Modelchecking safety properties in randomized security protocols

Matthew S. Bauer[1], Rohit Chadha[2], and Mahesh Viswanathan[3]

[1] Galois Inc.
[2] University of Missouri, Columbia
[3] University of Illinois, Urbana-Champaign

Abstract. Automated reasoning tools for security protocols model protocols as non-deterministic processes that communicate through a Dolev-Yao attacker. There are, however, a large class of protocols whose correctness relies on an explicit ability to model and reason about randomness. Although such protocols lie at the heart of many widely adopted systems for anonymous communication, they have so-far eluded automated verification techniques. We propose an algorithm for reasoning about safety properties for randomized protocols. The algorithm is implemented as an extension of Stochastic Protocol ANalyzer (SPAN), the mechanized tool that reasons about the indistinguishability properties of randomized protocols. Using SPAN, we conduct the first automated verification on several randomized security protocols and uncover previously unknown design weaknesses in several of the protocols we analyzed.

1 Introduction

As security protocols are vulnerable to design flaws, machine-aided formal analysis is often utilized to verify their security guarantees. Such analysis must be carried out in the presence of an attacker that can read, intercept, modify and replay all messages on public channels, and potentially send its messages. The presence of the attacker makes the analysis challenging. In order to aid automation, the analysis is often carried out in the so-called *Dolev-Yao* model where messages are modeled as terms in a first-order vocabulary, the assumption of perfect cryptography is made. In the Dolev-Yao model, the attacker controls all communication, non-deterministically schedule the participants, and non-deterministically inject new messages, which are computed using the whole communication transcript.

Until recently, verification techniques in this domain have converged around modeling and verifying protocols that are purely non-deterministic, where non-determinism is used to model concurrency as well as the interaction between protocol participants and their environment. In this setting, decades of work have produced many sophisticated analysis tools [11, 30, 45, 15, 5]. There are, however, a large class of protocols whose correctness depends on an explicit ability to model and reason about randomness. With privacy goals in mind, these protocols lie at the heart of many anonymity systems such as Crowds [41], mix-networks [22], onion routers [34] and Tor [29]. Cryptographic protocols also

employ randomness to achieve fair exchange [10, 31], vote privacy in electronic voting [42, 21, 4, 44] and denial of service prevention [37]. The formal verification of this class of protocols has thus-far received little systematic attention.

In the absence of a systematic framework, there have been primarily two approaches to verify randomized security protocols. Works such as [49] use probabilistic model checkers [39, 27] to reason about probabilistic behavior in systems like Crowds. These ad-hoc techniques fail to capture the Dolev-Yao attacker in full generality and do not provide a general verification framework. Other works in the symbolic model [28, 38] simply abstract away essential protocol components that utilize randomization, such as anonymous channels. By making these simplifying assumptions, such analysis may miss key attacks. Indeed, we discovered in our analysis an attack on the FOO electronic voting protocol [32] that has long served as a key benchmark in the analysis of anonymity properties in the Dolev-Yao model. Our attack emerges by realizing the perfectly anonymous channels in the FOO by threshold-mixes and was missed by previous analysis. [4]

The critical challenge in the formal verification of randomized security protocols is the subtle interaction between non-determinism and randomization. If the attacker can base its non-deterministic computation on the results of private coin tosses of the participants, then the analysis necessarily may yield false attacks in correct protocols (see examples in [23, 13, 33, 19, 16]). Thus, the attacker behavior should be restricted to perform the same *computation* in any two protocol executions whose communication transcripts are indistinguishable to it. This observation is at the heart of the first framework to analyze randomized security protocols proposed in [9, 43, 17]. In this framework, the indistinguishability of two traces is captured by the trace-equivalence from the applied π-calculus [2]. The first-of-its-kind model-checking tool Stochastic Protocol ANalyer (SPAN) for checking the indistinguishability of two protocols in this framework was presented in [8]. SPAN was used to verify the 3-ballot electronic voting protocol [44] in [8].

Contributions. In this work, we describe an algorithm for analyzing the reachability-based safety properties of randomized protocols that were implemented as an extension of SPAN. The algorithm follows the bounded model checking approach of the equivalence checking in SPAN and assumes that the attacker sends messages of bounded size. The problem of checking safety reduces to the problem of computing reachability of acyclic finite state Partially-Observable Markov Decision Processes (POMPDs). The analysis of finite POMDPs is, in general undecidable. However, since we deal with acyclic POMDPs, the problem of checking reachability is decidable and can be computed by converting the POMDP into a fully-observable belief Markov Decision Processes. Our algorithm exploits the acyclicity of the POMDPs to construct the belief MDP on-the-fly by discovering the states of the belief MDP using the Depth-First-Search strategy that is often used to solve graph reachability problems.

[4] A similar attack was also discovered by hand in [6] where the analysis of FOO protocol is carried out in the computational model.

We use SPAN to conduct the first automated symbolic analysis of several protocols including mix-networks [22], the FOO electronic voting protocol [32] and Prêt à Voter [42]. Our analysis shows that realizing perfectly anonymous channels in the FOO protocol requires non-trivial modification to the protocol design, which if not done carefully, can lead to errors. In addition, a bug in the design of the Prêt à Voter protocol was uncovered (see Section 2.2). In order to fix the bug, we propose computing the cyclic offsets in the construction of Prêt à Voter using psuedorandom permutations instead of hash functions.

Related Work. Modeling cryptographic protocols in a process calculus allowing operations for both non-deterministic and probabilistic choice was first proposed in [36]. Unfortunately, the calculus did not capture many important properties of the threat model, such as the ability for protocol participants to make private coin tosses. As a result, properties of these processes are required to be formulated through a notion of bisimulation too strong to capture many natural properties. The calculus upon which our techniques are built first appeared in [9], where the authors studied the conditions under which reachability properties of randomized security protocols are preserved by composition. In [43] the composition framework was extended to handle equivalence properties. SPAN was originally presented in [8], which discusses the design and implementation of the algorithms for checking equivalence properties. For randomized security protocols, the complexity of verifying reachability and equivalence properties was studied in [17]. The material presented here also appears in the Ph.D. thesis of Matthew S. Bauer (See [7]), and we refer the reader to the thesis for a detailed discussion of the tool architecture and of experimental results.

2 Randomized Security Protocols

In what follows, we give the details behind several security protocols that utilize randomization. These protocols will serve as running examples upon which we demonstrate how our techniques can be used for modeling and automated analysis.

