Analysis of Probabilistic Contract Signing*

Gethin Norman[1] and Vitaly Shmatikov[2]

[1]University of Birmingham, School of Computer Science,
Birmingham B15 2TT U.K.
email: G.Norman@cs.bham.ac.uk

[2]Department of Computer Sciences, The University of Texas at Austin,
Austin, TX 78712 U.S.A.
email: shmat@cs.utexas.edu
(this research was performed while at SRI International)

Abstract

We present three case studies, investigating the use of probabilistic model checking to automatically analyse properties of probabilistic contract signing protocols. We use the probabilistic model checker PRISM to analyse three protocols: Rabin's probabilistic protocol for fair commitment exchange; the probabilistic contract signing protocol of Ben-Or, Goldreich, Micali, and Rivest; and a randomised protocol for signing contracts of Even, Goldreich, and Lempel. These case studies illustrate the general methodology for applying probabilistic model checking to formal verification of probabilistic security protocols.

For the Ben-Or *et al.* protocol, we demonstrate the difficulty of combining fairness with timeliness. If, as required by timeliness, the judge responds to participants' messages immediately upon receiving them, then there exists a strategy for a misbehaving participant that brings the protocol to an unfair state with arbitrarily high probability, unless unusually strong assumptions are made about the quality of the communication channels between the judge and honest participants. We quantify the tradeoffs involved in the attack strategy, and discuss possible modifications of the protocol that ensure both fairness and timeliness.

For the Even *et al.* protocol, we demonstrate that the responder enjoys a distinct advantage. With probability 1, the protocol reaches a state in which the responder possesses the initiator's commitment, but the initiator does not possess the responder's commitment. We then analyse several variants of the protocol, exploring the tradeoff between fairness and the number of messages that must be exchanged between participants.

Keywords: Security Protocols, Contract Signing, Probabilistic Model Checking.

1 Introduction

Consider several parties on a computer network who wish to exchange some items of value but do not trust each other to behave honestly. *Fair exchange* is the

*Corresponding author: Gethin Norman, University of Birmingham, School of Computer Science, Edgbaston, Birmingham, B15 2TT, U.K., phone: +44-121-4144789, fax: +44-121-4144281, email: G.Norman@cs.bham.ac.uk

problem of exchanging data in a way that guarantees that either all participants obtain what they want, or none do. *Contract signing* is a particular form of fair exchange, in which the parties exchange commitments to a *contract* (typically, a text string spelling out the terms of the deal). Commitment is often identified with the party's digital signature on the contract. In commercial transactions conducted in a distributed environment such as the Internet, it is sometimes difficult to assess a counterparty's trustworthiness. Contract signing protocols are, therefore, an essential piece of the e-commerce infrastructure.

Contract signing protocols. The main property a contract signing protocol should guarantee is *fairness*. Informally, a protocol between A and B is fair for A if, in any situation where B has obtained A's commitment, A can obtain B's commitment regardless of B's actions. Ideally, fairness would be guaranteed by the simultaneous execution of commitments by the parties. In a distributed environment, however, simultaneity cannot be assured unless a trusted third party is involved in every communication. Protocols for contract signing are inherently asymmetric, requiring one of the parties to make the first move and thus put itself at a potential disadvantage in cases where the other party misbehaves.

Another important property of contract signing protocols is *timeliness*, or timely termination [4]. Timeliness ensures, roughly, that the protocol does not leave any participant "hanging" in an indeterminate state, not knowing whether the exchange of commitments has been successful. In a timely protocol, each participant can terminate the protocol timely and unilaterally, *e.g.*, by contacting a trusted third party and receiving a response that determines the status of the exchange.

Research on fair contract signing protocols dates to the early work by Even and Yacobi [17] who proved that fairness is impossible in a deterministic two-party contract signing protocol. Since then, there have been proposed randomised contract signing protocols based on a computational definition of fairness [15, 16], protocols based on gradual release of commitments [12, 8], as well as non-probabilistic contract signing protocols that make *optimistic* use of the trusted third party (a.k.a. *judge*). In an optimistic protocol, the trusted third party is invoked only if one of the participants misbehaves [4, 18]. In this paper, we focus on *probabilistic* contract signing, exemplified by the probabilistic contract signing protocol of Ben-Or, Goldreich, Micali, and Rivest [6] (henceforth, the `BGMR` protocol). To illustrate our general methodology for analysing probabilistic fair exchange protocols, we also consider the contract signing protocol of Even, Goldreich, and Lempel [16] (henceforth, the `EGL` protocol), and Rabin's protocol for probabilistic fair exchange [30].

Related work. A variety of formal methods have been successfully applied to the study of *nondeterministic* contract signing protocols, including finite-state model checking [32], alternating transition systems [23, 24], and game-theoretic approaches [9, 11, 10]. None of these techniques, however, are applicable to contract signing in a *probabilistic* setting. Since fairness in protocols such as Rabin's and `BGMR` is a fundamentally probabilistic property, these protocols can only be modelled with a probabilistic formalism such as Markov decision processes and verified only with probabilistic verification tools. This is a novel line of research, and very few results have been published. For non-repudiation (a restricted case of contract signing), Aldini and Gorrieri [1] used a probabilistic process algebra to analyse the fairness guarantees of the probabilistic non-repudiation protocol of Markowitch and Roggeman [26]

Even for non-fairness properties such as secrecy, authentication, anonymity, *etc.*, formal techniques for the analysis of security protocols have focused almost exclusively on nondeterministic attacker models. Attempts to incorporate prob-

ability into formal models have been limited to probabilistic characterization of non-interference [20, 33, 34], and process formalisms that aim to represent probabilistic properties of cryptographic primitives [25]. This paper is an attempt to demonstrate how fully automated probabilistic analysis techniques can be used to give a quantitative characterization of probability-based security properties.

2 Technique and Main Results

Our main contribution is a demonstration of how probabilistic verification techniques can be applied to the analysis of fairness properties of security protocols. To model the actions of an arbitrarily misbehaving participant, we endow him with nondeterministic attacker operations in addition to the probabilistic behaviour prescribed by the protocol specification. The resulting model for the protocol combines nondeterminism and probability, giving rise to a Markov decision process, which is then analysed with PRISM [22, 29], a probabilistic model checker. A preliminary version of this paper, consisting of sections 2-7 (analysis of the `BGMR` protocol), was published in conference proceedings as [28].

Quantifying fairness. We construct Markov decision process models for `BGMR`, Rabin's, and `EGL` protocols, and then use PRISM to calculate the probability that each protocol terminates in a fair state as a function of the number of message rounds. For the `EGL` protocol, we also explore several variants based on different message scheduling schemes. For each variant, we use PRISM to quantify the computational disadvantage of the losing party in the unfair state, formalised as the number of bits he needs to guess before he can compute the opponent's commitment. We also calculate the number of messages that must be exchanged before fairness is restored, *i.e.*, to reach a state in which both parties know each other's commitment from a state in which only one party knows the opponent's commitment.

Fairness and timeliness in the BGMR protocol. The `BGMR` protocol as specified in [6] consists of two phases: the "negotiation" phase of pre-agreed duration, in which participants exchange their partial commitments to the contract, and the "resolution" phase, in which the judge issues decisions in case one or both of the participants contacted him during the negotiation phase. The `BGMR` protocol does not guarantee timeliness. On the one hand, the negotiation phase should be sufficiently long to enable two honest participants to complete the exchange of commitments without involving the judge. On the other hand, if something goes wrong (*e.g.*, a dishonest party stops responding), the honest party may contact the judge, but then has to wait until the entire period allotted for the negotiation phase is over before he receives the judge's verdict and learns whether the contract is binding on him or not.

We study a variant of the `BGMR` protocol that attempts to combine fairness with timeliness by having the judge respond immediately to participants' messages, in the manner similar to state-of-the-art non-probabilistic contract signing protocols such as the optimistic protocols of Asokan *et al.* [4] and Garay *et al.* [18]. Our analysis uncovers that, for this variant of the `BGMR` protocol, fairness is guaranteed only if the judge can establish a communication channel with A, the initiator of the protocol, and deliver his messages faster than A and B are communicating with each other. If the channel from the judge to A provides no timing guarantees, or the misbehaving B controls the network and (passively) delays the judge's messages, or it simply takes a while for the judge to locate A (the judge knows A's identity, but they may have never communicated before), then B can exploit the fact that the

judge does not remember his previous verdicts and bring the protocol to an unfair state with arbitrarily high probability.

We quantify the tradeoff between the magnitude of this probability and the expected number of message exchanges between A and B before the protocol reaches a state which is unfair to A. Informally, the longer B is able to delay the judge's messages (and thus continue communicating with A, who is unaware of the judge's attempts to contact him), the higher the probability that B will be able to cheat A.

Fairness and computational disadvantage in the EGL protocol. In the `EGL` protocol [16], each party starts by generating a sequence of pairs of secrets and revealing the encryptions of the contract under each secret. A party is considered committed if the opponent knows both secrets in at least one pair. In the first phase, the parties use oblivious transfer to probabilistically reveal one secret from each pair. In the second phase, they release all secrets bit by bit.

For this protocol as specified in [16], we demonstrate that, with probability 1, it reaches a state in which the responder (second mover) knows both secrets from at least one of the initiator's pairs, but the initiator does not know both secrets from any of the responder's pairs. We use PRISM to quantify the initiator's computational disadvantage by calculating the expected number of bits he needs to guess to complete one of the responder's pairs. For the case when the responder does not stop communicating immediately after reaching a point of advantage, we calculate the expected number of messages that must be exchanged before fairness is restored, that is, the initiator learns both secrets in at least one of the responder's pairs.

