Conditional Observational Equivalence and Off-line Guessing Attacks in Multiset Rewriting

Petar Paradžik
Faculty of Electrical Engineering and Computing
Zagreb, Croatia
petar.paradzik@fer.hr

Ante Derek
Faculty of Electrical Engineering and Computing
Zagreb, Croatia
ante.derek@fer.hr

2022 IEEE 35th Computer Security Foundations Symposium (CSF) | 978-1-6654-8417-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/CSF54842.2022.9919659

***Abstract*—We propose *conditional observational equivalence* — a variant of observational equivalence that is more flexible since it can be made dependent on arbitrary safety trace properties. We extend an existing method for verifying observational equivalence in the multiset rewriting setting with the ability to handle conditions. Our extension can automatically verify conditional observational equivalence for a simple class of conditions that depend only on the structure of the execution. By using conditional observational equivalence, we give the first method for verifying off-line guessing resistance in the multiset rewriting setting and apply it to analyze and verify the properties of EAP-EKE, a password-authenticated key exchange (PAKE) protocol.**

***Index Terms*—observational equivalence, off-line guessing, EKE, PAKE, multiset rewriting**

I. Introduction

When analyzing cryptographic protocols in the symbolic model, security properties are usually specified either as *trace* properties or *observational equivalence* properties. Roughly speaking, trace properties state that desirable conditions — such as key secrecy or authentication — hold in all possible executions of a system consisting of the protocol participants and the attacker. Observational equivalence properties state that from the point of view of an attacker interacting with protocol participants, two different protocols appear exactly the same. Most often, observational equivalence is used to specify and verify properties related to anonymity and privacy. However, it can also be used to prove that a cryptographic protocol satisfies a "conventional" security property, such as key secrecy, by demonstrating its observational equivalence with an ideal protocol that has the desired property by construction.

Although verifying observational equivalence is generally undecidable, automated procedures were developed [1, 2, 3, 4, 5] that successfully verify equivalence in certain cases (see [6] for a survey of decidability results and [7] for a survey of verification methods). ProVerif [1], Tamarin prover [8, 2], and other tools were used to provide machine-verified security proofs based on observational equivalence for various real-world protocols — e.g., privacy-related properties in e-voting protocols [9, 10]; anonymity properties in attestation and key-exchange protocols [11, 12, 13, 14]; unlinkability in authentication protocols [15].

In symbolic protocol analysis, trace properties can be fine-grained — e.g., we can use Tamarin prover to state and prove that complex key secrecy and authentication properties hold under various assumptions. Assumptions can specify necessary preconditions for desired properties (e.g., that the attacker has not compromised the long-term keys), but can also be used to implement protocol features such as equality checks and branching. This is done by limiting the scope of the analysis to only those executions where the assumptions hold (these types of assumptions are commonly referred to as *restrictions* in Tamarin prover). In contrast, the observational equivalence of two protocols is usually an all-or-nothing property, and any assumption or restriction must be explicitly built into the protocols being compared or the formal model itself. We believe that this inflexibility of observational equivalence is a serious limitation when dealing with complex protocols and properties.

The goal of this paper is to address this limitation by *combining observational equivalence with trace properties*. Informally, we will define two protocols L and R to be *observationally equivalent under conditions ϕ_L and ϕ_R* if an attacker cannot distinguish L and R as long as the systems L and R are executed in a way that does not violate their respective conditions ϕ_L and ϕ_R. Here, the conditions ϕ_L and ϕ_R can be any *safety* trace properties. For example, such conditions can be used to enable protocol branching (cryptographic primitive agreement), to limit the number of protocol rule instances (initialization), and to force execution ordering (phases in a game between attacker and challenger).

We choose multiset rewriting as the formal setting and define conditional observational equivalence by extending the definitions of Basin et al. [2]. Also, we extend the method for automated verification of observational equivalence given in [2] to include the ability to verify conditional observational equivalence. Our extension can automatically verify conditional observational equivalence for a class of safety properties that we call *Type-0 properties*. Informally, these properties depend on the structure of the execution (the actions that protocol participants take), but not on the data used in the execution (message terms). Therefore, Type-0 properties cannot capture equality restrictions but they can enforce protocol phases, for example.

