Model Checking Nash Equilibria in Probabilistic Games

ABSTRACT

Verifying whether rational participants in a probabilistic distributed system would deviate from the specified behavior is an important but challenging task. We term this problem as verifying nash-equilibria in probabilistic cooperative distributed multi-player games. Existing algorithms only work for non-probabilistic systems. We develop a model checking algorithm for automatically verify nash-equilibria for probabilistic systems. The algorithm is implemented based on a model checker - PAT (Process Analysis Toolkit). Using the algorithm and our tool, a case study has been performed on analyzing the first probabilistic secret sharing scheme.

CCS Concepts

•**Software and its engineering** → **Model checking;** •**Mathematics of computing** → *Probability and statistics;* •**Theory of computation** → *Algorithmic game theory and mechanism design;* •**Security and privacy** → Security protocols;

Keywords

Model Checking; Nash Equilibria; Probabilistic Game; Contract Signing; Secret Sharing

1. INTRODUCTION

Many systems in the real-world involve collaboration of many distributed parties, e.g., Internet routing [26], peer-to-peer file sharing [3], cooperative backup [18], etc. In many of such systems, participants may have conflict interests. Thus, it is natural to consider each party as rational, meaning that each party only collaborates if his benefit from collaborating is bigger than not collaborating. That is, the system is actually a game. An important property for such systems/games is to ensure that every rational player would not deviate from the expected behavior specified in the systems. That is to say, to pursue the biggest benefit, the rational players would all follow the system, i.e., the system is the nash-equilibrium in the concept of game theory – no one derivate from the system because this would harm their benefits.

ACM ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

Traditionally, when designing a system, participants are assumed to be either "good guys" – following the system behavior, or "bad guys", who do everything in their power to break the system [1]. Rational/selfish participants are seldom considered; and few systems[1] work correctly under the rational settings (as stated in [22]).

When the desired properties, such as security or reliability, of such systems is critical, whether every collaborator follows the system becomes of great importance. In this case, precisely proving that the system is indeed the desired equilibrium is necessary. In the literature, such proofs are often manual, e.g., proofs in the contracting signing protocol [24] and the secret sharing protocol [13]. As the proof tasks are often heavy, manual proof is error-prone and inefficient, especially the designer of a system may not be an expert on formal proofs. Thus, automated proof is desirable. Model checking, which is a precise way to verify systems and has many automated tool support, is a promising method.

Automated model checking tools for games exist, e.g., MOCHA [2], MCMAS [19], GAVS+ [7] and PRISM-games [6]. The existing tools verify various types of properties in various types of games. However, none of them is able to verify the nash-equilibrium type of property – whether following the system is the optimal strategy for each rational players, in concurrent games. Recently, an algorithm is developed to verify such properties in a sub-class games which can be described using finite state mechanisms [22]. However, this work only considered games without probability. In real-world, many systems have probabilistic behavior: Some of the systems are inherently probabilistic, for example, economic and biology; while some other systems fail to achieve certain goal without involving probability, for example, it is proved that non-probabilistic secret sharing scheme is not practical in the rational settings [13]. Hence, we extend their algorithm to verify nash-equilibrium for probabilistic systems.

Contributions..

We develop a verification algorithm to verify whether a probabilistic system is a nash-equilibrium. We implement the algorithm based on PAT (Process Analysis Toolkit), which is a model checking platform providing efficient general probabilistic model checking algorithms [23]. We perform a case study on analyzing a probabilistic secret sharing scheme [13]. The authors of the scheme first prove that no practical mechanism for secret sharing works without using probability in the rational setting, then provide the first secret sharing with the rational participants in mind. This case study first illustrates how our algorithm and tool help ease the proofs of the non-existence of non-probabilistic scheme. Second, it shows how the tool automatically verifies the correctness of the probabilistic scheme. In addition, during verification, weaknesses of the proba-

[1]Notable exceptions include [1, 17]

bilistic scheme have been found.

2. RELATED WORKS

Nash-equilibria is a popular topic in game theory. Algorithms exist for finding nash-equilibria in games [11, 9, 8]. However, these works usually use simulation and mathematical analysis techniques [4]. Model checking has the advantages of both simulation, being automatic, and mathematical analysis, coving all possible behaviors. Hence, model checking has been applied to verify game theoretic properties, for example [14, 10, 20]. Among these works, we highlight the following which focused on probabilistic behaviors and automatic model checking tools for games.

The model checker MOCHA [2] has been widely used for verifying games, e.g., [15, 21, 28]. However, nash-equilibria checking is not explicitly supported, and probabilistic behaviors is not supported. Recently, a tool PRALINE is developed specifically for computing nash-equilibria [5], and a tool EAGLE is developed for verifying nash-equilibria for concurrent games [27]. However, these tools do not support probability.

Moder checker PRISM [16] supports probabilistic behaviors, but does not explicitly support verifying game properties. Using the Rubinstein's protocol as a case study, Ballarini et al. show that probabilistic model checking (using the model checker PRISM) can be used for game analysis [4]. This work inspirits automated probabilistic model checking for games, but it is specific to the negotiation game (a bargaining between a buyer and a seller) framework and does not address nash-equilibria analysis. Later, PRISM-Games extends the model checker PRISM to be able to verify turn-based games [6]. PRISM-game is able to verify nash-equilibriums, but only for turn-based games.