We also consider alternative message scheduling schemes for the second phase of the `EGL` protocol, and present fairness analysis for several variants. We are interested in two questions. First, for a given party P (initiator or responder), what is the probability of reaching a state where the *other* party knows both secrets from one of P's pairs, but P does not know both secrets from one of the opponent's pairs? (Obviously, we would like this probability to be as close to $\frac{1}{2}$ as possible). Second, after an "unfair" state is reached, how long does it take before fairness is restored, that is, how many bits does the "losing" party need to receive before he knows both of the opponent's secrets?

3 Probabilistic Model Checking

Probability is widely used in the design and analysis of software and hardware systems: as a means to derive efficient algorithms (*e.g.*, the use of electronic coin flipping in decision making); as a model for unreliable or unpredictable behaviour (*e.g.*, fault-tolerant systems, computer networks); and as a tool to analyse system performance (*e.g.*, the use of steady-state probabilities in the calculation of throughput and mean waiting time). *Probabilistic model checking* (see *e.g.*, [31]) refers to a range of techniques for calculating the likelihood of the occurrence of certain events during the execution of such a system, and can be useful to establish performance measures such as "shutdown occurs with probability at most 0.01" and "the video frame will be delivered within 5ms with probability at least 0.97". The system is usually specified as state transition system, with probability measures on the rate of transitions, and a probabilistic model checker applies algorithmic techniques to analyse the state space and calculate performance measures.

In the distributed scenario, in which concurrently active processors handle a great deal of unspecified nondeterministic behaviour exhibited by their environment, the state transition systems must include both probabilistic and nondeterministic

behaviour. Standard models of such systems are Markov decision processes (MDPs) [13]. Properties of MDPs can be specified in the probabilistic branching-time temporal logic PCTL [21, 7] which allows one to express properties such as "under any scheduling of nondeterministic choices, the probability of ϕ holding until ψ is true is *at least 0.78/at most 0.04*".

3.1 PRISM model checker

We use PRISM [22, 29], a probabilistic model checker developed at the University of Birmingham. The current implementation of PRISM supports the analysis of *finite*-state probabilistic models of the following three types: discrete-time Markov chains, continuous-time Markov chains and Markov decision processes. These models are described in a high-level language, a variant of reactive modules [2] based on guarded commands. The basic components of the language are *modules* and *variables*. A system is constructed as a number of modules which can interact with each other. A module contains a number of variables which express the state of the module, and its behaviour is given by a set of guarded commands of the form:

$$[] \texttt{<guard>} \rightarrow \texttt{<command>};$$

The guard is a predicate over the variables of the system and the command describes a transition which the module can make if the guard is true (using primed variables to denote the next values of variables). If a transition is probabilistic, then the command is specified as:

$$\texttt{<prob>} : \texttt{<command>} + \cdots + \texttt{<prob>} : \texttt{<command>}$$

PRISM accepts specifications in either PCTL, or CSL logic depending on the model type. This allows us to express various probabilistic properties such as "some event happens with probability 1", and "the probability of cost exceeding C is 95%". The model checker then analyses the model and checks if the property holds in each state. In the case of MDPs, specifications are written in the logic PCTL, and for the analysis PRISM implements the algorithms of [21, 7, 5].

4 Rabin's, BGMR, and TBGMR Protocols

Rabin's and `BGMR` protocols involve the trusted third party, while the `EGL` protocol is a two-party protocol. Since the structure of Rabin's and `BGMR` protocols is very similar, we describe and analyse them together. We also consider a timely variant of `BGMR` which we call the `TBGMR` protocol. The `EGL` protocol is described and analysed in section 8.

4.1 Rabin's protocol

Rabin's protocol [30] enables two parties, A and B, to exchange their commitments to a pre-defined contract C with the help of a trusted third party. The protocol is *not* optimistic: an action of the third party is required in every instance of the protocol. Every day, the third party must publicly broadcast a randomly chosen integer between 1 and N.

Before beginning the exchange, the parties agree on some future cutoff date D. A then sends to B message "sig_A(*I am committed to C, if integer i is chosen on date D*)," where i is initially 1. When B receives this message, he responds with "sig_B(*I am committed to C, if integer i is chosen on date D*)," to which A responds with "sig_A(*I am committed to C, if integer* $i+1$ *is chosen on date D*)," and so on. The exchange must complete before date D.

Intuitively, for any integer i between 1 and N, if the last message in the exchange was from B, then A and B have equal probability of being committed after the third party randomly chooses and broadcasts a number. If the number is less than or equal to i, then both parties are committed. If the number is greater than i, then neither party is committed.

Now suppose the last message in the exchange was from A. If the number broadcast by the third party is less than or equal to i, then both parties are committed. If it is greater than $i+1$, then neither is committed. If the number is exactly equal to $i+1$, then A is committed, but B is not. Therefore, the probability that the protocol terminates in a state in which only one party is committed is $\frac{1}{N}$.

4.2 BGMR protocol

The goal of the probabilistic contract signing protocol of Ben-Or, Goldreich, Micali, and Rivest [6] (the `BGMR` protocol) is the same as that of Rabin's protocol: to enable A and B to exchange their commitments to a pre-defined contract C. It is assumed that there exists a third party, called the *judge*, who is trusted by both A and B. Unlike Rabin's protocol, the `BGMR` protocol is *optimistic*. An honest participant following the protocol specification only has to invoke the judge if something goes wrong, *e.g.*, if the other party stops before the exchange of commitments is complete (a similar property is called *viability* in [6]). Optimism is a popular feature of fair exchange protocols [27, 3, 4]. In cases where both signers are honest, it enables contract signing to proceed without participation of a third party, and thus avoids communication bottlenecks inherent in protocols that involve a trusted authority in every instance.

Privilege and fairness. In the `BGMR` protocol, it can never be the case that the contract is binding on one party, but not the other. Whenever the judge declares a contract binding, the verdict always applies to *both* parties. For brevity, we will refer to the judge's ruling on the contract as *resolving* the contract.

Privilege is a fundamental notion in the `BGMR` protocol. A party is privileged if it has the power to cause the judge to rule that the contract is binding. The protocol is unfair if it reaches a state where one party is privileged (*i.e.*, it can cause the judge to declare the contract binding), and the other is not.

Definition 1 (Probabilistic fairness) *A contract signing protocol is* $(v, \varepsilon)-$fair *for A if, for any contract C, if A follows the protocol, then at any step of the protocol in which the probability that B is privileged is greater than v, the conditional probability that A is* not *privileged given that B is privileged is at most ε.*

The fairness condition for B is symmetric.

Probabilistic fairness means that at any step of the protocol where one of the parties has acquired the evidence that will cause the judge to declare the contract binding with probability v, the other party should possess the evidence that will cause the judge to issue the same ruling with probability of no less than $v - \varepsilon$. Informally, ε can be interpreted as the maximum *fairness gap* between A and B permitted at any step of the protocol.

Main protocol. Prior to initiating the protocol, A chooses a probability v which is sufficiently small so that A is willing to accept a chance of v that B is privileged while A is not.

A also chooses a value $\alpha > 1$ which quantifies the "fairness gap" as follows: at each step of the protocol, the conditional probability that A is privileged given that B is privileged should be at least $\frac{1}{\alpha}$, unless the probability that B is privileged is less

than v. B also chooses a value $\beta > 1$ such that at any step where A is privileged, the conditional probability that B is privileged should be at least $\frac{1}{\beta}$. Both parties maintain counters, λ_a and λ_b, initialised to 0.

A's commitment to C has the form "$\mathrm{sig}_A($*With probability* 1*, the contract* C *shall be valid*$)$". B's commitment is symmetric. It is assumed that the protocol employs an unforgeable digital signature scheme.

All messages sent by A in the main protocol have the form "$\mathrm{sig}_A($*With probability* p*, the contract* C *shall be valid*$)$". Messages sent by B have the same form and are signed by B. If both A and B behave correctly, at the end of the main protocol A obtains B's commitment to C, and vice versa. At the abstract level, the main flow of the `BGMR` protocol is as follows:

$$
\begin{array}{cccrcl}
A & \rightarrow & B & \mathrm{sig}_A(\textit{With probability } p_1^a\textit{, the contract } C \textit{ shall be valid}) & = & m_1^a \\
A & \leftarrow & B & \mathrm{sig}_B(\textit{With probability } p_1^b\textit{, the contract } C \textit{ shall be valid}) & = & m_1^b \\
 & & & \vdots & & \\
A & \rightarrow & B & \mathrm{sig}_A(\textit{With probability } p_i^a\textit{, the contract } C \textit{ shall be valid}) & = & m_i^a \\
 & & & \vdots & & \\
A & \rightarrow & B & \mathrm{sig}_A(\textit{With probability } 1\textit{, the contract } C \textit{ shall be valid}) & = & m_n^a \\
A & \leftarrow & B & \mathrm{sig}_B(\textit{With probability } 1\textit{, the contract } C \textit{ shall be valid}) & = & m_n^b
\end{array}
$$

In its first message m_1^a, A sets $p_1^a = v$.

Consider the ith round of the protocol. After receiving message "$\mathrm{sig}_A($*With probability* p_i^a*, the contract* C *shall be valid*$)$" from A, honest B checks whether $p_i^a \geq \lambda_b$. If not, B considers A to have stopped early, and contacts the judge for resolution of the contract as described below. If the condition holds, B computes $\lambda_b = \min(1, p_i^a \cdot \beta)$, sets $p_i^b = \lambda_b$ and sends message "$\mathrm{sig}_B($*With probability* p_i^b*, the contract* C *shall be valid*$)$" to A.