2.1 Mix Networks

A mix-network [22] is a routing protocol used to break the link between a message's sender and receiver. The unlinking is achieved by routing messages through a series of proxy servers, called mixes. Each mix collects a batch of encrypted messages, privately decrypts each message, and forwards the resulting messages in random order. More formally, consider a sender Alice (A) who wishes to send a message m to Bob (B) through mix (M). Alice prepares a cipher-text of the form

$$\mathsf{aenc}(\mathsf{aenc}(m, n_1, \mathsf{pk}(B)), n_0, \mathsf{pk}(M))$$

where aenc is asymmetric encryption, n_0, n_1 are nonces and $\mathsf{pk}(M)$, $\mathsf{pk}(B)$ are the public keys of the Mix and Bob, respectively. Upon receiving a batch of N such cipher-texts, the mix M unwraps the outer layer of encryption on each message using its secret key and then randomly permutes and forwards the messages. A passive attacker, who observes all traffic but does not otherwise modify the network, cannot (with high probability) correlate messages entering and exiting the mix M. Unfortunately, this simple design, known as a *threshold mix*, is vulnerable to a straightforward active attack. To expose Alice as the sender of the message $\mathsf{aenc}(m, n_1, \mathsf{pk}(B))$, an attacker forwards Alice's message along with $N-1$ dummy messages to the mix M. In this way, the attacker can distinguish which of M's N output messages is not a dummy message and hence must have originated from Alice. Although active attacks of this nature cannot be thwarted completely, several mix-network designs have been proposed to increase the overhead associated with carrying out such an attack.

2.2 Prêt à Voter

Prêt à Voter [42] is a mix-network based voting protocol that provides a simple and intuitive mechanism by which a set of voters $(V_1, ..., V_n)$ can carry out elections with the help of a set of honest tellers $(T_1, ..., T_k)$ and an honest election authority (A). Each teller has two public key pairs. Using these keys and a set of random values, the authority creates a set of ballot forms with the following properties. Each ballot has two columns; the left column lists the candidates in a permuted order and the right column provides space for a vote to be recorded. The bottom of the right column also holds an "onion" which encodes the permuted ordering (cyclic offset) for the candidates on the left-hand side of the ballot.

The precise construction of a ballot is as follows. The authority first generates a random seed,

$$\text{seed} := g_0, g_1, ..., g_{2k-1}$$

where each g_i (for $i \in \{1, ..., 2k-1\}$), called a germ, is drawn from an appropriately sized field. For a candidate list of size v, the seed is used generate the cyclic offset

$$\theta := \sum_{i=0}^{2k-1} d_i (\mathsf{mod}\ v)$$

where $d_i := \mathsf{hash}(g_i)(\mathsf{mod}\ v)$. Each teller i has public keys $\mathsf{pk}(T_{2i})$ and $\mathsf{pk}(T_{2i-1})$ which are used to construct the onion

$$\{\langle g_{2k-1}, \{\langle g_{2k-1}, ...\{\langle g_0, D_0\rangle\}_{\mathsf{pk}(T_0)}...\rangle\}_{\mathsf{pk}(T_{2k-2})}\rangle\}_{\mathsf{pk}(T_{2k-1})}$$

where D_0 is a nonce uniquely chosen for each onion. Each layer $D_{i+1} := \{\langle g_i, D_i\rangle\}_{\mathsf{pk}(T_i)}$ asymmetrically encrypts a germ and the previous layer of the onion.

The election authority generates a number of ballots which far exceed the number of voters. In order to cast a vote, a voter authenticates with the authority, after which a random ballot is chosen by the voter. In the voting booth, the voter

marks his/her choice on the right-hand side of the ballot and removes the left-hand side for shredding. The values on the right side of the ballot (the vote position and onion) are read by a voting device and then retained by the voter as a receipt. Once read by the voting device, the values are passed to the tellers that manipulate pairs of the form $\langle r_{2i}, D_{2i}\rangle$. The first teller receives the pair $\langle r, D_{2k}\rangle$ where r is the vote position, and D_{2k} is the onion. Upon receiving such a pair, each teller T_{i-1} performs the following operations.

- Apply the secret key $\mathsf{sk}(T_{2i-1})$ to D_{2i} to reveal the germ g_{2i-1} and the next layer of the onion D_{2i-1}.
- Recover $d_{2i-1} = \text{hash}(g_{2i-1})(\mathsf{mod}\ v)$ and obtain $r_{2i-1} = (r_{2i} - d_{2i-1})(\mathsf{mod}\ v)$.
- Form the new pair $\langle r_{2i-1}, D_{2i-1}\rangle$.

After applying this transformation for each pair in the batch it receives, teller T_{i-1} performs a secret shuffle on the resulting transformed pairs. Teller T_{i-1} then repeats this process on the shuffled values using its second secret key $\mathsf{sk}(T_{2i-2})$ to obtain a new set of pairs with the form $\langle r_{2i-2}, D_{2i-2}\rangle$. These pairs are shuffled again and then passed to the next teller T_{i-2}. The output of the last teller is the value of r_0 which identifies a voter's vote.

Our analysis of this version of the Prêt à Voter protocol has uncovered a previously unknown flaw in the protocol's design. The error arises from the assumption that the elements of the field from which the germs are drawn are evenly distributed when their hash is taken modulo v. To understand this error in more detail, let us consider the simple case when there are two candidates (0 and 1) and one teller. Let F be a field with M elements and

$$F_j = \{g \mid g \in F \text{ and } \mathsf{hash}(g)(\mathsf{mod}\ 2) = j\}$$

for $j \in \{0, 1\}$. There is no guarantee that $F_0 = F_1$ and thus the probability of the two cyclic offsets $\theta_0 = (\frac{F_0}{F})(\frac{F_0}{F}) + (\frac{F_1}{F})(\frac{F_1}{F})$ and $\theta_1 = 2(\frac{F_0}{F})(\frac{F_1}{F})$ in the randomly chosen ballots may be different. This can give an attacker an advantage in attempting to infer a vote from a ballot receipt: the attacker will guess that cyclic shift is the one happens with higher probability. To fix this issue, the hash function should be replaced by a pseudo-random permutation.

3 Randomized Applied π-Calculus

In this section, we present our core process calculus for modeling cryptographic protocols with coin tosses. The presentation of the calculus is borrowed from [8], and closely resembles the ones from [9, 43, 17]. As was first proposed in [36], it extends the applied π-calculus by the inclusion of a new operator for probabilistic choice.

3.1 Terms, equational theories and frames

A signature $\mathcal{F}$ contains a finite set of function symbols, each with an associated arity and two special countable sets of constant symbols $\mathcal{M}$ and $\mathcal{N}$ representing

public and private names, respectively. Variable symbols are the union of two disjoint sets $\mathcal{X}$ and $\mathcal{X}_w$, used to represent protocol and frame variables, respectively. The sets $\mathcal{F}$, $\mathcal{M}$, $\mathcal{N}$, $\mathcal{X}$ and $\mathcal{X}_w$ are required to be pairwise disjoint. Terms are built by the application of function symbols to variables and terms in the standard way. Given a signature $\mathcal{F}$ and $\mathcal{Y} \subseteq \mathcal{X} \cup \mathcal{X}_w$, we use $\mathcal{T}(\mathcal{F}, \mathcal{Y})$ to denote the set of terms built over $\mathcal{F}$ and $\mathcal{Y}$. The set of variables occurring in a term u is denoted by $\mathsf{vars}(u)$. A ground term is one that contains no free variables. The *depth* of a term t is defined to be the depth of the dag that represents t.