The most relevant work is the one by Mari et al. [22]. This work proposes a symbolic model checking algorithm for verifying nash-equilibrium in distributed cooperative systems. It it the first algorithm to automatically verify that it is in the best interest of each rational agent to follow exactly a given system [22]. A player's all possible behavior is modeled as a finite state mechanism. The given system describes the which action should be taken in which situation (by honest/altruistic players), which is modeled as constrains on state transitions of a game. Infinite sequences of actions is tackled by using *discount factor* [12] to decrease relevance of rewards that are far in the future, which allows them to only look at finite sequences of actions. The authors propose a notion of equilibrium which takes both *Byzantine* players – players that randomly take possible actions and ignore utilities/rewards, and a *tolerance* of rewards – the reward difference between following the system and deviating from the system, as parameters; and prove that using their algorithm the proposed equilibrium can be automatically verified by just looking at finite sequences of actions. This algorithm does not support verification of probabilistic systems. We adopt a similar specification framework of players and equilibrium, and develop nash-equilibria verification algorithm, which focuses on probabilistic systems.

3. SPECIFICATION

As mentioned in the introduction, we consider distributed systems with multiple players who need to cooperate in order to achieve a goal. Games for such systems are inherently distributed with multi-players, rather than turn based. In addition, a system can be non-terminating, i.e., there are infinite executing steps for some players. For instance, a system (e.g., the case study protocol) can be executed unbounded number of times.

Games are often represented in a normal form – a matrix of players, strategies and payoffs, or in the extensive form – a finite game tree specifying the sequences of actions. Such specification is no longer suitable when the game has infinite execution steps. To be able to model infinite games, we adopt the fine state mechanism representation defined in [22] – each player is specified as a finite state mechanism, potentially with loops. This allows us to specify an infinite execution steps in a finite manner.

We first briefly introduce the adopted specifications in Section 3.1, and then explain in details the probabilistic extension in Section 3.2. Finally, we use an example to illustrate how a probabilistic system is specified.

3.1 Preliminaries

We use the formalism of finite state mechanisms [22] to model all possible local states and local actions of players. All players' local states and actions together form all possible global states and actions in a game. The probability to perform the actions will be added in Section 3.2.

Formal Definition of Finite State Machine.

Each player i has a set of local states, denoted by S_i, a set of initial states, denoted by I_i, and a set of actions, denoted A_i. The global states in a game with n players is the product of the local states of each player, i.e., $S = S_1 \times S_2 \times \cdots \times S_n$; the set of initial states is $I = I_1 \times I_2 \times \cdots \times I_n$; and the set of global actions is $A = A_1 \times A_2 \times \cdots \times A_n$.

Function $B_i : S \times A_i \times S_i \to \mathbb{B}$ is used to specify the feasibility of player i performing a action $a \in A_i$, which leads i's local state to $s' \in S_i$, at a global state $s \in S$ – if it is feasible, $B_i(s, a, s')$ is true, otherwise false. The function B_i is required to have two properties: *serial* and *deterministic*:

- Serial property says that there is at least one action defined for each player at each global state, formally, $\forall s \in S, \exists a \in A_i, \exists s' \in S_i$ s.t. $B_i(s, a, s')$ is true;
- the deterministic property says that a player's action a should lead to a unique resulting local state, formally $(B_i(s, a, s_1) = true \wedge B_i(s, a, s_2) = true) \implies s_1 = s_2$.

Due to the deterministic property, we simply write $B_i(s, a)$ to represent that action a is valid in a global state s, since the local state s' is uniquely defined. Similarly, the global possible actions of all players are $B = B_1 \times B_2 \times \cdots \times B_n$.

In non-probabilistic settings, $\langle S_i, I_i, A_i, B_i \rangle$ defines a Byzantine player, since a Byzantine player can perform all actions that are feasible at a state, ignoring utility/rewards. Recall that a system specifies which action should be performed at which state by which player, i.e., a system is a specification of altruistic players. Similar to B_i, function $T_i : S \times A_i \to \mathbb{B}$ is defined to specify a player i's behavior of genuinely following the system – $T_i(s, a) = true$ ($s \in S$, $a \in A_i$) if i shall perform a at state s according to the system. Since following the system (altruistic player) is always part of the possible behavior, we have $T_i(s, a) \to B_i(s, a)$. In addition, we assume the system does not have unspecified situation or deadlocks. Formally, we require that T_i is non-blocking (i.e. $\forall s \in S, \exists a \in A_i$, s.t. $T_i(s, a)$). A system specifies the altruistic players, formally specified as $T = \langle T_1, \cdots, T_n \rangle$.

Since that function T_i models the altruistic player's behavior, and B_i models a Byzantine player's behavior, given a set of Byzantine players (denoted as Z) among the total n players, the behavior of Byzantine and altruistic players is summarised as:

$$BT(Z, s, a_i, s') = \begin{cases} B_i(s, a_i, s') & i \in Z \\ B_i(s, a_i, s') \wedge T_i(s, a_i) & i \notin Z \end{cases}$$

In probabilistic settings, in addition to the feasible actions, a probability is given to each action.

Payoff Function.

Differing from Byzantine players and altruistic players, a rational player's behavior is driven by rewards, a.k.a. payoffs. Player i's payoff is defined as a function $h_i : S \times A \to \mathbb{R}$, i.e., for each global state transition and a global action, there is a payoff value for player i, denoted as numbers in $\mathbb{R}$. Hence, $h = \langle h_1, \cdots, h_n \rangle$ denotes the payoff of all the n players at any state for performing any action. The payoff function can be directly adopted to the probabilistic settings since it is not influenced by adding probability.

Discount factor.

Let $\beta = (\beta_1, \cdots, \beta_n)$ be the n-tuple of discount factors for the n players, where $\beta_i \in (0, 1)$ is the discount factor for player i. The discount factor is used to decrease the relevance of rewards decreases in the future, so that to ensure efficient computation of a finite strategy in the settings of infinite games. We adopt the discount factor as well in order to handle infinite games in probabilistic settings.

Path.

