is a challenge we aim to address in the future, in addition to the extension to conditional actions.

**Acknowledgments.** The work on extensive form games with conditional actions in Section 7 involved joint work with Ivana Bocevska during her master thesis project at TU Wien. The model of the fAsset protocol was developed in collaboration with Filip Koprivec. The research presented in this paper 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.

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Blanchet, B.: Automatic Verification of Security Protocols in the Symbolic Model: The Verifier ProVerif. In: Aldini, A., Lopez, J., Martinelli, F. (eds.) Foundations of Security Analysis and Design VII: FOSAD 2012/2013 Tutorial Lectures. pp. 54–87. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-10082-1_3, https://doi.org/10.1007/978-3-319-10082-1_3
2. Brugger, L.S., Kovács, L., Petkovic Komel, A., Rain, S., Rawson, M.: CheckMate: Automated Game-Theoretic Security Reasoning. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. p. 1407–1421. CCS '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3576915.3623183, https://doi.org/10.1145/3576915.3623183

3. Capucci, M., Ghani, N., Ledent, J., Nordvall Forsberg, F.: Translating extensive form games to open games with agency. Electronic Proceedings in Theoretical Computer Science **372**, 221–234 (Nov 2022). https://doi.org/10.4204/eptcs.372.16, http://dx.doi.org/10.4204/EPTCS.372.16
4. Chandrakana Nandi, Mooly Sagiv, D.J.: Certora Technology White Paper: Unveiling the Power and Limitations of Certora's Smart Contract Verification Technology, https://www.certora.com/blog/white-paper
5. Cheung, Y.K., Leonardos, S., Piliouras, G., Sridhar, S.: From griefing to stability in blockchain mining economies (2021), https://arxiv.org/abs/2106.12332
6. Dxo, Soos, M., Paraskevopoulou, Z., Lundfall, M., Brockman, M.: Hevm, a fast symbolic execution framework for evm bytecode. In: Gurfinkel, A., Ganesh, V. (eds.) Computer Aided Verification. pp. 453–465. Springer Nature Switzerland, Cham (2024)
7. FAssets, https://dev.flare.network/fassets/overview/
8. Flare: The Blockchain for Data, https://flare.network/
9. Fooladgar, M., Manshaei, M.H., Jadliwala, M., Rahman, M.A.: On incentive compatible role-based reward distribution in algorand. In: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 452–463 (2020). https://doi.org/10.1109/DSN48063.2020.00059
10. Ghani, N., Hedges, J., Winschel, V., Zahn, P.: Compositional game theory. In: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science. p. 472–481. LICS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3209108.3209165, https://doi.org/10.1145/3209108.3209165
11. Ghani, N., Kupke, C., Lambert, A., Nordvall Forsberg, F.: Compositional game theory with mixed strategies: Probabilistic open games using a distributive law. Electronic Proceedings in Theoretical Computer Science **323**, 95–105 (Sep 2020). https://doi.org/10.4204/eptcs.323.7, http://dx.doi.org/10.4204/EPTCS.323.7
12. Holler, S., Biewer, S., Schneidewind, C.: Horstify: Sound security analysis of smart contracts. In: 36th IEEE Computer Security Foundations Symposium, CSF 2023, Dubrovnik, Croatia, July 10-14, 2023. pp. 245–260. IEEE (2023). https://doi.org/10.1109/CSF57540.2023.00023, https://doi.org/10.1109/CSF57540.2023.00023
13. Kobeissi, N., Nicolas, G., Tiwari, M.: Verifpal: Cryptographic protocol analysis for the real world. In: Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop. p. 159. CCSW'20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3411495.3421365, https://doi.org/10.1145/3411495.3421365
14. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: PRISM-games 3.0: Stochastic Game Verification with Concurrency, Equilibria and Time. In: CAV. pp. 475–487. Springer International Publishing, Cham (2020)
15. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In: Network and Distributed System Security Symposium. The Internet Society, San Diego, CA, USA (2019)
16. Manshaei, M.H., Jadliwala, M., Maiti, A., Fooladgar, M.: A game-theoretic analysis of shard-based permissionless blockchains. IEEE Access **6**, 78100–78112 (2018). https://doi.org/10.1109/ACCESS.2018.2884764

17. Mazumdar, S., Banerjee, P., Sinha, A., Ruj, S., Roy, B.K.: Strategic analysis of griefing attack in lightning network. IEEE Transactions on Network and Service Management **20**(2), 1790–1803 (Jun 2023). `https://doi.org/10.1109/tnsm.2022.3230768`, `http://dx.doi.org/10.1109/TNSM.2022.3230768`
18. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In: Computer Aided Verification. pp. 696–701. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). `https://doi.org/10.1007/978-3-642-39799-8_48`, `https://doi.org/10.1007/978-3-642-39799-8_48`
19. Poon, J., Dryja, T.: The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. white paper (2016), `https://lightning.network/lightning-network-paper.pdf`
20. Rain, S., Avarikioti, G., Kovács, L., Maffei, M.: Towards a game-theoretic security analysis of off-chain protocols. In: 2023 IEEE 36th Computer Security Foundations Symposium (CSF). pp. 107–122. IEEE Computer Society, Los Alamitos, CA, USA (2023). `https://doi.org/10.1109/CSF57540.2023.00003`, `https://doi.org/10.1109/CSF57540.2023.00003`
21. Rain, S., Brugger, L.S., Komel, A.P., Kovács, L., Rawson, M.: Scaling checkmate for game-theoretic security. In: Bjørner, N., Heule, M., Voronkov, A. (eds.) Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning. EPiC Series in Computing, vol. 100, pp. 222–231. EasyChair, Stockport, UK (2024). `https://doi.org/10.29007/llnq`, `/publications/paper/6ZDH`
22. Rain, S., Komel, A.P., Rawson, M., Kovács, L.: Game Modeling of Blockchain Protocols – Artifact. `https://zenodo.org/records/16925288` (2025)
23. Stewart, I.: A puzzle for pirates. Scientific American **280**(5), 98–99 (1999)
24. Wang, T., Bai, X., Wang, H., Liew, S.C., Zhang, S.: Game-theoretical analysis of mining strategy for bitcoin-ng blockchain protocol. IEEE Systems Journal **15**(2), 2708–2719 (2021). `https://doi.org/10.1109/JSYST.2020.3004468`
25. Zappalà, P., Belotti, M., Potop-Butucaru, M.G., Secci, S.: Game theoretical framework for analyzing Blockchains Robustness. IACR Cryptol. ePrint Arch. **2020** (2020), `https://api.semanticscholar.org/CorpusID:219616790`