The specification for A is similar. Upon receiving B's message with probability p_i^b in it, A checks whether $p_i^b \geq \lambda_a$. If not, A contacts the judge, otherwise he updates $\lambda_a = \max(v, \min(1, p_i^b \cdot \alpha))$, sets $p_{i+1}^a = \lambda_a$ and sends message "$\mathrm{sig}_A($*With probability* p_{i+1}^a*, the contract* C *shall be valid*$)$," initiating a new round of the protocol.

The main protocol is optimistic. If followed by both participants, it terminates with both parties committed to the contract.

The judge. Specification of the `BGMR` protocol assumes that the contract C defines a cutoff date D. When the judge is invoked, he does nothing until D has passed, then examines the message of the form "$\mathrm{sig}_X($*With probability* p*, the contract* C *shall be valid*$)$" supplied by the party that invoked it and checks the validity of the signature.

If the judge has not resolved contract C before, he flips a coin, *i.e.*, chooses a random value ρ_C from a uniform distribution over the interval $[0, 1]$. If the contract has been resolved already, the judge retrieves the previously computed value of ρ_C. In either case, the judge declares the contract binding if $p \geq \rho_C$ and cancelled if $p < \rho_C$, and sends his verdict to the participants. To make the protocol more efficient, ρ_C can be computed as $f_r(C)$, where r is the judge's secret input, selected once and for all, and f_r is the corresponding member of a family of pseudo-random functions [19]. This enables the judge to produce the same value of ρ_C each time contract C is submitted without the need to remember his past flips. The judge's procedure can thus be implemented in constant memory.

Observe that even though the judge produces the same value of ρ_C each time C is submitted, the judge's *verdict* depends also on the value of p in the message submitted by the invoking party and, therefore, may be different each time. If the judge is optimised using a pseudo-random function with a secret input to work in constant memory as described above, it is impossible to guarantee that he will produce the same verdict each time. To do so, the judge needs to remember the first verdict for each contract ever submitted to him, and, unlike ρ_C, the value of this verdict cannot be reconstructed from subsequent messages related to the same contract.

4.3 Timely BGMR

Asokan *et al.* [4] define *timeliness* as "one player cannot force the other to wait for any length of time—a fair and timely termination can be forced by contacting the third party." The `BGMR` protocol as specified in Section 4.2 does not guarantee timeliness in this sense. To accommodate delays and communication failures on a public network such as the Internet, the duration of the negotiation phase D should be long. Otherwise, many exchanges between honest parties will not terminate in time and will require involvement of the judge, making the judge a communication bottleneck and providing no improvement over a protocol that simply channels all communication through a trusted central server.

If D is long, then an honest participant in the `BGMR` protocol can be left "hanging" for a long time. Suppose the other party in the protocol stops communicating. The honest participant may contact the judge, of course, but since the judge in the original `BGMR` protocol does not flip his coin until D has passed, this means that the honest party must wait the *entire* period allotted for negotiation before he learns whether the contract will be binding on him or not. This lack of timeliness is inconvenient and potentially problematic if the contract requires resource commitment from the participants or relies on time-sensitive data.

In this paper, we investigate a variant of `BGMR` that we call `TBGMR` (for "Timely `BGMR`"). The only difference between `BGMR` and `TBGMR` is that, in `TBGMR`, the judge makes his decision *immediately* when invoked by one of the protocol participants. Once the verdict is announced and reaches an honest participant, the latter stops communicating. The rest of the protocol is as specified in Section 4.2. `TBGMR` protocol is timely. Any party can terminate it unilaterally and obtain a binding verdict at any point in the protocol without having to wait for a long time.

5 Model for Rabin's, BGMR, and TBGMR protocols

We formalise Rabin's, `BGMR` and `TBGMR` protocols as Markov decision processes. Since these models have infinite state-spaces, we discretise the probability space of each model and then analyse chosen finite configurations with PRISM. We use PRISM to calculate the probability of each protocol terminating in a fair state as a function of the number of rounds, and to determine whether `TBGMR` is fair or not. For simplicity, we will refer to the third party in all protocols as the *judge*.

5.1 Overview of the model

First, we model the normal behaviour of protocol participants and the judge according to the protocol specification, except that in the model for `TBGMR` the judge makes his coin flip and responds with a verdict immediately rather than waiting for the cutoff date D. A dishonest participant might be willing to deviate from the protocol

in an attempt to cheat the honest participant. We equip the dishonest participant with an additional set of dishonest actions, any of which he can take nondeterministically at any point in the protocol. To obtain a finite probabilistic model, we fix the parameters chosen by the participants, and discretise the probability space of the judge's coin flips.

Modelling the dishonest participant. Conventional formal analysis of security protocols is mainly concerned with security against the so called *Dolev-Yao attacker*, following [14]. A Dolev-Yao attacker is a nondeterministic process that has complete control over the communication network and can perform any combination of a given set of attacker operations, such as intercepting any message, splitting messages into parts, decrypting if he knows the correct decryption key, assembling fragments of messages into new messages and replaying them out of context, *etc.*

We assume that the communication channels between protocol participants ensure authentication and integrity of all messages (in practice, this is usually achieved by employing cryptographically secure digital signatures and message authentication codes). Therefore, it is impossible for the misbehaving participant to forge a message from an honest participant, nor modify the content or origin of an existing message. We will also assume that the channels between the participants and the judge are *resilient*: it is possible for the attacker to delay messages and schedule them out of order, but he must eventually deliver every message to its intended recipient.

Due to our assumptions about the communication channels, our adversary model is strictly weaker than the Dolev-Yao model. A misbehaving participant cannot completely remove other participants' messages from the network, nor introduce forged messages from other participants, nor modify their existing messages. Dishonest actions available to a misbehaving participant are limited to (i) stopping prematurely, and, in `BGMR` and `TBGMR` protocols, (ii) invoking the judge even though the honest party has not stopped communicating in the main flow of the protocol, and (iii) delaying and re-ordering messages between the judge and the honest party. A misbehaving participant can nondeterministically attempt any of these actions at any point in the protocol. When combined with the probabilistic behaviour of the judge, the nondeterminism of the dishonest participant gives rise to a Markov decision process.

Modelling fairness. To model fairness in `BGMR` and `TBGMR` protocols, we compute the maximum probability of reaching a state where the dishonest participant is privileged and the honest one is not. Note that this probability is *not* conditional on the judge's coin not having been flipped yet, because the dishonest participant may choose to contact the judge even after the coin has been flipped, ρ_C computed and verdict rendered. In contrast, the proof of fairness in [6] calculates this probability under the assumption that the coin has not been flipped yet.

5.2 Model for Rabin's protocol

To ensure a finite model, we fix the value of $N \in \mathbb{N}$ in advance, that is, fix N such that when the judge chooses the random value each day, it takes the value i for $i = 1, \ldots, N$ with probability $1/N$. The resulting model for Rabin's protocol is given in Figure 1 for the case $N = 100$.

5.3 Model for BGMR and TBGMR protocols

Since PRISM is currently only applicable to finite configurations and the input language allows only integer valued variables, we must discretise the judge's coin

```
// non-deterministic choice between arrival of date D and message sending
// models the fact that a party can stop sending messages at any time

module parties

    mA : [0..100]; // number in the last message A sent to B
    mB : [0..100]; // number in the last message B sent to A
    turn : [0..1]; // which party last sent a message (0 – B did, 1 – A did)

    // can only send messages while d = 0 (i.e., before cutoff date arrives)
    [] turn=0 ∧ mA<100 ∧ d=0 → (turn′=1) ∧ (mA′=mA+1); // A sends
    [] turn=1 ∧ mB<100 ∧ d=0 → (turn′=0) ∧ (mB′=mA); // B sends

endmodule

module date

    d : [0..1];

    [] d=0 → (d′=1); // cutoff date arrives

endmodule

module third_party

    i : [0..100]; // randomly chosen integer

    [] i=0 ∧ d=1 → 1/100 : (i′=1)
                   +1/100 : (i′=2)
                          ⋮
                   +1/100 : (i′=100);

endmodule
```

Figure 1: PRISM code for Rabin's protocol.

by fixing some $N \in \mathbb{N}$ and suppose that when the judge flips the coin, it takes the value i/N for $i = 1, \ldots, N$ with probability $1/N$. Furthermore, we must fix the parameters of the parties, namely v, α and β.

Once the parameters are fixed, the possible messages that can be sent between the parties and the ordering of the messages are predetermined. More precisely, as shown in Figure 2, the probability values included in the messages between the parties are known. Note that, for $v, \alpha, \beta > 0$, these values converge to 1 and there are only finitely many distinct values. Therefore, with a simple script, we can calculate all the probability values included in the messages of the two parties.

To model the parties in PRISM, we encode the state of each party as the probability value included in the last message the party sent to the other party, *i.e.*, the states of A and B are identified by the current values of λ_a and λ_b respectively. Formally, the states of each party range from 0 up to k, for some k such that $p_i^a, p_i^b = 1$ for all $i \geq k$, where A is in state 0 when A has sent no messages to B, and in state $i \in \{1, \ldots, k\}$ when the last message A sent to B included the probability value $\min(1, v \cdot \alpha^{i-1} \cdot \beta^{i-1})$. Similarly, B is in state 0 when B has not sent any messages to A, and in state $i \in \{1, \ldots, k\}$ when the last message B sent to A included the probability value $\min(1, v \cdot \alpha^{i-1} \cdot \beta^{i})$.