With the above definitions, a non-probabilistic game can be specified as $(S, I, A, B, T, h, \beta)$. Given such a game with a set of Byzantine Z, a path is defined as a sequence of possible (s, a) pairs where $s \in S$ and $a \in A$. We denote a path as $X = \langle X_0, \cdots, X_n, \cdots \rangle$, where each step X_t ($t \geq 0$) is a (s, a) pair. We use $X_t^{(s)}$ to denote the first element of a step in a path – the state s and $X_t^{(a)}$ to denote the second element – the action a. A path is valid if $\forall t$, $BT(Z, X_t^{(s)}, X_t^{(a)}, X_{t+1}^{(s)}, X_{t+1}^{(a)})$ is true, meaning that the the transition from one step to the next step is valid.

The utility of a player i in a path is evaluated by adding up the discounted payoff values along the path, formally, $V_i(X) = \Sigma_{t=0}^{|X|-1} \beta_i^t h_i(X_t^{(s)}, X_t^{(a)})$, where $|X|$ is the path length, which can be infinite. When the path length is infinite, $V_i(X)$ converges to an upper bound, because of β_i.

Strategy.

To reason on a game, it is often needed to distinguish one player's actions from other players. Hence, the notion of strategy is defined as follows: A strategy of a player is a sequence of local actions of the player. A strategy at a state s defines what a player would behave in the future. Depending on other players' actions, there will be multiple paths with multiple utility values. The minimum utility value is called the *guaranteed outcome* of the strategy for the player at the state s. In the algorithm in [22], two important values are calculated:

- the *value* of a state s at horizon k for player i with Byzantine players Z, denoted as $v_i^k(Z, s)$, is defined as the guaranteed outcome of the best strategy of length k starting ate state s;
- the *worst case value* of a state s ate horizon k for player i with Byzantine players Z, denoted as $u_i^k(Z, s)$, is defined as the guaranteed outcome of the worst strategy of length k starting at state s.

We show, in the next section, how these two values are calculated in the probabilistic settings.

3.2 Probability Extension

We extend the above specification to be able to specify probabilistic systems. A probabilistic system specifies which actions of a player is probabilistic and specifies a probability for each actions. To be more general, we allow hybrid probabilistic systems, meaning that some actions of a player in the systems are probabilistic, some of the actions can be deterministic and some others can be non-deterministic.

First of all, we update the model of players. Then we extend the state representation to be probabilistic. More importantly, we present the calculation of the value $v_i^k(Z, s)$ and worst value $u_i^k(Z, s)$ of a state in the probabilistic settings.

Byzantine Player.

In a non-probabilistic setting, a Byzantine player is modeled as all possible local states and their corresponding feasible local actions. In a probabilistic setting, a Byzantine player has exactly the same local states and corresponding local actions, since probabilistic systems do not introduce additional local actions. That is to say, similar as in the non-probabilistic setting, a Byzantine player can deviate from a system specification by performing actions that are not defined in the system. In addition to perform extra actions, for the probabilistic actions in a system, a Byzantine player is able to deviate from the specification by changing the probability distributions of the actions. For example, if a system specifies a player's action as follows: the player toss a coin - with probability $1/2$ to be head and $1/2$ to be tail, if it is head, the player sends 1, otherwise, sends 0. A Byzantine player can deviate from the specification by changing the coin into a special one – with probability $1/3$ to be head and probability $2/3$ to be tail. Furthermore, a Byzantine player is able to assign probability to non-probabilistic actions. For example, a system specifies a deterministic action of sending 1, a non-probabilistic Byzantine player can deviate by sending 0, a probabilistic Byzantine player can toss a coin – with probability α to be head and probability $1 - \alpha$ to be tail, and sends 1 if it is head and sends 0 if it is tail. To summarize, a Byzantine player is able to assign an arbitrary probability distribution (which adds up to 1) to at a local state.

Since local actions at a local states are feasible actions at the corresponding global states, this can be formally modelled by defining a function $P_i : S \times A_i \to \mathbb{R}$ where $\mathbb{R}$ ranges over $[0, 1]$, with the requirement that at a global state s, $\Sigma_{a \in A_i \wedge B_i(s,a)} P_i(s, a) = 1$.

Altruistic Player.

An altruistic player follows system specification, and thus has predefined probability for each probabilistic action and may have deterministic and non-deterministic actions depending on the system specification. Locally, this can be modelled by adding the predefined probability to probabilistic actions and keeping other actions the same as in the non-probabilistic settings.

Globally, in order to specify the hybrid actions, we need to distinguish which actions are probabilistic and which are not. Since the system specifies the distinction. At a global state, we are able to tell the probabilistic actions from the non-probabilistic actions. We use $PA_i(s)$ to denote the probabilistic actions of player i at global state s. We require that $PA_i(s) = \{a | a \in A_i \wedge T_i(s, a)\}$ and $\Sigma_{a \in A_i(s)} P_i(s, a) = 1$. This definition only applies to the altruistic behaviour of a player. The former requirement means that the probabilistic actions should be one of the possible actions and is valid at state s. The later requirement says that the probability of all probabilities actions adds up to 1. Define $PA = (PA_1, PA_2, \cdots, PA_n)$.

df: Will assume that all the actions are either probabilistic or deterministic for honest players

Rational Player.

A rational player is a special case of a Byzantine player who has the full power to change the probability distribution of actions at a state, driven by utility. We show that a rational player would always choose an action which gives the maximum utility with probability 1. Taking the simplest case with two actions,

- If both of the actions lead to paths that give exactly the same utility, the probability distribution of the two actions does not influence the utility. Hence, we can assume the player choose one action with probability 1, the other with probability 0, without losing of generality.
- If the two actions lead to paths that give different utility, the player would choose the one gives bigger utility with probability 1, the other with probability 0.

The same reasoning holds for the case that the player has more than two feasible actions. Therefore, we can safely model a rational player with non-probabilistic actions. Note that this does not mean that probabilistic games are the same as non-probabilistic games, as we will show later.