A substitution σ is a partial function with a finite domain that maps variables to terms, where $\mathsf{dom}(\sigma)$ will denote the domain and $\mathsf{ran}(\sigma)$ will denote the range. For a substitution σ with $\mathsf{dom}(\sigma) = \{x_1, \ldots, x_k\}$, we will denote σ as $\{x_1 \mapsto \sigma(x_1), \ldots, x_k \mapsto \sigma(x_k)\}$. A substitution σ is said to be ground if every term in $\mathsf{ran}(\sigma)$ is ground and a substitution with an empty domain will be denoted as $\emptyset$. Substitutions can be extended to terms in the usual way and we write $t\sigma$ for the term obtained by applying the substitution σ to the term t.

Our process algebra is parameterized by an equational theory $(\mathcal{F}, E)$, where $\mathcal{F}$ is a signature and E is a set of $\mathcal{F}$-Equations. By an $\mathcal{F}$-Equation, we mean a pair $u = v$ where $u, v \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X})$ are terms that do not contain private names.

Example 1. We can model primitives for symmetric encryption/decryption and a hash function using the equational theory $(\mathcal{F}_{\mathsf{senc}}, E_{\mathsf{senc}})$ with signature $\mathcal{F}_{\mathsf{senc}} = \{\mathsf{senc}/2,\ \mathsf{sdec}/2,\ \mathsf{h}/1\}$ and equations $E_{\mathsf{senc}} = \{\mathsf{sdec}(\mathsf{senc}(m, k), k) = m\}$.

Two terms u and v are said to be equal with respect to an equational theory $(\mathcal{F}, E)$, denoted $u =_E v$, if $E \vdash u = v$ in the first order theory of equality. For equational theories defined in the preceding manner, if two terms containing private names are equivalent, they will remain equivalent when the names are replaced by arbitrary terms. We often identify an equational theory $(\mathcal{F}, E)$ by E when the signature is clear from the context. An equational theory E is said to be trivial if $u =_E v$ for any terms u and v and, otherwise it is said to be non-trivial. For the remainder of this work, we will assume equational theories are non-trivial. Processes are executed in an environment that consists of a frame $\varphi : \mathcal{X}_w \to \mathcal{T}(\mathcal{F})$ and a binding substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$.

Definition 1. *Two frames φ_1 and φ_2 are said to be statically equivalent in equational theory E, denoted $\varphi_1 \equiv_E \varphi_2$, if $\mathsf{dom}(\varphi_1) = \mathsf{dom}(\varphi_2)$ and for all $r_1, r_2 \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X}_w)$ we have $r_1\varphi_1 =_E r_2\varphi_1$ iff $r_1\varphi_2 =_E r_2\varphi_2$.*

Intuitively, two frames are statically equivalent if an attacker cannot distinguish between the information they contain. A term $u \in \mathcal{T}(\mathcal{F})$ is deducible from a frame φ with recipe $r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathsf{dom}(\varphi))$ in equational theory E, denoted $\varphi \vdash^r_E u$, if $r\varphi =_E u$. We often omit r and E and write $\varphi \vdash u$ if they are clear from the context.

3.2 Process syntax

We assume a countably infinite set of labels $\mathcal{L}$ and an equivalence relation $\sim$ on $\mathcal{L}$ that induces a countably infinite set of equivalence classes. For $\ell \in \mathcal{L}$, $[\ell]$

denotes the equivalence class of ℓ. Each equivalence class is assumed to contain a countably infinite set of labels. Operators in our grammar will come with a unique label from $\mathcal{L}$, which, together with the relation $\sim$, will be used to mask the information an attacker can obtain about the actions of a process. When an action with label ℓ is executed, the attacker will only be able to infer $[\ell]$.

Processes in our calculus are a finite parallel composition of roles, which intuitively are used to model a single actor in a system/protocol. Please note that we are modeling only a finite number of sessions. Hence we do not allow replication in our protocol syntax. Roles, in turn, are constructed by combining atomic actions through sequential composition and probabilistic choice. Formally, an atomic action is derived from the grammar

$$A \;:=\; 0 \,\big|\, \nu x^{\ell} \,\big|\, (x := u)^{\ell} \,\big|\, [c_1 \wedge \ldots \wedge c_k]^{\ell} \,\big|\, \mathtt{in}(x)^{\ell} \,\big|\, \mathtt{out}(u)^{\ell}$$

where $\ell \in \mathcal{L}$, $x \in \mathcal{X}$ and $c_i \in \{\top, u = v\}$ for all $i \in \{1, \ldots, k\}$ where $u, v \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X})$. In the case of the assignment rule $(x := u)^{\ell}$, we additionally require that $x \notin \mathsf{vars}(u)$. A role is derived from the grammar

$$R \;:=\; A \,\big|\, (R \cdot R) \,\big|\, (R +^{\ell}_{p} R)$$

where $p \in [0, 1]$, $\ell \in \mathcal{L}$ and $x \in \mathcal{X}$. The 0 process does nothing. The process νx^{ℓ} creates a fresh name and binds it to x while $(x := u)^{\ell}$ assigns the term u to the variable x. The test process $[c_1 \wedge \ldots \wedge c_k]^{\ell}$ terminates if c_i is $\top$ or c_i is $u = v$ where $u =_E v$ for all $i \in \{1, \ldots, k\}$ and otherwise, if some c_i is $u = v$ and $u \neq_E v$, the process deadlocks. The process $\mathtt{in}(x)^{\ell}$ reads a term u from the public channel and binds it to x and the process $\mathtt{out}(u)^{\ell}$ outputs a term on the public channel. The processes $R \cdot R'$ sequentially executes R followed by R' whereas the process $R +^{\ell}_{p} R'$ behaves like R with probability p and like R' with probability $1 - p$. Note that protocols in our formalism are *simple*; a protocol is said to be simple if there is no principal-level nondeterminism [25].

We will use P and Q to denote processes, which are the parallel composition of a finite set of roles $R_1, \ldots, R_n$, denoted $R_1 \mid \ldots \mid R_n$. For a process Q, $\mathsf{fv}(Q)$ and $\mathsf{bv}(Q)$ denote the set of variables that have some free or bound occurrence in Q, respectively. The formal definition is standard and is omitted for lack of space. Processes containing no free variables are called ground. We restrict our attention to processes that do not contain variables with both free and bound occurrences. That is, for a process Q, $\mathsf{fv}(Q) \cap \mathsf{bv}(Q) = \emptyset$.