Following this construction process, the specification of the `BGMR` protocol used as input into PRISM in the case when $N = 100$ is given in Figure 3.

Note that there is a nondeterministic choice as to when the judge's coin is flipped, *i.e.*, when a party first sends a message to the judge asking for a verdict. Furthermore, once the coin is flipped, the parties cannot send any messages to each other, that is, this model corresponds to the original protocol.

In the timely variant of the protocol (`TBGMR`), if the misbehaving participant is

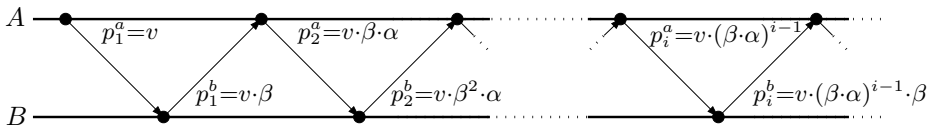


Figure 2: Probability values included in the messages sent between the parties (BGMR protocol).

```
// non-deterministic choice between flipping of coin and message sending
// models the fact that a party can stop sending messages (and contact the judge)

module parties

    lambdaA : [0..k]; // probability value in the last message A sent to B
    lambdaB : [0..k]; // probability value in the last message B sent to A
    turn : [0..1]; // which party sends message next (0 – party A, 1 – party B)

    // parties alternately send messages and stop when the coin has been flipped
    [] turn=0 ∧ c=0 → (lambdaA′=min(k, lambdaA + 1)) ∧ (turn′=1);
    [] turn=1 ∧ c=0 → (lambdaB′=min(k, lambdaB + 1)) ∧ (turn′=0);

endmodule

module judge

    c : [0..1]; // status of the coin (0 – not flipped and 1 – flipped)
    rho : [0..100]; // value of the coin (rho=x and c=1 means its value is x/100)

    // flip coin any time after a message has been sent
    [] c=0 ∧ (lambdaA>0 ∨ lambdaB>0) → 1/100 : (c′=1) ∧ (rho′=1)
                                      +1/100 : (c′=1) ∧ (rho′=2)
                                      ⋮
                                      +1/100 : (c′=1) ∧ (rho′=100);

endmodule
```

Figure 3: PRISM code for BGMR protocol.

capable of delaying messages from the judge, then the parties may continue sending messages to each other even after the coin has been flipped. To model this, the two lines corresponding to the parties sending messages are replaced with:

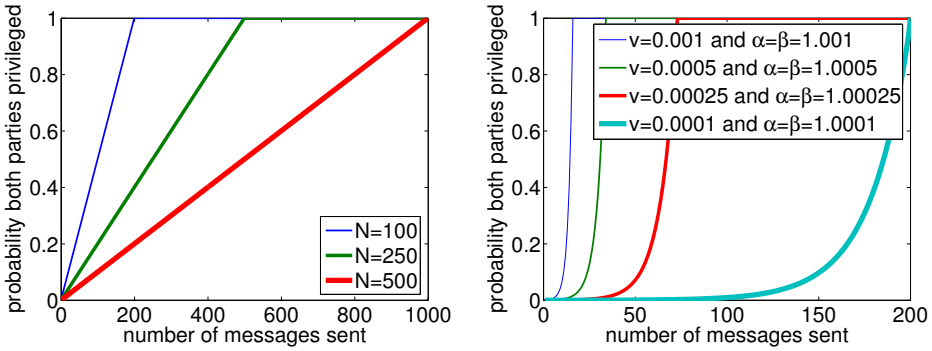
```
// parties can continue sending messages after the coin has been flipped
[] turn=0 → (lambdaA′=min(k, lambdaA + 1)) ∧ (turn′=1);
[] turn=1 → (lambdaB′=min(k, lambdaB + 1)) ∧ (turn′=0);
```

6 Analysis Results for Rabin's, BGMR, and TBGMR protocols

First, we analyse fairness as a function of the number of rounds for Rabin's and BGMR protocols. We then use PRISM to quantify the tradeoff between fairness and timeliness in TBGMR, the timely version of the BGMR protocol.

6.1 Fairness in Rabin's and BGMR protocols

A probabilistic contract signing protocol terminates in an unfair state if one party is committed (in Rabin's protocol) or privileged (in BGMR and TBGMR protocols), while



(a) Rabin's protocol

(b) BGMR protocol

Figure 4: Fairness and communication complexity of the protocols.

the other is not.

Fairness in Rabin's protocol. In Rabin's protocol, unfair states are defined by the condition $(i{>}0) \wedge (mA{\geq}i) \wedge (mB{<}i)$, *i.e.*, the number i randomly chosen by the third party is such that B possesses A's commitment, but A does not possess B's commitment.

We have computed the maximum probability of reaching such a state for a number of different values of N (between 10 and 1000) and, as expected, in each case the maximum probability equals $\frac{1}{N}$. The probability of reaching a state in which both parties are committed as a function of the number of message rounds is plotted in Figure 4(a) for different values of N. This figure illustrates that probabilistic fairness in Rabin's protocol depends linearly on the number of rounds.

Fairness in BGMR protocol. We used PRISM to calculate the probability of reaching a state in which both parties are privileged as a function of the number of message rounds. The results for different parameters are in Figure 4(b).

6.2 Unfairness in TBGMR protocol

Recall that if the coin has been flipped and the last messages sent by parties A and B include the probability values p^a and p^b respectively, then in the current state of the protocol B is privileged and A is not if and only if the value of the coin is in the interval $(p_b, p_a]$. Let's call this set of states **unfair**. We used PRISM to calculate the *maximum probability* of reaching a state in **unfair** from the initial state (where no messages have been sent and the coin has not been flipped). Table 1 gives the summary of the results obtained using PRISM when considering a number of different parameters for both the `BGMR` and `TBGMR` versions of the protocol.

The probability of reaching a state in **unfair** is maximised, over all nondeterministic choices that can be made by a misbehaving B, by the following strategy. As soon as B receives the first message from A, he invokes the judge, pretending that A has stopped communicating and presenting A's message to the judge. In `TBGMR`, this will cause the judge to flip the coin and announce a verdict immediately. B delays the judge's announcement message to A, and keeps exchanging messages with A in the main flow of the protocol. After each message from A, B invokes the judge and presents A's latest message, until one of them results in a positive verdict. Depending on his choice of β, B can steer the protocol to a state unfair to A with arbitrarily high probability.

Table 1: Model checking results for the BMGR and TBMGR protocols.

v	α	β	N	number of states	**max probability of reaching a state where only B privileged** BGMR	TBGMR
0.1	1.1	1.05	10	408	**0.1000**	**0.8000**
			100	3,738	**0.1000**	**0.7000**
			1,000	37,038	**0.1000**	**0.7080**
0.1	1.1	1.01	10	518	**0.1000**	**0.9000**
			100	4,748	**0.1000**	**0.9000**
			1,000	47,048	**0.1000**	**0.9180**
0.01	1.01	1.005	10	6,832	**0.1000**	**0.6000**
			100	62,722	**0.0100**	**0.6600**
			1,000	621,622	**0.0100**	**0.6580**
0.01	1.01	1.001	10	9,296	**0.1000**	**1.0000**
			100	85,346	**0.0100**	**0.9400**
			1,000	845,846	**0.0100**	**0.9070**
0.001	1.001	1.0005	10	101,410	**0.1000**	**0.6000**
			100	931,120	**0.0100**	**0.7200**
			1,000	9,228,220	**0.0010**	**0.6840**
0.001	1.001	1.0001	10	138,260	**0.1000**	**0.9000**
			100	1,269,470	**0.0100**	**0.8700**
			1,000	12,581,570	**0.0010**	**0.9010**

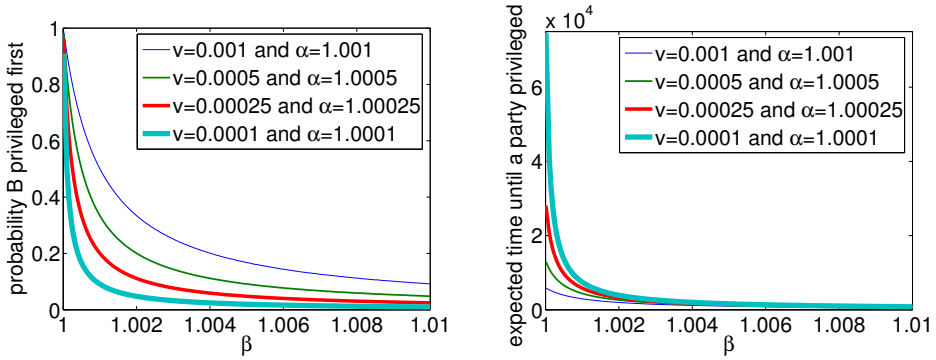


Figure 5: Probability and expected time of B's win in TBGMR protocol.

Table 1 lists the maximum probabilities that can be achieved by this strategy in the TBGMR protocol. Recall that the results obtained with PRISM are for a simplified model, where the coin is discretised. Nevertheless, due to the simplicity of this strategy, we were able to write a simple MATLAB script which calculated the probability of reaching a state which is unfair to A under this strategy for a general coin, that is, a coin whose flips take a value uniformly chosen from the $[0, 1]$ interval.