States.

We extend the global state representation in [22] with probability. Since a state may be reached via different paths. Each path that leads to the state may have different probability. In this case, we consider the global state with different probability two different states. In order to model it, we learn from the extensive form and use the history of states that lead to the current state to represent the current global state. df: I believe that all paths to a global state should be considered to calculate the reachability probability In this way, given a state, the path to the state is unique, hence the probability reaching the state is deterministic. A global probabilistic state is an extensive global state with a probability that the state is reached, denoted as $(s, P(s))$ where s is a path from the initial state, and $P(s)$ is the probability of reaching s along the path. The probability of reaching a particular global state is the steady state probability of that particular state. Steady state probabilities of all global states should add up to 1. So, It is not necessary for a global probabilistic state to have their probability values add up to 1. These probability values should be normalized to make the summation equal to 1 before inputting to any algorithm.

Strategy.

Strategy is defined the same as in the previous section. The only difference is that all the strategies are assigned a probability.

Reachability.

Define a relation $nstate \subseteq S \times S$. Let s , $s' \in S$. s nstate s' iff $\exists a \in A$ s.t. $BT(Z, s, a, s')$. So, we can define the relation $reaches \subseteq S \times S$ as $reaches$= $nstate^+$ It should be noted that these relations are defined for notational convenience and would not add more information to the finite state mechanism.

Probabilistic Value Function.

$vp_i^{k+1}(Z, s)$ and $up_i^{k+1}(Z, s)$ are defined for probabilistic states where, $s \in S_i$

Hence, the probabilistic game is specified as $(S, I, A, B, T, h, P, PA\beta)$ where P and PA are newly added. df: probabilistic game definition vector has to be updated

We extend the following example in [22] with probability, to show how a probabilistic system is modeled.

EXAMPLE 1. *Suppose there are m jobs $\{J_1, \cdots, J_m\}$, and q tasks $\{T_0, \cdots, T_n\}$. Each job is consisted of a set of tasks. A system, consisting of n workers, specifies a sequence of tasks for each worker. A task sequence for player i is denoted as $\mathcal{T}_i = \langle \tau(i,0), \cdots, \tau(i,l_i) \rangle$, where $l_i + 1$ is the length of the sequence. When finishing one task, a work waits for the rewards. Once the rewards are received, the worker starts the next task. After finishing all tasks in the sequence (i.e., after finishing $\tau(i,l)$), a worker repeats the sequence of tasks from beginning (i.e., task $\tau(i,0)$). Once a job is completed, the rewards are granted to the works who finished the tasks that compose the job. Assume that tasks for different jobs do not overlap. In completing a job, if more than one workers finish the same task, all of them will be rewarded. A worker may deviate from the system by delaying execution of a task or not executing a task in his task sequence.*

We extend the above example system with probability. In addition to the task sequence, each worker is assigned a probability 0.4 to take the next task, and probability 0.6 to skip the next task. For example, a work will toss a coin before taking the next task: if it is head, he takes the next task, if it is tail, he skip the next task.

Similar to the formalization in [22], in the example, a worker's states are modeled as $X_i = Y_i \times Z_i$, where Y_i is the worker's task sequence and Z_i is the status of each task, $Z_i = \{0, 1, 2\}$, with 0 denoting that the worker is not working, 1 denoting that the work finishes his assigned work, and 2 denoting that the current (finished) work is used by a job. Note that in this example, the states for altruistic, rational or Byzantine players/workers are exactly the same. A player's initial state is $I_i = (\tau(i,0), 0)$. Since the only deviation is not taking the task, the set of actions for all players are the same, i.e., $A_i = \{a_0^i, a_1^i\}$, where a_0^i denotes the player i does not take the task, and a_1^i denotes that the player i will take the task. The action a_1 will lead a state $(\tau(i,j), 0)$ to $(\tau(i,j), 1)$, and the action a_0 leads to a self-loop. The transition from $(\tau(i,j), 1)$ to $(\tau(i,j), 2)$ happens if the task $\tau(i,j)$ is used for a job. Hence, $S = \langle (Y_1, Z_1), \cdots, (Y_n, Z_n) \rangle$, $I = \langle (\tau(1,0), 0), \cdots, (\tau(n,0), 0) \rangle$ and $A = \{\{a_0^1, a_1^1\}, \cdots, \{a_0^n, a_1^n\}\}$.

We use the function π_i to project a global state s or a global action a to the local state of player i. The function $B_i(s, a, s')$ is defined as follows:

$$B_i(s, a_i, s') = \begin{cases} \text{true} & \text{if } \pi_i(s) = (\tau(i,j), 0), \pi_i(s') = (\tau(i,j), 0), a_i = a_0^i \\ \text{true} & \text{if } \pi_i(s) = (\tau(i,j), 0), \pi_i(s') = (\tau(i,j), 1), a_i = a_1^i \\ \text{true} & \text{if } \pi_i(s) = (\tau(i,j), 1), \pi_i(s') = (\tau(i,j), 2), \\ & \tau(i,j) \text{ is completed and needed for a job} \\ \text{true} & \text{if } \pi_i(s) = (\tau(i,j), 1), \pi_i(s') = (\tau(i,j), 1), \\ & \tau(i,j) \text{ is not completed or not needed for a job} \\ \text{true} & \text{if } \pi_i(s) = (\tau(i,j), 2), \pi_i(s') = (\tau(i, (j+1)\%l_i), 0) \\ \text{false} & \text{otherwise} \end{cases}$$

The first case says that if a player i's next job is j, he can choose to not take the task ($a_i = a_0^i$), and this action leads to a state where i's local state does not change ($\pi_i(s') = \pi_i(s)$). The second case models that if the player takes the task ($a_i = a_1^i$), the action leads to a state where the local state of i changes into $\tau(i,j), 1)$. The third case represents that if the player's task is finished and is used for a job, then the status of his task is changed to 2, otherwise, the status of the task remains to be 1 (the fourth case). The fifth case states that if the task is rewarded, then the system automatically transit the state of the player to the beginning status of the next task.