Definition 2. *A process $Q = R_1 \mid \ldots \mid R_n$ is said to be well-formed if the following hold.*

1. *Every atomic action and probabilistic choice in Q has a distinct label.*
2. *If label ℓ_1 (resp. ℓ_2) occurs in the role R_i (resp. R_j) for $i, j \in \{1, \ldots, n\}$ then $i \neq j$ iff $[\ell_1] \neq [\ell_2]$.*

For the remainder of this work, processes are assumed to be well-formed. Unless otherwise stated, we will also assume that the labels occurring a role come from the same equivalence class.

Remark 1. For readability, we will omit process labels when they are not relevant in a particular context.

We now present an example illustrating the type of protocols that can be modeled in our process algebra.

Example 2. Using our process syntax, we model a simple threshold mix, as described in Section 2.1. We will consider the situation when there two users A_0 and A_1 who want to communicate anonymously through a single mix server M with users B_0 and B_1, respectively. The protocol is built over the equational theory with signature $\mathcal{F}_{\mathsf{aenc}} = \{\mathsf{sk}/1,\ \mathsf{pk}/1,\ \mathsf{aenc}/3,\ \mathsf{adec}/2,\ \mathsf{pair}/2,\ \mathsf{fst}/1,\ \mathsf{snd}/1\}$ and the equations E_{aenc} given below.

$$\begin{aligned}\mathsf{adec}(\mathsf{aenc}(m, r, \mathsf{pk}(k)), \mathsf{sk}(k)) &= m\\ \mathsf{fst}(\mathsf{pair}(m_1, m_2)) &= m_1\\ \mathsf{snd}(\mathsf{pair}(m_1, m_2)) &= m_2\end{aligned}$$

For generation of their pubic key pairs, the parties A_0, A_1, B_0, B_1 and M will hold private names k_{A_0}, k_{A_1}, k_{B_0}, k_{B_1}, and k_M, respectively. The protocol will also have private names $n_0, n_1, n_2, \ldots$ to model nonces. The nonces n_0 and n_1 are the messages that A_0 and A_1 want to communicate. The behavior of each user and the mix can be described by the roles below (where we use $\langle , \rangle$ in place of pair for succinctness).

$$\begin{aligned}A_0 &= \mathtt{out}(\mathsf{aenc}(\mathsf{aenc}(n_0, n_2, \mathsf{pk}(k_{B_0})), n_4, \mathsf{pk}(k_M)))\\ A_1 &= \mathtt{out}(\mathsf{aenc}(\mathsf{aenc}(n_1, n_3, \mathsf{pk}(k_{B_1})), n_5, \mathsf{pk}(k_M)))\\ M &= \mathtt{in}(z_1) \cdot \mathtt{in}(z_2) \cdot\\ &\quad \mathtt{out}(\langle \mathsf{adec}(z_1, \mathsf{sk}(k_M)), \mathsf{adec}(z_2, \mathsf{sk}(k_M))\rangle +_{\frac{1}{2}}\\ &\qquad \langle \mathsf{adec}(z_2, \mathsf{sk}(k_M)), \mathsf{adec}(z_1, \mathsf{sk}(k_M))\rangle)\end{aligned}$$

3.3 Partially Observable Markov Decision Processes

POMDPs are used to model processes that exhibit both probabilistic and nondeterministic behavior, where the states of the system are only partially observable. Formally, a POMDP is a tuple $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ where Z is a countable set of *states*, $z_s \in Z$ is the *initial state*, Act is a countable set of *actions*, $\Delta : Z \times \mathsf{Act} \hookrightarrow \mathsf{Dist}(Z)$ is a partial function called the *probabilistic transition relation*, $\mathcal{O}$ is a countable set of observations and $\mathsf{obs} : Z \to \mathcal{O}$ is a labeling of states with observations. The POMDP $\mathcal{M}$ is said to be a *fully observable MDP* if obs is an injective function. For a distribution μ over Z, let $\mathsf{support}(\mu) = \{z \in Z \mid \mu(z) > 0\}$. An *execution* ρ of the $\mathcal{M}$ is a finite sequence $z_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} z_m$ such that $z_0 = z_s$ and for each $i \geq 0$, $z_i \xrightarrow{\alpha_{i+1}} \mu_{i+1}$ and $z_{i+1} \in \mathsf{support}(\mu_{i+1})$. Such an execution is said to have length m, denoted $|\rho| = m$. The probability an execution ρ in $\mathcal{M}$ is $\mathsf{prob}_{\mathcal{M}}(\rho) = \prod_{i=0}^{|\rho|-1} \Delta(z_i, \alpha_{i+1})(z_i + 1)$ and the set of all executions will be denoted by $\mathsf{Exec}(\mathcal{M})$.

For each state in a POMDP, there is a choice amongst several possible probabilistic transitions. The choice of which probabilistic transition to *trigger* is resolved by an *attacker*. Informally, the process modeled by $\mathcal{M}$ evolves as follows. The process starts in the state z_s. After i execution steps, if the process is in the state z, then the attacker chooses an action α such that $\Delta(z, \alpha) = \mu$ and the process moves to state z' at the $(i+1)$-st step with probability $\mu(z')$. The choice of which action to take is determined by the sequence of observations seen by the attacker.

For an execution $\rho = z_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} z_m$ we write $\mathsf{tr}(\rho)$ to represent the *trace* of ρ, defined as the sequence $\mathsf{obs}(z_0)\alpha_1 \cdots \alpha_m \mathsf{obs}(z_m)$. The set of all traces is $\mathsf{Trace}(\mathcal{M}) = (\mathcal{O}, \mathsf{Act})^* \cdot \mathcal{O}$ and an attacker is a function $\mathcal{A} : \mathsf{Trace}(\mathcal{M}) \hookrightarrow \mathsf{Act}$. Let $\mathsf{Exec}^{\mathcal{A}}(\mathcal{M}) \subseteq \mathsf{Exec}(\mathcal{M})$ be the smallest set such that $z_s \in \mathsf{Exec}^{\mathcal{A}}(\mathcal{M})$ and if $\rho = \rho' \xrightarrow{\alpha} z \in \mathsf{Exec}^{\mathcal{A}}(\mathcal{M})$ then $\rho' \in \mathsf{Exec}^{\mathcal{A}}(\mathcal{M})$ and $\mathcal{A}(\mathsf{tr}(\rho)) = \alpha$.