Figure 5 (left chart) shows the probability, for various choices of parameters, of reaching a state in which B is privileged and A is not. Note that B can bring this probability arbitrarily close to 1 by choosing a small β. We assume that the value of β is constant and chosen by B beforehand. In practice, B can decrease his β adaptively. A may respond in kind, of course, but then the protocol will crawl to a halt even if B is not cheating.

This attack on TBGMR is feasible because the judge remembers (or reconstructs, using a pseudo-random function with a secret input—see Section 4.2) only the value of his coin flip, ρ_C, and *not* his previous verdicts. Therefore, B induces the coin flip

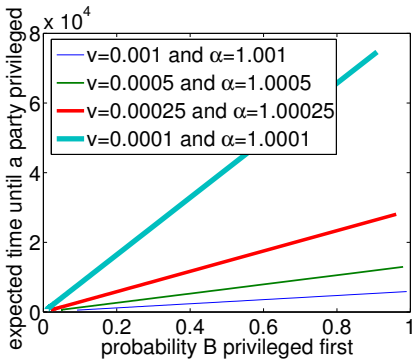


Figure 6: Expected time to B's win vs. probability of B's win.

as soon as possible, and then gets the judge to produce verdict after verdict until the probability value contained in A's message is high enough to result in a positive verdict.

Accountability of the judge. Under the assumption that the channels are resilient (*i.e.*, eventual delivery is guaranteed), at some point A will receive all of the judge's verdicts that have been delayed by B. Since these verdicts are contradictory (some declare the contract binding, and some do not), A may be able to argue that somebody—either B, or the judge, or both—misbehaved in the protocol. Note, however, that the protocol specification does *not* require the judge to produce consistent verdicts, only consistent values of ρ_C.

To repair this, the protocol may be augmented with "rules of evidence" specifying what constitutes a violation (see also the discussion in Section 7). Such rules are not mentioned in [6] and appear to be non-trivial. A discussion of what they might look like is beyond the scope of this paper.

6.3 Attacker's tradeoff

Although a Dolev-Yao attacker is assumed to be capable of delaying messages on public communication channels, in practice this might be difficult, especially if long delays are needed to achieve the attacker's goal. Therefore, it is important to quantify the relationship between B's probability of winning (bringing the protocol into a state that's unfair to A), and how long B needs to delay the judge's messages to A. As a rough measure of time, we take the expected number of message exchanges between A and B. The greater the number of messages A has to send before B reaches a privileged state, the longer the delay.

We wrote a MATLAB script to calculate the expected number of messages sent before a party becomes privileged. The results are shown in Figure 5 (right chart) for various values of v and α as β varies. As expected, the lower β, the greater the number of messages that the parties have to exchange before B becomes privileged. Recall, however, that lower values of β result in higher probability that B *eventually* wins (left chart on Figure 5).

The misbehaving participant thus faces a tradeoff. The higher the probability of winning, the longer the judge's messages to the honest participant must be delayed. This tradeoff is quantified by the chart in Figure 6. As the chart demonstrates, there is a linear tradeoff for a misbehaving B between the expected time to win and the probability of winning. If B is not confident in his ability to delay the judge's messages to A and/or exchange messages with A fast enough (before the judge's

verdict reaches A), B can choose a large value of β and settle for a shorter delay and thus lower probability of steering the protocol into an unfair state.

7 Making BGMR Protocol Timely

As demonstrated in Section 6, the `BGMR` protocol cannot be made timely by simply having the judge flip his coin immediately after he is invoked, because then fairness is lost. In this section, we discuss modifications to the protocol that may enable it to provide both timeliness and fairness.

Fast and secure communication channels. The attack described in Section 6 can be prevented if the protocol requires the communication channel between the judge and the honest participant to be of very high quality. More precisely, both fairness and timeliness will be guaranteed if B is prevented from delaying the judge's messages to A. Also, the judge's channel to A must be faster than the channel between A and B. This ensures that A will be notified of the judge's verdict immediately after the judge flips his coin and will stop communicating with B.

While a high-quality channel *from* A to the judge might be feasible, it is significantly more difficult to ensure that the channel from the judge *to* A is faster than the channel between A and B. The judge is necessarily part of the infrastructure in which the protocol operates, servicing multiple instances of the protocol at the same time. Therefore, it is reasonable to presume that the judge is always available and responsive. On the other hand, A is simply one of many participants using the protocol, may not be easy to locate on a short notice, may not be expecting a message from the judge before the cutoff date D, *etc.* On a public network, it is usually much easier for a user to contact the authorities than for the authorities to locate and contact the user.

Judge with unbounded memory. The attack also succeeds because the judge does not remember his past verdicts, only the value of his coin flip ρ_C. If the judge is made to remember his first verdict, and simply repeat it in response to all subsequent invocations regarding the same contract, the attack will be prevented. Note, however, that such a judge will require unbounded memory, because, unlike ρ_C, the value of the first verdict cannot be reconstructed from subsequent evidence using a pseudo-random function. Therefore, the judge will need to store the verdict he made for every instance of the protocol he has ever been asked to resolve in case he is invoked again regarding the same instance.

A possible optimization of this approach is to expunge all verdicts regarding a particular contract from the judge's memory once the cutoff date for that contract has passed. This introduces additional complications, however, since participants are permitted to ask the judge for a verdict even after the cutoff date. If the verdict in this case is constructed from ρ_C and the newly presented evidence, it may be inconsistent with the verdicts the judge previously announced regarding the same contract. To avoid this, the old verdicts must be stored forever and the judge's memory must be infinite, regardless of the quality of the communication channels between the parties and the judge.

Signed messages to the judge. If all invocations of the judge are signed by the requesting party, and that signature is included in the judge's verdict, then A will be able to prove B's misbehaviour (that B invoked the judge more than once) when all of the judge's verdicts finally reach him after having been delayed by B. If invoking the judge more than once is construed as a violation, however, the protocol might be problematic for an honest B in case there is a genuine delay on the

channel between A and B. Such a delay may cause an honest B to time out, invoke the judge, and then, after A's message with a new probability value finally arrives, invoke the judge again, which would be interpreted as an attempt to cheat.

Combining fairness with timeliness seems inherently difficult in the case of probabilistic contract signing. The approaches outlined above demand either extremely stringent requirements on the communication channels, which would limit applicability of the protocol, or introduction of the additional "rules of evidence" that define what is and what is not a violation. Such rules are not included in the protocol specification [6] and would have to be designed from scratch. Designing an appropriate set of rules appears difficult, as they should accommodate legitimate scenarios (a participant should be permitted to request a verdict from the judge more than once, because the judge's messages might have been lost or corrupted in transmission), while preventing a malicious party from repeatedly contacting the judge until the desired verdict is obtained.

8 EGL protocol

The probabilistic contract signing protocol of Even, Goldreich, and Lempel [16] proceeds as follows. Initiator A generates $2n$ random secret $a_1, \ldots, a_{2n}$ and reveals $F_{a_1}(c), \ldots, F_{a_{2n}}(c)$, where c is the text of the contract and $F_{a_i}(c)$ can be thought of, roughly, as encryption of c under secret a_i. The assumption is that F is a *uniformly secure* cryptosystem (see below). Similarly, B generates $2n$ random secrets $b_1, \ldots, b_{2n}$, and reveals $F_{b_1}(c), \ldots, F_{b_{2n}}(c)$. Intuitively, both A's and B's sequences of secrets can be thought of as sequences of pairs $\{(a_i, a_{n+i})\}$ and $\{(b_i, b_{n+i})\}$, respectively.

A is considered committed to contract c if B can present both secrets from any one of A's pairs, that is, if B knows both a_i and a_{n+i} for some i. B is considered committed if A can present b_j and b_{n+j} for some j. The secrets can be verified by using them to decrypt the publicly announced values $F_{a_i}(c), F_{a_{n+i}}(c), F_{b_j}(c), F_{b_{n+j}}(c)$ (they should decrypt to c).

The protocol itself consists in *partial secret exchange* as presented in Figure 7 (the *oblivious transfer* primitive OT is explained below). At the end of the exchange, if both parties behave correctly, B learns all of A's secrets, and vice versa.

Assumptions. The `EGL` protocol assumes the existence of a uniformly secure cryptosystem F. Informally, given c, $F_k(c)$ and first i bits of k, the expected time of computing k is independent of the i known bits. In particular, there exists an algorithm $\mathcal{A}$ which, given a pair $c, F_k(c)$, finds k in expected time $t_{\mathcal{A}}(s)$ where s is a bit string consistent with k (s represents partial knowledge of the key k), and $t_{\mathcal{A}}(s)$ depends only on the number of unknown bits in k, *i.e.*, $t_{\mathcal{A}}(s) = T_{\mathcal{A}}(|k|-i)$ for each bit string s of length i consistent with k. Moreover, there is no algorithm that achieves this in less than half the time. (A precise definition can be found in [16, section 2].)

The `EGL` protocol also assumes the existence of a *1-out-of-2 oblivious transfer protocol* (OT_2^1). The protocol $\mathrm{OT}_2^1(S, R, x, y)$ can be defined axiomatically as a protocol in which, if S follows the protocol, R receives x with probability $\frac{1}{2}$, or else receives y. S does not learn whether R received x or y (*i.e.*, the a-posteriori probability for S remains equal to $\frac{1}{2}$). If S tries to deviate from the protocol to increase the a-posteriori probability, R can detect this with probability at least $\frac{1}{2}$. We omit the discussion of how such a primitive can be implemented, and assume for the purposes of our analysis that a correct implementation is available.