Differing from the definition in [22], the function $T_i(s, a_i)$ in this system is defined exactly the same as the first two cases of $B_i(s, a_i)$, since both of the actions a_0^i and a_1^i are enabled in the probabilistic system.

$$T_i(s, a_i) = \begin{cases} \text{true} & \text{if } \pi_i(s) = (\tau(i,j), 0), a_i = a_0^i \end{cases}$$

The payoff function is defined the same as original as follows.

$$h_i(s,a_i) = \begin{cases} -1 & \pi_i(s) = (\tau(i,j),0) \wedge (a_i = a_1^i) \\ 4 & \pi_i(s) = (\tau(i,j),2) \\ 0 & \text{otherwise} \end{cases}$$

The first case represents that the player works for a task, meaning that the player is paying, instead of gaining, and thus the payoff is negative. The second case capture the situation that the player's task is used in a job and thus the player is paid with positive payoff value. If the player does not work then he gains nothing and pays nothing, and thus has payoff value 0.

The probabilistic function $P(Z,s,a_i)$ is defined as follows:

$$P(Z,s,a_i) = \begin{cases} \begin{cases} p & \text{if } i \in Z, a_i = a_0^i \\ 1-p & \text{if } i \in Z, a_i = a_1^i \\ & \text{where } p \text{ is any value between 0 and 1} \end{cases} \\ \begin{cases} 0.6 & \text{if } i \notin Z, a_i = a_0^i \\ 0.4 & \text{if } i \notin Z, a_i = a_1^i \end{cases} \end{cases}$$

If a player is a Byzantine player $i \in Z$, then the probability can be an arbitrary distribution (p can be any value between and include 0 and 1). If a player is an altruistic player $i \notin Z$, the probability distribution is pre-defined by the system (0.6 for not taking the task and 0.4 for taking the task).

Finally, all actions of a player are probabilistic, hence, $PA_i(s)$ is defined as follows:

$$PA_i(s) = \begin{cases} \{a_0^i, a_1^i\} & \text{if } T_i(s,a_i) = \text{true} \\ \emptyset & otherwise \end{cases}$$

4. VERIFICATION ALGORITHM

We are going to verify the Nash Equilibrium of synchronous parallel finite state mechanisms. In order to define the equilibrium we should define a utility function with respect to the execution of the protocol. The concepts of *path* and *strategy* has to be used in order to define utility.

The dynamic programming algorithm to evaluate the value and guaranteed outcome at horizon k for player i from a global state s is as follows. Let $s_{pr} \in S_{pr}$ df: updating the algorithm as per new definitons

$v_i^0(Z,s) = u_i^0(Z,s) = 0$ [22]

$$v_i^{k+1}(Z,s) = \begin{cases} i \in Z \\ max_{a_i \in A_i(s)}\{E_{a_{[n]-Z} \in A_{[n]-Z}(s)}(min_{a_{Z-\{i\}} \in A_{Z-\{i\}}} \\ \{h_i(s, < a_{z-\{i\}}, a_i, a_{[n]-Z} >) + \\ \beta_i {v_i}^k(Z,s') | BT(Z,s, < a_{-i}, a_i >, s')\})\} \\ i \notin Z \\ E_{a_i \in A_i(s)}(E_{a_{[n]-Z-\{i\}} \in A_{a_{[n]-Z-\{i\}}}}(min_{a_Z \in A_Z(s)} \\ \{h_i(s, < a_{[n]-Z-\{i\}}, a_Z, a_i >) + \\ \beta_i {v_i}^k(Z,s') | BT(Z,s, < a_{-i}, a_i >, s')\})) \end{cases} \quad (1)$$

df: Define $A_{at(-i)}$

$$u_i^{k+1}(Z,s) = \begin{cases} i \in Z \\ min_{a_i \in A_i(s)}\{E_{a_{[n]-Z} \in A_{[n]-Z}(s)}(min_{a_{Z-\{i\}} \in A_{Z-\{i\}}} \\ \{h_i(s, < a_{z-\{i\}}, a_i, a_{[n]-Z} >) + \\ \beta_i {u_i}^k(Z,s') | BT(Z,s, < a_{-i}, a_i >, s')\})\} \\ i \notin Z \\ E_{a_i \in A_i(s)}(E_{a_{[n]-Z-\{i\}} \in A_{a_{[n]-Z-\{i\}}}}(min_{a_Z \in A_Z(s)} \\ \{h_i(s, < a_{[n]-Z-\{i\}}, a_Z, a_i >) + \\ \beta_i {u_i}^k(Z,s') | BT(Z,s, < a_{-i}, a_i >, s')\})) \end{cases} \quad (2)$$

Probabilistic state for $s_i \in S_i$ can be defined as,

$PS(s_i) = \{t | t \in S \wedge t_i = s_i \wedge \exists s0 \in I, s0 \text{ reaches } t\}$

Now we can define the values of the game in terms of player i's states. $vp_i^{k+1}(Z,s_i) = E_{x \in PS(s_i)}(v_i^{k+1}(Z,x))$ and $up_i^{k+1}(Z,s_i) = E_{x \in PS(s_i)}(u_i^{k+1}(Z,x))$

df: The above equation is finalized. Calculation of probability of reaching states is handled follows. Consistency of the symbols have to be checked To calculate the expected values we have to be aware of the probability distributions of the probabilistic states. The steady state probability distribution of the global states. Expected value should be taken over normalized probabilities.

Explaining the algorithm, we consider the expected values of different sets of non deterministic strategy choices and calculate max/min values among these expected values.