State-based safety properties Given a POMDP $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$, a set $\Psi \subseteq Z$ is said to be a *state-based safety property*. An execution $\rho = z_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} z_m$ of $\mathcal{M}$ satisfies Ψ, written $\rho \models \psi$, if $z_j \in \Psi$ for all $0 \leq j \leq m$. Otherwise $\rho \not\models \psi$. We say that $\mathcal{M}$ satisfies Ψ with probability $\geq p$ against attacker $\mathcal{A}$, denoted $\mathcal{M}^{\mathcal{A}} \models_p \psi$, if the sum of the measures in the set $\{\rho \in \mathsf{Exec}^{\mathcal{A}}(\mathcal{M}) \mid \rho \text{ is a maximal and } \rho \models \psi\}$ is $\geq p$. $\mathcal{M}$ is said to satisfy Ψ with probability $\geq p$, denoted $\mathcal{M} \models_p \psi$, if for all adversaries $\mathcal{A}$, $\mathcal{M}^{\mathcal{A}} \models_p \psi$.

3.4 Process semantics

Given a process P, an extended process is a 3-tuple (P, φ, σ) where φ is a frame and σ is a binding substitution. Semantically, a ground process P over equational theory $(\mathcal{F}, E)$ is a POMDP $[\![P]\!] = (Z \cup \{\mathsf{error}\}, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ where Z is the set of all extended processes $z_s = (P, \emptyset, \emptyset)$, $\mathsf{Act} = (\mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X}_w) \cup \tau) \times \mathcal{L}/\sim$ and $\Delta, \mathcal{O}, \mathsf{obs}$ are defined below. Let $\mu \cdot Q$ denote the distribution μ_1 such that $\mu_1(P', \varphi, \sigma) = \mu(P, \varphi, \sigma)$ if P' is $P \cdot Q$ and 0 otherwise. The distributions $\mu \mid Q$ and $Q \mid \mu$ are defined analogously. For a conjunct c_i ($i \in \{1, \ldots, n\}$) in a test process $[c_1 \wedge \ldots \wedge c_n]$ and a substitution σ we write $c_i \vdash \top$ when c_i is $\top$ or c_i is $u = v$ where $\mathsf{vars}(u, v) \subseteq \mathsf{dom}(\sigma)$ and $u\sigma =_E v\sigma$. We define Δ in Figure 1, where we write $(P, \varphi, \sigma) \xrightarrow{\alpha} \mu$ if $\Delta((P, \varphi, \sigma), \alpha) = \mu$. For any extended process (P, φ, σ) and action $\alpha \in \mathsf{Act}$, if $\Delta((P, \varphi, \sigma), \alpha)$ is undefined in Figure 1 then $\Delta((P, \varphi, \sigma), \alpha) = \delta_{\mathsf{error}}$. Note that Δ is well-defined, as roles are deterministic and each equivalence class on labels identifies at most one role. For a frame φ and equational theory E, we write $[\varphi]$ to denote the equivalence class of φ with respect to the static equivalence relation $\equiv_E$. We use EQ to denote the set of all such equivalence classes. Let $\mathcal{O} = \mathsf{EQ}$ and define obs as a function from extended processes to $\mathcal{O}$ such that for any extended process $\eta = (P, \varphi, \sigma)$, $\mathsf{obs}(\eta) = [\varphi]$.

Definition 3. *An extended process (P, φ, σ) preserves the secrecy of a term u in the equational theory $(\mathcal{F}, E)$, denoted $(P, \varphi, \sigma) \models_E u$, if there is no $r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathsf{dom}(\varphi))$ such that $\varphi \vdash^r_E u\sigma$. We write $\mathsf{secret}(u)$, to represent the set*

Fig. 1 *Process semantics.*

$$\frac{r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X}_w) \quad \varphi \vdash^r u \quad x \notin \mathsf{dom}(\sigma)}{(\mathtt{in}(x)^\ell, \varphi, \sigma) \xrightarrow{(r,[\ell])} \delta_{(0,\varphi,\sigma\cup\{x\mapsto u\})}}\ \text{IN}$$

$$\frac{\mathsf{vars}(u) \subseteq \mathsf{dom}(\sigma) \quad i = |\mathsf{dom}(\varphi)| + 1}{(\mathtt{out}(u)^\ell, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(0,\varphi\cup\{w_{(i,[\ell])}\mapsto u\sigma\},\sigma)}}\ \text{OUT}$$

$$\frac{\forall i \in \{1,\ldots,n\}, c_i \vdash \top}{([c_1 \wedge \ldots \wedge c_n]^\ell, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(0,\varphi,\sigma)}}\ \text{TEST}$$

$$\frac{\mathsf{vars}(u) \subseteq \mathsf{dom}(\sigma) \quad x \notin \mathsf{dom}(\sigma)}{((x := u)^\ell, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(0,\varphi,\sigma\cup\{x\mapsto u\sigma\})}}\ \text{ASGN}$$

$$\frac{}{(Q_1 +_p^\ell Q_2, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(Q_1,\varphi,\sigma)} +_p \delta_{(Q_2,\varphi,\sigma)}}\ \text{PROB}$$

$$\frac{x \notin \mathsf{dom}(\sigma) \quad n \text{ is a fresh name}}{(\nu x^\ell, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(0,\varphi,\sigma\cup\{x\mapsto n\})}}\ \text{NEW}$$

$$\frac{Q_0 \neq 0 \quad (Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0 \cdot Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu \cdot Q_1}\ \text{SEQ}$$

$$\frac{(Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(0 \cdot Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}\ \text{NULL}$$

$$\frac{(Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0 \mid Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu \mid Q_1}\ \text{PAR}_\text{L}$$

$$\frac{((Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0 \mid Q_1, \varphi, \sigma) \xrightarrow{\alpha} Q_0 \mid \mu}\ \text{PAR}_\text{R}$$

of states of $[\![P]\!]$ *that preserve the secrecy of* u *and* $\mathsf{secret}(\{u_1,\ldots,u_n\})$ *to denote* $\mathsf{secret}(u_1) \cap \ldots \cap \mathsf{secret}(u_n)$.

Remark 2. For a process P and terms $u_1,\ldots,u_n$, $\mathsf{secret}(\{u_1,\ldots,u_n\})$ is a state-based safety property of $[\![P]\!]$. For a probability p, we will write $P \models_{E,p} \mathsf{secret}(u_1,\ldots,u_n)$, if $[\![P]\!] \models_p \mathsf{secret}(\{u_1,\ldots,u_n\})$.

Example 3. Consider the mix-net protocol $P = A_0 \mid A_1 \mid M$ defined in Example 2. The protocol is designed to ensure that the messages output by the mix cannot be linked to the original sends with high probability. That is, the adversary should be able to do no better than "guess" which output message belongs to which sender. This hypothesis is violated if, for an output of the mix, the adversary can identify the sender of the message with probability $> \frac{1}{2}$. We can model this property in our framework by adding, for each $i \in \{0,1\}$, a role

$$S_i = \mathtt{in}(z_i') \cdot [z_i' = \mathsf{aenc}(n_i, n_{i+2}, \mathsf{pk}(k_{B_i}))] \cdot \mathtt{out}(s_i)$$

to the process, where s_i is a private name. The protocol P preserves the anonymity of sender A_i if $(A_0 \mid A_1 \mid M \mid S_0 \mid S_1) \models_{E_{\mathsf{aenc}},\frac{1}{2}} \mathsf{secret}(s_i)$.