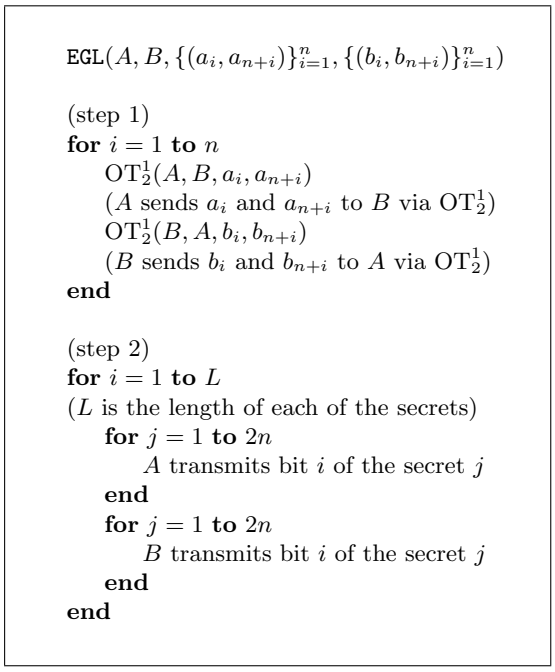


Figure 7: Even, Goldreich and Lempel partial secret exchange protocol (EGL protocol).

9 Model for EGL protocol

In our model for the EGL protocol, the state of each party is represented by its current knowledge of the secrets of the other party. More precisely, for party A (B), it is the number of bits of each of the secrets $b_1, \ldots, b_{2n}$ ($a_1, \ldots, a_{2n}$) that it have been sent by B (A) in either step 1, or step 2 of the protocol. Therefore, in our PRISM specification we encode the state of the party A with the variables $nb1, \ldots, nb(2n)$ (each taking values in the range $\{0, \ldots, L\}$) where nbi represents the current number of bits of the secret b_i that A possesses. These variables are updated when A receives a message from B. The update depends on the type of transmission B performs. For example:

- In step 1 of the protocol, when B sends the secrets b_i and b_{n+i} to A via OT_2^1, with probability $\frac{1}{2}$ either the variable nbi or the variable $nb(n+i)$ is updated to L (*i.e.*, after receiving the message, with probability $\frac{1}{2}$, A knows either all the bits of the secret b_i, or of b_{n+i}).

- In step 2 of the protocol, if B sends a bit of the secret b_i to A, then the variable nbi is incremented by 1, unless A already knows all the bits of b_i (nbi=L). In the latter case, A does not gain any additional information, thus nbi remains unchanged.

To specify this protocol in PRISM, we define a module for each party (which updates the above variables) and an additional module that is used to keep track of what messages have been sent and the order in which these transmissions take place, that is, how far the parties have actually progressed through the protocol. Following this approach, the specification of the EGL protocol used as the input to PRISM in the case when n=4 and L=2 is given in Figure 8.

```
const int n = 4; // number of pairs of secrets
const int L = 2; // number of bits of each secret

module protocol

   b : [1..L]; // counter for current bit to send
   n : [1..n]; // counter for which secret to send
   party : [1..2]; // which party sends next (1 – A and 2 – B)
   phase : [1..4]; // phase of the protocol
   // 1 - sending secrets using OT^1_2 (step 1 of the protocol)
   // 2 - send bits of the secrets 1,...,n (step 2 of the protocol)
   // 3 - send bits of the secrets n+1,...,2n (step 2 of the protocol)

   // STEP 1
   [recB] phase=1 ∧ party=1 → (party′=2); // A sends
   [recA] phase=1 ∧ party=2 ∧ n<n → (party′=1) ∧ (n′=n+1); // B sends (move onto next secret)
   [recA] phase=1 ∧ party=2 ∧ n=n → (party′=1) ∧ (phase′=2) ∧ (n′=1); // B sends (finished step)
   // STEP 2
   // when A sends
   [recB] phase=2..3 ∧ party=1 ∧ n<n → (n′=n+1); // next secret
   [recB] phase=2 ∧ party=1 ∧ n=n → (phase′=3) ∧ (n′=1); // move onto secrets n+1,...,2n
   [recB] phase=3 ∧ party=1 ∧ n=n → (phase′=2) ∧ (party′=2) ∧ (n′=1); // move onto party B
   // when B sends
   [recA] phase=2..3 ∧ party=2 ∧ n<n → (n′=n+1); // next secret
   [recA] phase=2 ∧ party=2 ∧ n=n → (phase′=3) ∧ (n′=1); // move onto secrets n+1,...,2n
   [recA] phase=3 ∧ party=2 ∧ n=n ∧ b<L → (phase′=2) ∧ (party′=1) ∧ (n′=1) ∧ (b′=b+1); // next bit
   [recA] phase=3 ∧ party=2 ∧ n=n ∧ b=L → (phase′=4); // finished protocol
   // FINISHED
   [] phase=4 → (phase′=4); // loop

endmodule

module partyA

   // nbj the number of bits of the secret b_i that A possesses
   nb1 : [0..L];
   nb2 : [0..L];
   nb3 : [0..L];
   nb4 : [0..L];
   nb5 : [0..L];
   nb6 : [0..L];
   nb7 : [0..L];
   nb8 : [0..L];

   // first step (get either secret i or n+i with equal probability)
   [recA] phase=1 ∧ n=1 → 0.5 : (nb1′=L) + 0.5 : (nb5′=L);
   [recA] phase=1 ∧ n=2 → 0.5 : (nb2′=L) + 0.5 : (nb6′=L);
   [recA] phase=1 ∧ n=3 → 0.5 : (nb3′=L) + 0.5 : (nb7′=L);
   [recA] phase=1 ∧ n=4 → 0.5 : (nb4′=L) + 0.5 : (nb8′=L);
   // second step (secrets 1,...,n)
   [recA] phase=2 ∧ n=1 → (nb1′=min(nb1+1, L));
   [recA] phase=2 ∧ n=2 → (nb2′=min(nb2+1, L));
   [recA] phase=2 ∧ n=3 → (nb3′=min(nb3+1, L));
   [recA] phase=2 ∧ n=4 → (nb4′=min(nb4+1, L));
   // second step (secrets n+1,...,2n)
   [recA] phase=3 ∧ n=1 → (nb5′=min(nb5+1, L));
   [recA] phase=3 ∧ n=2 → (nb6′=min(nb6+1, L));
   [recA] phase=3 ∧ n=3 → (nb7′=min(nb7+1, L));
   [recA] phase=3 ∧ n=4 → (nb8′=min(nb8+1, L));

endmodule

// construct module for party B through renaming
module
     partyB = partyA[b1=a1, b2=a2, b3=a3, b4=a4, b5=a5, b6=a6, b7=a7, b8=a8, recA=recB]
endmodule
```

Figure 8: PRISM code for EGL protocol ($n = 4$ and $L = 2$).

10 Fairness in EGL protocol

When investigating the EGL protocol, we discovered what might be considered to be a weakness in the protocol when party B is allowed to act maliciously by quitting the protocol early (note that this type of behaviour was not considered in [16]). More precisely, our PRISM analysis shows that, with probability 1, the protocol reaches a state where B possesses both secrets from at least one of A's pairs, while A does not have complete knowledge of any pair of B's secrets. Intuitively, this happens because in the last round of step 2 of the protocol, A sends the last bit for all of his secrets before B sends the last bit for even a single secret.

This means that if B stops participating in the protocol (*i.e.*, stops sending messages to A) as soon as he has obtained at least one of A's pairs, then B enjoys a distinct advantage over A: he knows a pair of A's secrets, while A does not know both secrets in any of B's pairs. This advantage is limited by the fact that in the state where B obtains the last bit he needs to complete a pair of A's secrets, A possesses, for any pair of B's secrets, one complete secret and $L-1$ bits of the other secret. Since $|k| = L$, this means that A can compute the bit that he requires to obtain full knowledge of a pair of B's secrets in expected time $T_{\mathcal{A}}(1)$ (see Section 8). This computation will take some time, however. Supposing that A is behaving honestly, A will first wait for B to send a message before realizing that B has quit the protocol and trying to compute B's secret on his own.

The precise definition of fairness in [16] says that, provided B does not deviate from the protocol, in any state where B can compute one of A's pairs in expected time T, A can compute one of B's pairs in expected time $16T$. If B does deviate in order to reach a situation in which B can compute one of A's pairs in expected time T, but A cannot compute at least one of B's pairs in expected time $4T$, then A can detect this with probability of at least $1 - 2^{-n}$. This property cannot hold if $T = 0$, as demonstrated by our analysis (with asynchronous communication, some party has to be the first to completely learn one of the other's pairs). What is more worrying is that the "losing" party is *always* the initiator A.

In Section 11, we investigate possible ways of making this protocol more "fair" by changing the order of bit revelation messages exchanged in step 2 of the protocol (see Figure 7), while relying on the properties of the OT_2^1 protocol.

An alternative, more general formulation of (un)fairness could be based on the number of bits that each party still has to compute before he, too, knows both secrets in at least one of his opponent's pairs (this is different from the definition in [16], which uses expected time). We say that party A (B) has *l-knowledge* ($0 \leq l \leq L$) if there exists a pair of secrets (b_i, b_{n+i}) ((a_i, a_{n+i})) such that A (B) fully knows one of the secrets and at least l bits of the other secret.

Our analysis shows that, with probability 1, for any $1 \leq l \leq L$ the protocol reaches a state where B has l-knowledge, while A does not. In particular, considering the case when $l = L$ (*i.e.*, both secrets are completely known), we obtain that, with probability 1, the protocol reaches a state where B knows one of A's pairs while A does know any of B's pairs. All of our subsequent analyses of EGL variants can be generalized in the same way, by replacing "knowledge of a pair of secrets" with "l-knowledge" for any $1 \leq l \leq L$.