Now, we are ready to define $\epsilon - f$ Nash Equilibrium.

Let $(S, I, B, T, h, P, \beta)$ is a mechanism $(\mathcal{M})$ and $f \in \{0, 1, ..., n\}$ and $\epsilon > 0$. $(\mathcal{M})$ is a $\epsilon - f$ Nash-Equilibrium for player $i \in [n]$ if $\forall Z \in \mathcal{P}_f([n] - \{i\}), \forall s \in \mathcal{I}_i$, $up_i(Z,s) + \epsilon \geq vp_i(Z \cup \{i\}, s)$.[22]

Meaning of the above definition is that player i has a Nash Equilibrium if the worst case value (when i is altruistic) obtained among the infinite length paths starting from initial states is at most less than ϵ to the optimal value obtained by i, playing rationally.

It is evident that we need to calculate an infinite sum to evaluate the $\epsilon - f$ Nash equilibrium. It would be very useful if we can decide the existence of Nash Equilibrium by looking at sequences of sufficient length. [22] presents counter examples which imply that some $\epsilon - f$ Nash equilibriums cannot be verified by looking at finite length traces. They propose an algorithm to find $\epsilon - f$ Nash Equilibrium with an arbitrary precision (δ). We extend their algorithm such that it supports probabilistic systems.

Here, $I : Boolean \rightarrow \{0,1\}$ is the indicator function where $I(k) = 1 \iff k = true$.

df: Writing the algorithm is finished. Have to define the notations and explain the calculation of expected value

5. CASE STUDY: ANALYZING A PROBABILISTIC SECRET SHARING SCHEME

To validate our algorithm and implementations, we performed a case study on a probabilistic secret sharing scheme [13].

Secret sharing is a way for a group of users to share a *secret*: the secret is distributed among the group members, such that each member has a *share* of the secret; and the secret can only be reconstructed with a sufficient number of shares. In other words, the secret is kept confidential, even in the presence of a limited number of traitors (compromised members or non-cooperating members who prevent the reconstruction of the secret). This property makes secret sharing an ideal scheme for storing high sensitive informations,

Algorithm 1 $CheckNash$(mechanism $\mathcal{M}$, int f, double ϵ , δ)

for all $i \in [n]$ **do**
  Let $k, 4E_i(k) \leq \delta$
  Let $s \in S$ and $Z \in \mathcal{P}([n])\{i\}$
  $v_i^0(Z,s) \leftarrow u_i^0(Z,s) \leftarrow 0$
  for t=1 to k **do**
    $v_i^{t+1}(Z,s) \leftarrow I(i \in Z)\{max_{a_i \in A_i(s)}$
    $E_{a_{[n]-Z} \in A_{[n]-Z}(s)}$
    $min_{a_{Z-\{i\}} \in A_{Z-\{i\}}}$
    $\{h_i(s, < a_{z-\{i\}}, a_i, a_{[n]-Z} >)+$
    $\beta_i^t {v_i}^t(Z,s')|BT(Z,s, < a_{-i}, a_i >, s')\}\}+$
    $I(i \notin Z)\{E_{a_i \in A_i(s)}$
    $E_{a_{[n]-Z-\{i\}} \in A_{[n]-Z-\{i\}}}$
    $min_{a_Z \in A_Z(s)}$
    $\{h_i(s, < a_{[n]-Z-\{i\}}, a_Z, a_i >)+$
    $\beta_i^t {v_i}^t(Z,s')|BT(Z,s, < a_{-i}, a_i >, s')\}\}$
    $u_i^{t+1}(Z,s) \leftarrow I(i \in Z)\{min_{a_i \in A_i(s)}$
    $E_{a_{[n]-Z} \in A_{[n]-Z}(s)}$
    $min_{a_{Z-\{i\}} \in A_{Z-\{i\}}}$
    $\{h_i(s, < a_{z-\{i\}}, a_i, a_{[n]-Z} >)+$
    $\beta_i^t {u_i}^t(Z,s')|BT(Z,s, < a_{-i}, a_i >, s')\}\}+$
    $I(i \notin Z)\{E_{a_i \in A_i(s)}$
    $E_{a_{[n]-Z-\{i\}} \in A_{[n]-Z-\{i\}}}$
    $min_{a_Z \in A_Z(s)}$
    $\{h_i(s, < a_{[n]-Z-\{i\}}, a_Z, a_i >)+$
    $\beta_i^t {u_i}^t(Z,s')|BT(Z,s, < a_{-i}, a_i >, s')\}\}$
  end for
  for t=1 to k **do**
    $vp_i^{t+1}(Z,s_i) = E_{x \in PS(s_i)}(v_i^{k+1}(Z,x))$
    $up_i^{t+1}(Z,s_i) = E_{x \in PS(s_i)}(u_i^{k+1}(Z,x))$
  end for
  $\Delta_i \leftarrow \max\{vp_i^k(Z \cup \{i\}, s_i) - up_i^k(Z,s_i) | s \in I, Z \in \mathcal{P}_f([n]\ \{i\})\}$
  $\epsilon_1(i) \leftarrow \Delta_i - 2E_i(k)$; $\epsilon_2(i) \leftarrow \Delta_i + 2E_i(k)$
  if $\epsilon < \epsilon_1(i)$ **then**
    return FAIL
  end if
end for
if $\forall i \in [n]$ $(\epsilon_2(i) < \epsilon$ **then**
  return PASS with ϵ
else
  return PASS with $(\epsilon + \delta)$
end if

such as encryption keys and launch codes. In the parametric form, a n-m ($n \geq m$) secret sharing ensures that it requires at least m honest players to reconstruct the secret within a group of n players. In other words, less than m compromised players together cannot reconstruct the secret; and less than $n - m$ non-cooperating members together cannot stop the honest players to reconstruct of the secret.