4 Model Checking Algorithm

As seen in Section 3, analyzing randomized protocols requires reasoning about their underlying semantic objects, POMDPs. In particular, we are interested in finding an attacker for a given POMDP that maximizes the probability of reaching a set of target (bad) states. Unfortunately, techniques for solving reachability problems in POMDPs are far less efficient than those for Markov Decision Processes (MDPs), the fully observable counterpart to POMDPs (where attackers are a function from *executions* to actions). The reason for the added complexity is that at any given point in the execution of a POMDP, the attacker only knows

a distribution over the current state. Further, an attacker for a POMDP needs to define a consistent strategy across all executions that produce the same sequence of observations. The actions chosen in one branch of an execution may affect the actions that can be made in another branch of the same execution. By contrast, when trying to maximize a reachability probability in an MDP, one can make a local decision about which action maximizes the probability of reaching the target states.

Several results [18, 26] corroborate this story, showing that many key verification problems for POMDPs are undecidable. Although various solution techniques have been proposed [12], and there have been successful applications to AI and planning [14], tractable reasoning about POMDPs typically relies on approximation techniques or simplifications to the model (discounts). Complicating matters further, randomized security protocols induce POMDPs that are infinitely branching. At every transition corresponding to protocol input, an infinite number of possible recipes can be supplied by a Dolev−Yao attacker. Taming the state space explosion that results from this infinite branching on inputs is a huge challenge, even in the non-randomized case. We adopt the philosophy of the SATMC [5] tool, in that, we will search for bounded attacks. That is, our tool answers the question; for a given input recipe depth k, what is the maximum probability of reaching a set of target states? The assumption of bounded recipe depth allows randomized security protocol to be modeled by POMDPs that are finite branching.

One of the most successful techniques in the approximation of optimal attackers for POMDPs is to translate a POMDP $\mathcal{M}$ into a fully observable belief MDP $\mathcal{B}(\mathcal{M})$ that emulates it. One can then analyze $\mathcal{B}(\mathcal{M})$ to infer properties of $\mathcal{M}$. The states of $\mathcal{B}(\mathcal{M})$ are probability distributions over the states of $\mathcal{M}$. Further, given a state b of $\mathcal{B}(\mathcal{M})$, if states z_1, z_2 of $\mathcal{M}$ are such that $b(z_1), b(z_2)$ are non-zero then z_1 and z_2 must have the same observation. Hence, by abuse of notation, we can define $\mathsf{obs}(b)$ to be $\mathsf{obs}(z)$ if $b(z) \neq 0$. Intuitively, an execution $\rho = b_0 \xrightarrow{\alpha_1} b_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} b_m$ of $\mathcal{B}(\mathcal{M})$ corresponds to the set of all executions ρ' of $\mathcal{M}$ such that $\mathsf{tr}(\rho') = \mathsf{obs}(b_0)\alpha_1\mathsf{obs}(b_1)\alpha_2 \cdots \alpha_m\mathsf{obs}(b_m)$. The measure of execution ρ in $\mathcal{B}(\mathcal{M})$ is exactly $\mathsf{prob}_{\mathcal{M}}(\mathsf{obs}(b_0)\alpha_1\mathsf{obs}(b_1)\alpha_2 \cdots \alpha_m\mathsf{obs}(b_m))$.

The initial state of $\mathcal{B}(\mathcal{M})$ is the distribution that assigns 1 to the initial state of $\mathcal{M}$. Intuitively, on a given state b of $\mathcal{B}(\mathcal{M})$ and an action α, there is at most one successor state $b^{\alpha,o}$ for each observation o. The probability of transitioning from b to $b^{\alpha,o}$ is the probability that o is observed given that the distribution on the states of $\mathcal{M}$ is b and action α is performed; $b^{\alpha,o}(z)$ is the conditional probability that the actual state of the POMDP is z. The formal definition follows.

Definition 4. *Let $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ be a POMDP. The belief MDP of $\mathcal{M}$, denoted $\mathcal{B}(\mathcal{M})$, is the tuple $(\mathsf{Dist}(Z), \delta_{z_s}, \mathsf{Act}, \Delta^{\mathcal{B}})$ where $\Delta^{\mathcal{B}}$ is defined as follows. For $b \in \mathsf{Dist}(Z)$, action $\alpha \in \mathsf{Act}$ and $o \in \mathcal{O}$, let*

$$p_{b,\alpha,o} = \sum_{z \in Z} b(z) \cdot \left(\sum_{z' \in Z \wedge \mathsf{obs}(z') = o} \Delta(z, \alpha)(z') \right).$$

$\Delta^{\mathcal{B}}(b,\alpha)$ *is the unique distribution such that for each* $o \in \mathcal{O}$, *if* $p_{b,\alpha,o} \neq 0$ *then* $\Delta^{\mathcal{B}}(b,\alpha)(b^{\alpha,o}) = p_{b,\alpha,o}$ *where for all* $z' \in Z$,

$$b^{\alpha,o}(z') = \begin{cases} \frac{\sum_{z \in Z} b(z) \cdot \Delta(z,\alpha)(z')}{p_{b,\alpha,o}} & \text{if } \mathsf{obs}(z') = o \\ 0 & \text{otherwise} \end{cases}.$$

This definition results in a correspondence between the maximal reachability probabilities in a POMDP $\mathcal{M}$ and the belief MDP $\mathcal{B}(\mathcal{M})$ it induces. The following proposition, due to Norman et al. [40], makes this correspondence precise. In the result below, for a POMDP (resp. MDP) $\mathcal{M}$ and a set of observations O (resp. states T), we write $\mathsf{prob}^{max}_{\mathcal{M}}(O)$ (resp. $\mathsf{prob}^{max}_{\mathcal{M}}(T)$) to denote the maximum probability with which $\mathcal{M}^A$ reaches states with observations in O (resp. states from T) for any adversary $\mathcal{A}$.

Proposition 1. *Let* $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ *be a POMDP,* $O \subseteq \mathcal{O}$ *and* $T_O = \{b \in \mathsf{Dist}(Z) \mid \forall z \in Z.(b(z) > 0 \Rightarrow \mathsf{obs}(z) \in O)\}$. *Then* $\mathsf{prob}^{max}_{\mathcal{M}}(O) = \mathsf{prob}^{max}_{\mathcal{B}(\mathcal{M})}(T_O)$.