11 Making EGL Protocol Fair

Recall that the weakness we have discovered in the EGL protocol is due to the fact that in step 2 (see Figure 7), A reveals the last bit for each of his secrets before B reveals any of the last bits of B's secrets. Therefore, B always gains full knowledge of all of A's secrets before A learns even a single pair of B's secrets. In this section,

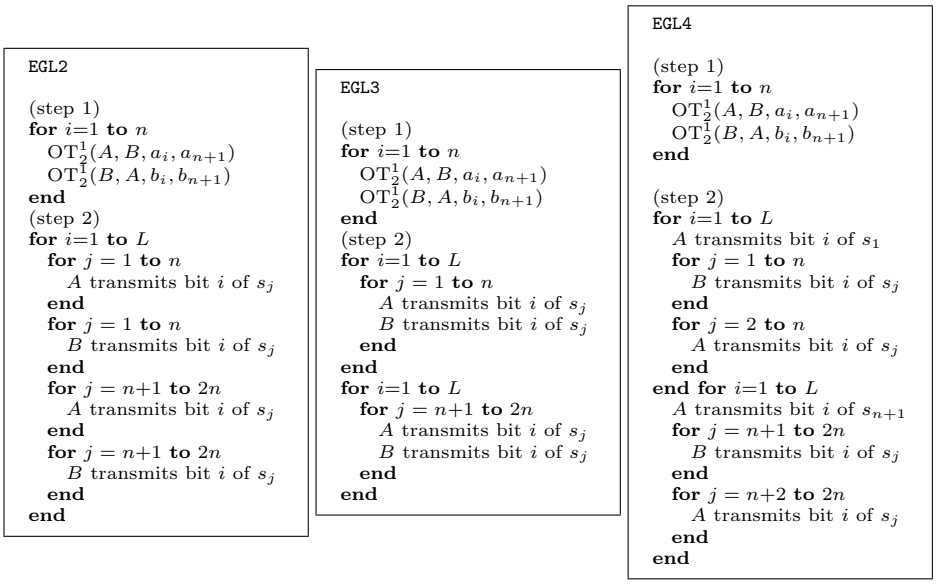


Figure 9: Alternatives to EGL protocol for input $(A, B, \{(a_i, a_{n+i})\}, \{(b_i, b_{n+i})\})$ which use different message scheduling schemes in step 2 (s_j is used to denote j).

we will investigate whether the protocol can be made more fair by considering alternative message scheduling schemes for step 2 of the protocol.

Several alternative protocols to EGL are shown in Figure 9. Note that step 1 of each protocol is identical to that of the original EGL protocol. The idea behind these alternatives is that they attempt, in varying degrees, to overcome A's "first-mover" disadvantage by interleaving bit-sending by A and B without becoming unfair towards B.

In the original EGL protocol, the message schedule in step 2 involves A sending the jth bit for all of his secrets before B does the same, and then moving to the next bit.

In the EGL2 protocol, B sends a bit for each of the secrets $1, \ldots, n$ after A has done this, and before either party has sent the bits for the secrets n+1, $\ldots, 2n$.

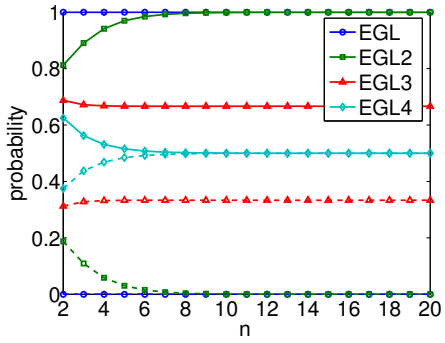
The other two schemes we consider (*i.e.*, EGL3 and EGL4) separate the sending of the bits for the secrets $1, \ldots, n$ from those for the secrets n+1, $\ldots, 2n$. More precisely, in these schemes both parties send all the bits for the secrets $1, \ldots, n$ before they start sending the bits for the secrets n+1, $\ldots, 2n$.

In the EGL3 protocol, A and B interleave the exchange of secrets in both phases bit by bit. In the EGL4 protocol, in the second step, A sends a bit for his first secret and then waits for B to send a bit for each of his secrets $1, \ldots, n$, before moving to A's secrets $2, \ldots, n$.

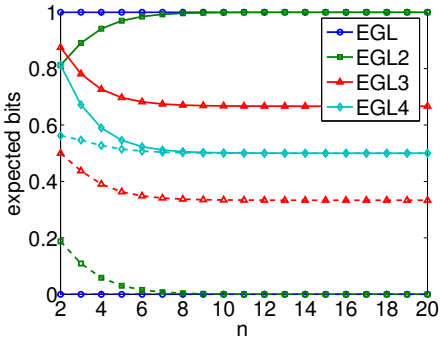
We have evaluated the fairness these protocols by calculating with PRISM the following fairness measures as the number of secrets n varies. Note that, under the assumption that all the secrets have the same number of bits, these results are independent of the number of bits L in each secret.

- *The probability of reaching a state where A (B) does not know a pair while B (A) does know a pair* (Figure 10(a)). This measure can be interpreted as the "chance" that the protocol is unfair towards either party.

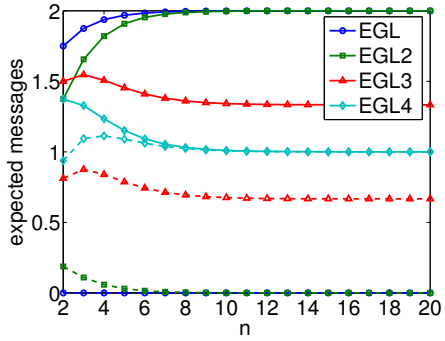
- *The expected number of bits that A needs to know a pair once B (A) knows a pair* (Figure 10(b)). This property is a quantification of how unfair the protocol is with respect to either party.



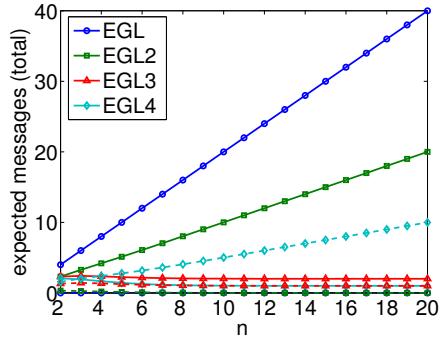
(a) Probability reach a state where the party does not knows a pair while the other party knows a pair



(b) Expected bits the party requires to knows a pair once the other party knows a pair



(c) Expected messages from the other party the party requires to knows a pair once the other party knows a pair



(d) Expected total messages before the party knows a pair once the other party knows a pair

Figure 10: Performance results for the protocols `EGL`, `EGL2`, `EGL3` and `EGL4` (results for party A solid line and results for party B dashed line).

- *The expected number of messages from B (A) that A (B) needs to knows a pair once B (A) knows a pair* (Figure 10(c)). This measure can be interpreted as an indication of how much influence a corrupted party has on the fairness of the protocol, since a corrupted party can try and delay these messages in order to gain an advantage.
- *The expected number of messages that need to be sent (by either party) before A (B) knows a pair once B (A) knows a pair* (Figure 10(d)). This measure can be interpreted as representing the "duration" of unfairness, *i.e.*, the time that one of the parties has an advantage.

We first consider the results in Figure 10(a), that is, the probability of reaching a state where one party does not know a pair of the opponent's secrets while the opponent does. As described in Section 10, this probability is 1 for party A in the original protocol `EGL`. In `EGL2`, as n increases, the probability of the protocol being unfair towards A converges towards 1, which is due to the following two properties of the protocol:

1. In `EGL2`, A sends a bit for each of the secrets $1, \ldots, n$ before B does;
2. As n increases, at the end of step 1 (which is the same for all the variants

we consider), the probability of a party receiving at least one of the secrets $n{+}1, \ldots, 2n$ converges to 1.

On the other hand, for `EGL3` and `EGL4`, increasing n does not increase the probability of the protocol being unfair towards A. For `EGL3` this is due to the fact that bit-by-bit revelation of the secrets is interleaved between A and B. However, because A always goes first in this protocol, the probability of `EGL3` being unfair towards A is always larger than for B. As n increases, these probabilities converge to $\frac{2}{3}$ and $\frac{1}{3}$, respectively. In the case of the protocol `EGL4`, it follows from point 2 above that, if B does not obtain the secret $n{+}1$ in the first step of the protocol, then the probability that A will know a pair of B's secrets before B knows a pair of A's secrets converges to 1 as n increases. Since B learns the secret $n{+}1$ in step 1 with probability $\frac{1}{2}$, as n increases, the probability that either party knows a pair of secrets before the opponent converges to $\frac{1}{2}$. This is the best that can be hoped for in an asynchronous protocol, since one party will always know a pair of secrets before the other. All we can do is to make the selection of the "winner" as random as possible.

One can understand the results in Figures 10(b) and 10(c) using the same reasoning as that presented for Figure 10(a) and these again show:

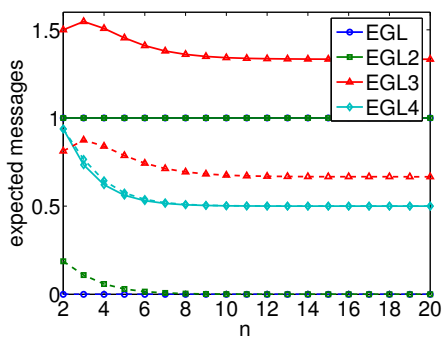
- both `EGL` and `EGL2` lead to a very "unfair" protocol;
- `EGL2` becomes more "unfair" towards A as n increases;
- `EGL4` is the "fairest" protocol.