The most well-known secret sharing scheme is the Shamir's secret sharing [25]. However, the scheme does not work when the players are *rational* – a player follows the protocol only if it increases his utility. It is shown that a rational player will not share his secret [13]. In fact, it is proved that in general, in all practical mechanisms for shared-secret reconstruction with an upper bound on the running time, the best strategy for each rational player is doing nothing [13]. Hence, a randomized mechanism (the probabilistic secret sharing scheme) is proposed to ensure that rational players would reveal their shares to reconstruct the secret, i.e., achieving a Nash-Equilibrium with a constant expected running time.

The probabilistic scheme is a 3-3 secret sharing[2]. Given the 3 players, we index each player is indexed with a distinct number in $\{1, 2, 3\}$. We use i to denote one player, then the other two players are $(i+1)\%3$, denoted as i^+, and $(i-1)\%3$, denoted as i^-. There is an issuer who assign each player a share of secret. Assuming the share cannot be changed, for example, it is signed by the issuer. The scheme works as follows.

1. Each player i
   - chooses a bit c_i (c_i can only be 1 or 0) such that $c_i = 1$ with probability α and $c_i = 0$ with probability $1 - \alpha$;
   - chooses a bit $c_{(i,+)}$ ($c_{(i,+)}$ can only be 1 or 0) randomly, so that $c_{(i,+)} = 1$ with probability $1/2$ and $c_{(i,+)} = 0$ with probability $1/2$;
   - calculates $c_{(i,-)} = c_i \oplus c_{(i,+)}$.

   Then player i sends the bit $c_{(i,+)}$ to player i^+ and sends the bit $c_{(i,-)}$ to player i^-. This means that the player i receives a bit $c_{(i^+,-)}$ from player i^+ and a bit $c_{(i^-,+)}$ from player i^-.

2. On receiving $c_{(i^+,-)}$, player i computes $c_{(i^+,-)} \oplus c_i$ and sends it to player i^-. Correspondingly, i receives $c_{((i^+)^+,-)} \oplus c_{i^+}$ from i^+. Since there are only 3 players, player $(i^+)^+$ is the player i^-. Hence $c_{((i^+)^+,-)} \oplus c_{i^+} = c_{(i^-,-)} \oplus c_{i^+}$.

3. When receiving both $c_{(i^-,+)}$ from player i^- (step 1) and receiving $c_{(i^-,-)} \oplus c_{i^+}$ from i^+ (step 2), i computes $p = c_{(i^-,+)} \oplus (c_{(i^-,-)} \oplus c_{i^+}) \oplus c_i$. Note that $c_{(i^-,+)} \oplus (c_{(i^-,-)} \oplus c_{i^+}) \oplus c_i = c_{i^-} \oplus c_{i^+} \oplus c_i$, i.e., $p = c_1 \oplus c_2 \oplus c_3$ (It holds for all players). If $p = c_i = 1$ then player i sends his share to the others.

4. Depending on the value of p and c_i, player i decides whether to send his share.
   - If $p = 1$, i receives 3 shares, then he construct the secret.
   - If $p = 0$ and i received no secret shares, or if $p = 1$ and i received exactly one share (possibly from itself; that is, we allow the case that i did not receive any shares from other players but sent its own), the issuer restarts the protocol.

[2]A general n-m secret sharing is considered as variations of the basic 3-3 secret sharing (for details, see [13])

- Otherwise, player i stops the protocol. Because someone must have been cheating.

The intuition behind the last step is as follows: When $p = 0$, no one should send his share; and thus i receives no secret share. Otherwise, some one calculated wrong (cheating). When $p = 1$ either $c_i = c_{i^+} = c_{i^-} = 1$ or there exists exactly one player chooses 1, i.e., $c_i = c_{i^+} = 0$ and $c_{i^-} = 1$, $c_i = c_{i^-} = 0$ and $c_{i^+} = 1$, or $c_{i^+} = c_{i^-} = 0$ and $c_i = 1$. In the former case, each player can construct the secret. In the later case, i should receive exactly one share. Otherwise, some player calculated wrong (cheating).

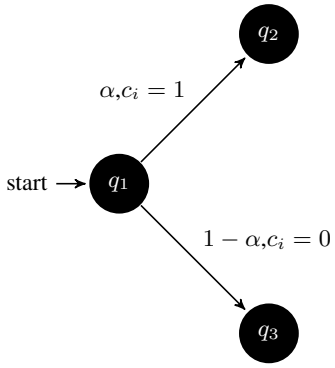
Model.

The above randomized version of the shamir secret sharing scheme can be modelled with the probabilistic extension proposed.

Here, we assume that message channels are secure and sender's authenticity is preserved. $(S, I, A, B, T, h, P, PA, \beta)$. The above protocol is a deterministic protocol where at each step an honest player has to perform a single predefined task(may be probabilistic), depending on the global state he/she is in. This scenario has a unique initial global state. Let's say player i's initial state is s_{i0}. The initial global state is (s_{10}, s_{20}, s_{30}). $I = \{(s_{10}, s_{20}, s_{30})\}$ Let's have a look at their Byzantine behaviours starting from the initial state.

1. Player i's probabilistic action to choose c_i. $c_i = 1$ with probability α.
2. choose c_{i^+} with 0.5 probability.
3. send messages to (i^+ and i^-. (since the messages are boolean, player has four different actions)
4. send 'xor' to player i^-. (two options - sending correct one or incorrect one)
5. send or not send the messages to player i^- and i^+. (total four possibilities)
6. restart the protocol

The above actions sets are activated in different levels. For example, first action set starts from the initial state.



In this example, the set $Z = \{\}$ so that each player is either Altruistic or Rational. The pay-off function is only defined for actions which end in terminal states. For all other actions pay-off is simply 0.