In general, belief MDPs are defined over a continuous state space; even simple POMDP models can yield an infinite number of distributions on states. It is this continuous state space that makes belief MDPs difficult to analyze. Fortunately, the calculus from Section 3.2 doesn't include an operator for replication. This means that protocol executions are of a fixed length and can be encoded as acyclic POMDPs that reach a set of finite absorbing states after a bounded number of actions. However, even for acyclic POMDPs, the number of reachable belief states can grow much larger than the number of states in the original POMDP.

Let Q be a randomized security protocol such that $[\![Q]\!] = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$. Define $[\![Q_d]\!] = (Z, z_s, \mathsf{Act}_d, \Delta_d, \mathcal{O}, \mathsf{obs})$ where every $\alpha \in \mathsf{Act}_d$ is such that $\mathsf{depth}(\alpha) \leq d$ and for all $z \in Z$, $\Delta_d(z,\alpha) = \Delta(z,\alpha)$ if $\alpha \in \mathsf{Act}_d$ and otherwise $\Delta_d(z,\alpha)$ is undefined. For a security protocol Q, probability p and safety property ψ, the *bounded model checking problem* for depth d is to determine if $[\![Q_d]\!] \models_p \psi$. As described above, $[\![Q_d]\!]$ can be translated into a finite acyclic fully observable belief MDP $\mathcal{B}([\![Q_d]\!])$. By analyzing $\mathcal{B}([\![Q_d]\!])$, one can generate an attacker for $[\![Q_d]\!]$ that optimizes the probability of reaching a target set of states $Z \setminus \psi$. These optimal reachability probabilities can be computed using Algorithm 1, where we assume a finite set of absorbing states B_{abs}. The algorithm works by recursively computing the maximum probability of attack by exploring states in a depth-first fashion. Such an approach can avoid exploring many redundant portions of the state space.

The correctness of our algorithm, which follows from Proposition 1, is given below.

Theorem 1. *Let* Q *be a protocol and* $d \in \mathbb{N}$ *be such that* $[\![Q_d]\!] = (Z, z_s, \mathsf{Act}_d, \Delta, \mathcal{O}, \mathsf{obs})$. *For a given probability* p *and state-based safety property* $\psi \subseteq Z$, *if* $[\![Q_d]\!] \models_p \psi$ *iff* $\textsc{maxAttack}(\delta_z, Z \setminus \psi) \leq 1 - p$ *for the belief MDP* $\mathcal{B}([\![Q_d]\!])$.

Algorithm 1 *On-the-fly model checking of safety properties in finite-length belief MDPs.*

```
1: procedure MAXATTACK(beliefState b, targetStates T)
2:     p ← 0
3:     if b ∈ B_abs then
4:         for z ∈ support(b) do
5:             if z ∈ T then
6:                 p ← p+b(z)
7:         return p
8:     for α ∈ Act do
9:         for o ∈ O do
10:            p ← max(p, MAXATTACK(b^{α,o}, T))
11:            if p == 1 then
12:                return 1
13:    return p
```

5 Tool Description and Evaluation

The algorithm for checking safety in randomized security protocols is implemented in the tool, SPAN. We refer the reader to [7] for a detailed discussion of the implementation and evaluation of SPAN. We describe the salient features briefly.

Implementation. As described in Section 4, the fundamental routine of SPAN translates a randomized security protocol into a belief MDP. Each translation step requires operations from term rewriting as well as solving the static equivalence and deduction problems on protocol frames. Currently, SPAN supports two external engines for solving the static equivalence and deduction questions: KISS [3] and AKISS [15]. KISS tool supports sub-term convergent theories, while the AKISS tool supports more general optimally reducing theories and the AC operation XOR. SPAN implements its own unification algorithm for convergent equational theories for its term-rewriting engine. For rewriting in the presence of AC operations, support for integration with Maude [30, 24] is also included. Because attacks on randomized protocols are trees (as opposed to sequences) attacks are exported to DOT format, which can be rendered visually using the graphviz framework [1].

Evaluation. We evaluated SPAN on a variety of protocols. Our experiments were conducted on an Intel core i7 dual quad-core processor at 2.67GHz with 12GB of RAM. The host operating system was 64 bit Ubuntu 16.04.3 LTS. The examples that we verified were sender anonymity in Dining Cryptographers-Net [35, 20],, threshold mixes [22] and pool mixes [46–48], and vote privacy in FOO voting protocol [32] and Prêt à Voter protocol [42]. We attempted to verify all protocols with a recipe depth of 10; however, for some examples, SPAN did not terminate within a reasonable time-bound. In such cases, we report the time for a recipe

Table 1 *Experimental results for safety properties.* Columns 1-5 describe the example under test, where column 2 is the number of users in the protocol, column 3 is maximum recipe depth, column 4 is the maximum attack probability and column 5 is the security threshold: if the value of column 4 exceeds the value of column 5, then an attack was found. Columns 6 and 7 give the running times (in seconds) under the KISS and Akiss, respectively. Column 8 reports the number of belief states explored during the model checking procedure. All test were conducted using Maude 2.7.1 as the term rewriting engine. For protocols with requiring equational theories with XOR we write n/s (not supported) for the KISS engine.

1	2	3	4	5	6	7	8
PROTOCOL	PARTIES	DEPTH	ATTACK	THRESHOLD	TIME (S)		BELIEFS
					w/ KISS	w/ AKISS	
DC-net	2	10	1/2	1/2	n/s	23	110
Threshold Mix	4	10	1	1/4	22	70	49
Cascade Mix	2	5	1	1/2	917	2832	55303
Pool Mix	3	5	2/3	1/3	1824	6639	26273
FOO 92 (corrected)	2	10	3/4	3/4	321	918	1813
Prêt à Voter	2	10	7/8	3/4	n/s	288	103

depth of 5. As mentioned above, mixes are vulnerable to active attacks, and our tool was able to capture these attacks. For the FOO voting protocol, we implemented the anonymous channels using threshold mixes. In the previous automated analysis of FOO voting protocol (See [15], for example), perfectly anonymous channels are assumed to exist. This abstraction misses possible attacks. For example, if a threshold mix is used to implement the FOO protocol, then SPAN found an attack on vote privacy that exploits the flooding attack on mixes. A similar attack has also been previously reported in [6], which carries out the analysis of FOO voting protocol in the computational model, and was discovered by hand. We propose corrections to the FOO protocol to avoid such attacks. Finally, in order to capture the attack on Prêt à Voter protocol described above, we assumed that the sum of two hashes is even with probability $\frac{3}{4}$ and odd with probability $\frac{1}{4}$.