Finally, consider Figure 10(d), that is, the expected number of messages that need to be sent (by either party) before a party knows a pair given the opponent already knows a pair. In contrast to other fairness measures we considered, `EGL4` is no longer "fair" to both parties under this measure. Once A knows a pair, the expected number of messages that need to be sent (by either party) before B knows a pair increases as n increases, while, once B knows a pair, the expected number of messages that need to be sent (by either party) before A knows a pair actually *decreases* as n increases.

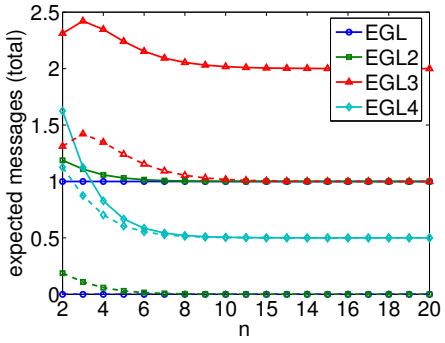
This result is due to the fact that when B does *not* get the secret $n{+}1$ in step 1 of the protocol (which happens with probability $\frac{1}{2}$), then he must send the last bit for all of his secrets $1, \ldots, n$ to A before he fully learns any of A's pairs. Therefore, increasing n increases the expected number of messages. On the other hand, as n increases, the probability of A knowing a pair before B is done sending these messages converges to 1. Using the message schedule from the `EGL3` variant solves this non-uniformity by symmetrically interleaving the sending of bits. However, we demonstrated above that `EGL3` is not as fair as `EGL4` with respect to the other fairness measures considered.

Considering the results in Figure 10(d) for `EGL` and `EGL2`, we see that the expected number of messages A must receive is greater than $2n$ and n, respectively. The reason is that in the message schedule used in step 2 of `EGL` (`EGL2`), A must send the last bits of the secrets $1, \ldots, 2n$ $(1, \ldots, n)$ before B sends the last bits of any of his secrets.

Improving EGL4 under "duration of unfairness" measure. Summarizing the results presented in Figure 10, we observe that the protocol `EGL4` appears to be the "fairest" protocol, except for the following: if the protocol ends up being unfair to B during a particular execution, this unfairness has the longest expected duration. Having said this, most of this is the time spent by B sending messages (compare the results for Figure 10(c) and Figure 10(d)). Therefore, B can reduce this bias by sending his messages quickly.



(a) Expected messages from the other party the party requires to knows a pair once the other party knows a pair



(b) Expected total messages before the party knows a pair once the other party knows a pair

Figure 11: Performance results for the protocols `EGL`, `EGL2`, `EGL3` and `EGL4` after combining sequential transmissions (results for party A solid line and results for party B dashed line).

An easy way to rectify this problem is to require that, whenever a party has to send a sequence of bits in a row (without the other party sending bits in between), these bits must be sent in a single message. For example, for `EGL2` this would mean that A has to transmit the first bits of all the secrets $1, \ldots, n$ to B in a single message at the start of step 2.

Figure 11 presents the effect this change would have on the performance of each of the variants we have considered. We only show the measures related to the expected number of messages as the other measures are not affected by this change. Note that this has no effect on the protocol `EGL2` because it is fully interleaved: after a party has sent a single bit, the next transmission is from the other party. As the results indicate, this does indeed solve the fairness issues we found with the protocol `EGL4`. The limitation of this approach is that the size of messages increases as the number of secrets increases. Party B (A) is required to send a message containing n ($n-1$) bits.

12 Conclusions

We presented several case studies, demonstrating how formal analysis techniques can be used to verify properties of probabilistic security protocols. For a timely variant of the probabilistic contract signing protocol of Ben-Or, Goldreich, Micali, and Rivest we used PRISM, a probabilistic model checker, to show that there exists an attack strategy for a misbehaving participant that brings the protocol into an unfair state with arbitrarily high probability. We also quantified the attacker's tradeoff between the probability of winning and the need to delay messages from the judge to the honest participant.

For the probabilistic contract signing protocol of Even, Goldreich, and Lempel, we demonstrated that an "unfair" state in which the responder knows the initiator's commitment, but the initiator does not know the responder's commitment is reached with probability 1. We then used PRISM to demonstrate better fairness guarantees provided by alternative message scheduling schemes. Designing a probabilistic contract signing protocols with desirable properties such as fairness, timeliness, and minimal computational disadvantage for both parties is a challeng-

ing task. Our case studies, in addition to demonstrating feasibility of probabilistic model checking as an analysis technique for security protocols, are a step in this direction.

While probabilistic model checking is a useful tool for analyzing protocols involving *discrete* probabilities (such as those studied in this paper), applying it to conventional cryptographic protocols requires the ability to reason about *asymptotically negligible* probabilities. Investigating applications of probabilistic model checking to reduction-based proofs used in modern cryptography is an interesting direction of future research.

Acknowledgements

This work was supported in part by FORWARD, EPSRC grants GR/N22960 and GR/S46727, ONR grants N00014-01-1-0837 and N00014-03-1-0961, and DARPA contract N66001-00-C-8015.

References

[1] A. Aldini and R. Gorrieri. Security analysis of a probabilistic non-repudiation protocol. In H. Hermanns and R. Segala, editors, *Proc. PAPM/PROBMIV'02*, volume 2399 of *LNCS*, pages 17–36. Springer-Verlag, 2002.

[2] R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999.

[3] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 7–17, 1997.

[4] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Selected Areas in Communications*, 18(4):593–610, 2000.

[5] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

[6] M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[7] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. Foundations of Software Technology and Theoretical Computer Science (FST/TCS)*, volume 1026 of *LNCS*, pages 499–513. Springer-Verlag, 1995.

[8] D. Boneh and M. Naor. Timed commitments. In C. Linnhoff-Popien and H. Hegering, editors, *Proc. CRYPTO'00*, volume 1880 of *LNCS*, pages 236–254. Springer-Verlag, 2000.

[9] L. Buttyán and J.-P. Hubaux. Toward a formal model of fair exchange — a game theoretic approach. Technical Report SSC/1999/39, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, 1999.

[10] L. Buttyán, J.-P. Hubaux, and S. Čapkun. A formal analysis of Syverson's rational exchange protocol. In *Proc. 15th IEEE Computer Security Foundations Workshop*, pages 193–205, 2002.

[11] R. Chadha, M. Kanovich, and A. Scedrov. Inductive methods and contract-signing protocols. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 176–185, 2001.

[12] I. Damgård. Practical and provably secure release of a secret and exchange of signatures. *J. Cryptology*, 8(4):201–222, 1995.

[13] C. Derman. *Finite-State Markovian Decision Processes*. New York: Academic Press, 1970.

[14] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[15] S. Even. A protocol for signing contracts. Technical Report 231, Computer Science Dept., Technion, Israel, 1982.

[16] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[17] S. Even and Y. Yacobi. Relations among public key signature schemes. Technical Report 175, Computer Science Dept., Technion, Israel, 1980.

[18] J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In M. Wiener, editor, *Proc. CRYPTO'99*, volume 1666 of *LNCS*, pages 449–466. Springer-Verlag, 1999.

[19] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[20] J. Gray. Toward a mathematical foundation for information flow security. *J. Computer Security*, 1(3):255–294, 1992.

[21] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[22] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer-Verlag, 2006.

[23] S. Kremer and J.-F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In K. Larsen and M. Nielsen, editors, *Proc. CONCUR'01*, volume 2154 of *LNCS*, pages 551–565. Springer-Verlag, 2001.

[24] S. Kremer and J.-F. Raskin. Game analysis of abuse-free contract signing. In *Proc. 15th IEEE Computer Security Foundations Workshop*, pages 206–220, 2002.

[25] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security analysis. In J. Wing, J. Woodcock, and J. Davies, editors, *Proc. World Congress on Formal Methods*, volume 1708 of *LNCS*, pages 776–793. Springer-Verlag, 1999.

[26] O. Markowitch and Y. Roggeman. Probabilistic non-repudiation without trusted third party. In G. Persiano, editor, *Proc. 2nd Conference on Security in Communication Networks (SCN 99)*, 1999.

[27] S. Micali. Certified e-mail with invisible post offices. Presented at RSA Security Conference, 1997.

[28] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. In A. Abdallah, P. Ryan, and S. Schneider, editors, *Proc. 1st International Conference on Formal Aspects of Security (FASec '02)*, volume 2629 of *LNCS*, pages 119–128. Springer-Verlag, 2003.

[29] PRISM web page. http://www.cs.bham.ac.uk/~dxp/prism/.

[30] M. Rabin. Transaction protection by beacons. *J. Computer and System Sciences*, 27(2):256–267, 1983.

[31] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (eds.), volume 23 of *CRM Monograph Series*. American Mathematical Society, 2004.

[32] V. Shmatikov and J. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, 2002.

[33] P. Syverson and J. Gray. The epistemic representation of information flow security in probabilistic systems. In *Proc. 8th IEEE Computer Security Foundations Workshop*, pages 152–166, 1995.

[34] D. Volpano and G. Smith. Probabilistic non-interference in a concurrent language. In *Proc. 11th IEEE Computer Security Foundations Workshop*, pages 34–43, 1998.