6. EXPERIMENTAL RESULTS

Experiments are based on the example mechanism presented in section 1. Exact same mechanism is tested for running time and memory performance for different numbers of players and different Byzantine set sizes. In all cases $\epsilon = 0.01$ and $\delta = 0.005$. Experiments were run in 64-bit x-86 Intel machine having 2.40 GHz processor with 8GB RAM. Presentation of the results has a similar format to [22].

7. CONCLUSIONS AND FUTURE WORKS

8. REFERENCES

[1] I. Abraham, L. Alvisi, and J. Y. Halpern. Distributed computing meets game theory: combining insights from two fields. *SIGACT News*, 42(2):69–76, 2011.

[2] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. MOCHA: modularity in model checking. In *Proc. 10th Computer Aided Verification*, volume 1427 of *LNCS*, pages 521–525. Springer, 1998.

[3] M. Backes, O. Ciobotaru, and A. Krohmer. Ratfish: A file sharing protocol provably secure against rational users. In *Proc. 15th European Symposium on Research in Computer Security*, volume 6345 of *LNCS*, pages 607–625. Springer, 2010.

[4] P. Ballarini, M. Fisher, and M. Wooldridge. Automated game analysis via probabilistic model checking: a case study. *Electr. Notes Theor. Comput. Sci.*, 149(2):125–137, 2006.

[5] R. Brenguier. PRALINE: A tool for computing nash equilibria in concurrent games. In *Proc. 25th Computer Aided Verification*, volume 8044 of *LNCS*, pages 890–895. Springer, 2013.

[6] T. Chen, V. Forejt, M. Z. Kwiatkowska, D. Parker, and A. Simaitis. Prism-games: A model checker for stochastic multi-player games. In *Proc. 19th Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *LNCS*, pages 185–191. Springer, 2013.

[7] C. Cheng, A. Knoll, M. Luttenberger, and C. Buckl. GAVS+: an open platform for the research of algorithmic game solving. In *Proc. 17th Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *LNCS*, pages 258–261. Springer, 2011.

[8] C. Daskalakis. On the complexity of approximating a nash equilibrium. *ACM Transactions on Algorithms*, 9(3):23, 2013.

[9] C. Daskalakis, A. Mehta, and C. H. Papadimitriou. Progress in approximate nash equilibria. In *Proc. 8th ACM Conference on Electronic Commerce*, pages 355–358. ACM, 2007.

[10] L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. *J. Comput. Syst. Sci.*, 68(2):374–397, 2004.

[11] A. Fabrikant, C. H. Papadimitriou, and K. Talwar. The complexity of pure nash equilibria. In *Proc. 36th Annual ACM Symposium on Theory of Computing*, pages 604–612. ACM, 2004.

[12] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, Cambridge, MA, 1991.

[13] J. Halpern and V. Teague. Rational secret sharing and multiparty computation: Extended abstract. In *Proc. 36th Annual ACM Symposium on Theory of Computing*, pages 623–632. ACM, 2004.

Table 1: Experimental Results

Agents	Jobs	Byzantines	Nash	CPU(sec)	Mem(MB)	Present State Bits	Action Bits
3	2	1	PASS	7.8405		9	3
3	2	2	PASS	16.7238		9	3
3	2	3	PASS	7.6124		9	3
4	2	1	PASS	979.3278		12	4
4	2	2	PASS	1667.687		12	4
4	2	3	PASS	1892517		12	4

[14] T. A. Henzinger. Model checking game properties of multi-agent systems (abstract). In *Proc. 25th Automata, Languages and Programming*, volume 1443 of *LNCS*, page 543. Springer, 1998.

[15] S. Kremer and J. Raskin. A game-based verification of non-repudiation and fair exchange protocols. *Journal of Computer Security*, 11(3):399–430, 2003.

[16] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification*, CAV'11, pages 585–591, 2011.

[17] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR gossip. In *Proc. 7th Symposium on Operating Systems Design and Implementation*, pages 191–204. USENIX Association, 2006.

[18] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard. A cooperative internet backup scheme. In *Proc. the General Track: 2003 USENIX Annual Technical Conference*, pages 29–41. USENIX, 2003.

[19] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proc. 21st Computer Aided Verification*, volume 5643 of *LNCS*, pages 682–688. Springer, 2009.

[20] A. Lomuscio and F. Raimondi. Model checking knowledge, strategies, and games in multi-agent systems. In *Proc. 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 161–168. ACM, 2006.

[21] A. Mahimkar and V. Shmatikov. Game-based analysis of denial-of-service prevention protocols. In *Proc. 18th IEEE Computer Security Foundations Workshop*, pages 287–301. IEEE Computer Society, 2005.

[22] F. Mari, I. Melatti, I. Salvo, E. Tronci, L. Alvisi, A. Clement, and H. C. Li. Model checking nash equilibria in MAD distributed systems. In *Formal Methods in Computer-Aided Design*, pages 1–8. IEEE, 2008.

[23] P. project. Pat: Process analysis toolkit. pat.comp.nus.edu.sg, last visted at 25 Feb. 2016.

[24] M. O. Rabin. Transaction protection by beacons. *J. Comput. Syst. Sci.*, 27(2):256–267, 1983.

[25] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[26] J. Shneidman and D. C. Parkes. Specification faithfulness in networks with rational nodes. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing*, pages 88–97. ACM, 2004.

[27] A. Toumi, J. Gutierrez, and M. Wooldridge. A tool for the automated verification of nash equilibria in concurrent games. In *Proc. 12th Theoretical Aspects of Computing*, volume 9399 of *LNCS*, pages 583–594. Springer, 2015.

[28] Y. Zhang, C. Zhang, J. Pang, and S. Mauw. Game-based verification of contract signing protocols with minimal messages. *ISSE*, 8(2):111–124, 2012.