We demonstrate the utility of conditional observational equivalence by showing how it can be used to prove the resistance of protocols to off-line guessing attacks. Informally, the attacker interacts with the protocol during the learning phase

DOI 10.1109/CSF54842.2022.00001

and tries to distinguish the secret in question from a random value in the guessing phase. The protocol is then resistant to off-line guessing if the observational equivalence holds under the condition that the execution respects the protocol phases. We believe that this is the first attempt to model off-line guessing attacks using multiset rewriting, but we note that such an analysis has been performed in a very similar way using the applied π-calculus and the tool PROVERIF [1]. We claim that our approach is more flexible since we implement protocol phases with conditions, while PROVERIF explicitly builds them into the execution model.

Finally, we perform two case studies. We show that the EAP-EKE protocol for password-authenticated key exchange is resistant to off-line guessing attacks in a general setting that includes an unbounded number of client sessions and two server sessions per client. We also extend the analysis of RFID protocols given in [16] to a more general setting.

As mentioned earlier, this work relies heavily on the automated procedure for verifying observational equivalence in the multiset rewriting setting given in Basin et al. [2]. In fact, most of the definitions and results in Sections III, IV, and V can be viewed as generalizations of the results in [2] for the case of conditional observational equivalence. In particular, we define *conditional dependency graph equivalence* and show that this implies conditional observational equivalence. We also prove a lemma that provides sufficient conditions for terminating the verification of conditional dependency graph equivalence when there are open (unresolved) premises in dependency graphs. The said lemma is crucial for achieving termination when analyzing protocols with an unbounded number of sessions.

It is important to note that TAMARIN prover already supports restrictions when proving observational equivalence. However, to the best of our knowledge, the exact semantics of such statements and the soundness of the proof method have not yet been published. Moreover, several issues have been reported by users when combining observational equivalence with restrictions. In fact, the practical issues we encountered while trying to perform a case study with TAMARIN prover and using observational equivalence with restrictions were the main motivation for this research.

In summary, our contributions are as follows:

- We propose the definition of conditional observational equivalence that combines trace properties with observational equivalence properties.
- By extending [2], we give a method for automatically verifying conditional observational equivalence for a simple class of safety properties that depend only on the execution structure. We implement our method by extending the TAMARIN prover tool.
- We use conditional observational equivalence to give the first method for verifying off-line guessing resistance in the multiset rewriting setting.
- We give machine-verified proofs of off-line guessing resistance for EAP-EKE protocol.
- We extend the case study of RFID protocols given in [16] considering readers as well.

II. PRELIMINARIES

In this section we include the necessary mathematical background. We use the notation and definitions from Basin et al. [2], but we give more verbose definitions for recipes.

1) Multiset rewriting: We use an *order-sorted signature* $\mathbf{\Sigma} = (S, \leq, \Sigma)$ with the set of sorts $S = \{fr, pub, msg\}$, a partial order $\leq$ with top sort msg, and a signature $\Sigma = \Sigma_{Fun} \uplus \Sigma_{Fact}$ as a disjoint union of the *function symbols* Σ_{Fun} and the *fact symbols* Σ_{Fact}. The fr subsort is used for the fresh values such as keys and nonces, while the pub subsort is used for the public values such as an agent name and a group generator; they are not compatible. We assume that there are countably infinite sets FN and PN of fresh and public names respectively. Function symbols Σ_{Fun} are used for the cryptographic operators, where zero-arity symbols are called *constants*, and fact symbols Σ_{Fact} are used for (recording) the system state. Moreover, we partition fact symbols on the *linear facts* — resources that can be consumed only once, and the *persistent facts* — resources that can be consumed any number of times, and define $\Sigma_{Fact} = \Sigma_{Fact_l} \uplus \Sigma_{Fact_p}$. Persistent facts will be prefixed with !, for example $!\mathsf{Fr}(x)$.