6 Conclusion

We present a bounded model checking algorithm to verify safety properties of acyclic randomized security protocols. As randomized security protocols are naturally modeled as POMDPs, we adapt the belief MDP construction from POMPDP literature in the design of the algorithm. The algorithm exploits the acyclic nature of the protocols considered and constructs the belief MDP by traversing the belief MDP in a Depth First Search fashion. The algorithm is implemented

as an extension of SPAN. Our experiments demonstrate the effectiveness of the tool in uncovering previously unknown attacks in protocols.

We plan to investigate the use of partial order reduction and symmetry reduction techniques to combat the state explosion problem. We also plan to investigate the verification of randomized security protocols without any restriction of recipe sizes. Another line of investigation that we plan to pursue is the verification of cyclic randomized security protocols.

Acknowledgements. Andre Scedrov's foundational work on formal analysis of security protocols has been an unmistakable inspiration for us, and we thank him for his mentorship. Rohit Chadha thanks Andre Scedrov for introducing him to the exciting and challenging field of security protocol analysis, and his invaluable counsel.

Rohit Chadha was partially supported by grants NSF 1553548 CNS and NSF CCF 1900924. Mahesh Viswanathan was partially supported by NSF CCF 1901069.

References

1. Graphviz. `https://www.graphviz.org/`.
2. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Acm Sigplan Notices*, volume 36, pages 104–115. ACM, 2001.
3. Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 367(1):2 – 32, 2006. Automated Reasoning for Security Protocol Analysis.
4. Ben Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.
5. Alessandro Armando and Luca Compagna. SAT-based model-checking for security protocols analysis. *International Journal of Information Security*, 7(1):3–32, Jan 2008.
6. Gergei Bana, Rohit Chadha, and Ajay Kumar Eeralla. Formal analysis of vote privacy using computationally complete symbolic attacker. In *23rd European Symposium on Research in Computer Security, ESORICS*, volume 11099 of *Lecture Notes in Computer Science*, pages 350–372. Springer, 2018.
7. Matthew S. Bauer. *Analysis of randomized security protocols.* PhD thesis, University of Illinois at Urbana-Champaign, 2018.
8. Matthew S. Bauer, Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan. Model checking indistinguishability of randomized security protocols. In *Computer Aided Verification - 30th International Conference, CAV*, volume 10982 of *Lecture Notes in Computer Science*, pages 117–135. Springer, 2018.
9. Matthew S. Bauer, Rohit Chadha, and Mahesh Viswanathan. Composing protocols with randomized actions. In *Principles of Security and Trust*, pages 189–210, 2016.
10. Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.
11. Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *The Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

12. Darius Braziunas. POMDP solution methods. *University of Toronto*, 2003.
13. Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Task-structured probabilistic I/O automata. In *Discrete Event Systems*, 2006.
14. Anthony R Cassandra. A survey of POMDP applications. In *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*, volume 1724, 1998.
15. Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Transactions on Computational Logic*, 17(4), 2016.
16. Rohit Chadha, A Prasad Sistla, and Mahesh Viswanathan. Model checking concurrent programs with nondeterminism and randomization. In *Foundations of Software Technology and Theoretical Computer Science*, pages 364–375, 2010.
17. Rohit Chadha, A Prasad Sistla, and Mahesh Viswanathan. Verification of randomized security protocols. In *Logic in Computer Science*, pages 1–12. IEEE, 2017.
18. Krishnendu Chatterjee, Martin Chmelík, and Mathieu Tracol. What is decidable about partially observable Markov decision processes with omega-regular objectives. *Journal of Computer and System Sciences*, 82(5):878 – 911, 2016.
19. Konstantinos Chatzikokolakis and Catuscia Palamidessi. Making random choices invisible to the scheduler. *Information and Computation*, 2010, to appear.
20. David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.
21. David Chaum, Peter YA Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In *European Symposium on Research in Computer Security*, pages 118–139. Springer, 2005.
22. David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
23. Ling Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. PhD thesis, Radboud University of Nijmegen, 2006.
24. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martı-Oliet, José Meseguer, and José F Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2):187–243, 2002.
25. Véronique Cortier and Stéphanie Delaune. A method for proving observational equivalence. In *Computer Security Foundations*, pages 266–276, 2009.
26. Luca de Alfaro. The verification of probabilistic systems under memoryless partial-information policies is hard. Technical report, 1999.
27. Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *Computer Aided Verification*, pages 592–600. Springer, 2017.
28. Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.
29. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.
30. Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design*, pages 1–50. Springer, 2009.
31. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

32. Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *International Workshop on the Theory and Application of Cryptographic Techniques*, pages 244–251. Springer, 1992.
33. Flavio D Garcia, Peter Van Rossum, and Ana Sokolova. Probabilistic anonymity and admissible schedulers. *arXiv preprint arXiv:0706.1019*, 2007.
34. David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Workshop on Information Hiding*, pages 137–150, 1996.
35. Philippe Golle and Ari Juels. Dining cryptographers revisited. In *Theory and Applications of Cryptographic Techniques*, pages 456–473. Springer, 2004.
36. Jean Goubault-Larrecq, Catuscia Palamidessi, and Angelo Troina. A probabilistic applied pi–calculus. In *Asian Symposium on Programming Languages and Systems*, pages 175–190, 2007.
37. Carl A. Gunter, Sanjeev Khanna, Kaijun Tan, and Santosh S. Venkatesh. DoS protection for reliably authenticated broadcast. In *Network and Distributed System Security*, 2004.
38. Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi calculus. In *European Symposium on Programming*, pages 186–200. Springer, 2005.
39. Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification*, pages 585–591. Springer, 2011.
40. Gethin Norman, David Parker, and Xueyi Zou. Verification and control of partially observable probabilistic systems. *Real-Time Systems*, 53(3):354–402, May 2017.
41. Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.
42. Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à voter: A voter-verifiable voting system. *IEEE Transactions on Information Forensics and Security*, 4(4):662–673, 2009.
43. Matthew S. Bauer, Rohit Chadha, and Mahesh Viswanathan. Modular verification of protocol equivalence in the presence of randomness. In *European Symposium on Research in Computer Security*, pages 187–205, 01 2017.
44. Altair O Santin, Regivaldo G Costa, and Carlos A Maziero. A three-ballot-based secure electronic voting system. *Security and Privacy*, 6(3):14–21, 2008.
45. Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *Computer Security Foundations*, pages 78–94, 2012.
46. Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In *International Workshop on Information Hiding*, pages 36–52. Springer, 2002.
47. Andrei Serjantov and Richard E Newman. On the anonymity of timed pool mixes. In *International Information Security Conference*, pages 427–434. Springer, 2003.
48. Andrei Serjantov and Peter Sewell. Passive attack analysis for connection-based anonymity systems. In *European Symposium on Research in Computer Security*, pages 116–131. Springer, 2003.
49. Vitaly Shmatikov. Probabilistic analysis of anonymity. In *Computer Security Foundations*, pages 119–128. IEEE, 2002.