We assume that there is a countable number of the variables $\mathcal{V}_s$ for each sort $s \in S$, and define the set of all variables to be $\mathcal{V} = \biguplus_{s \in S} \mathcal{V}_s$. For $x \in \mathcal{V}_s$ we will write $x{:}s$. Given signature Σ and $\mathcal{V}' \subseteq \mathcal{V}$, the set of all *well-sorted terms* of the sort s, over $\Sigma \cup \mathcal{V}'$, is denoted by $\mathcal{T}_\Sigma(\mathcal{V}')_s$. Terms without any variables are called *ground terms*; the set of all ground terms over Σ is denoted by $\mathcal{T}_\Sigma$. Given a term $t \in \mathcal{T}_\Sigma(\mathcal{V}')$, $vars(t)$ denotes the set of all variables in t; the $root(t)$ is equal to f if $t = f(t_1, \ldots, t_k)$, and t otherwise. The set of all n-ary facts is defined as $\mathcal{F}^n = \{F(t_1, \ldots, t_n) \mid t_i \in \mathcal{T}_{\Sigma_{Fun}}(\mathcal{V}), F \in \Sigma_{Fact}, arity(F) = n\}$; the set of all facts is denoted by $\mathcal{F}$.

Given a set S and $s_1, \ldots, s_n \in S$, a *sequence* over S is denoted by $s = [s_1, \ldots, s_n]$, where $[]$ stands for the empty sequence; S^* is the set of all sequences over S; $set(s) = \{s_1, \ldots, s_n\}$ is the *set* of the sequence; $mset(s) = \{s_1, \ldots, s_n\}^\#$ is the *multiset* of the sequence; $idx(s) = \{1, \ldots, |s|\}$ is the set of *indices* of the sequence; s_i, $i \in idx(s)$ is the i-th sequence element; $|s|$ is the sequence *length*. *Concatenation* of sequences s and s' is denoted by $concat(s, s')$. *Lexicographically ordered* sequence s is denoted by $s^{\leq}$. If s is the sequence of facts, then $lfacts(s) = \{F \in mset(s) \mid root(F) \in \Sigma_{Fact_l}\}$ is the multiset of its linear facts, and $pfacts(s) = \{F \in mset(s) \mid root(F) \in \Sigma_{Fact_p}\}$ is the multiset of its persistent facts.

A *substitution* σ is a well-sorted function $\sigma{:}\,\mathcal{V} \to \mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V})$. An application of σ to a variable x is denoted by $x\sigma$. A set of variables on which σ is not the identity function is denoted by $dom(\sigma)$. If $dom(\sigma) = \{x_1, \ldots, x_k\}$, and $x_i\sigma = t_i$, we will write $\{{}^{t_1}/_{x_1}, \ldots, {}^{t_k}/_{x_k}\}$. We extend σ to an endomorphism on $\mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V})$ and use the same notation $t\sigma$ for an arbitrary term t.

Properties of cryptographic operators are specified as a set of *equations*. An equation, over the signature Σ, is an unordered pair $\{s, t\}$ of terms $s, t \in \mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V})$, written as $s \simeq t$. We extend this definition for a set of equations over a signature. An

equational presentation is the tuple $(\mathbf{\Sigma}, E)$ where E is the set of equations over the signature $\mathbf{\Sigma}$. Given an equational presentation $\mathcal{E} = (E, \mathbf{\Sigma})$, *equational theory* $=_{\mathcal{E}}$ is the smallest $\mathbf{\Sigma}$-congruence containing all $s \simeq t \in E$ instances. We use $=_E$ instead of $=_{\mathcal{E}}$ when the signature is obvious from the context. Each relation modulo E is denoted by indexing the relation with E, for example $\in_E$.

A *rewrite rule* over a signature $\mathbf{\Sigma}$ is an ordered pair (l, r) of terms $l, r \in \mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V})$, denoted by $l \to r$; a *rewrite system* $\mathcal{R}$ is the set of rewrite rules. $\mathcal{R}$ induces a *rewrite relation* $\to_{\mathcal{R}}$: $s \to_{\mathcal{R}} t$ iff there is a sequence of rewrite rules $s \to \ldots \to t$. Term s is in a *normal form* with respect to $\mathcal{R}$ written as $s \downarrow_{\mathcal{R}}$ if there is no t such that $s \to_{\mathcal{R}} t$; function symbol f is *irreducible* with respect to the rewrite system $\mathcal{R}$ if there is no $l \to r \in \mathcal{R}$ such that $root(l) = f$. We use rewrite system as a *decision procedure* for an equational theory. This is possible if an equational theory has a rewrite system that is *confluent* and *terminating*, i.e. *convergent* where every term has a unique normal form. In that case, it holds that $s =_E t$ iff $s \downarrow_{\mathcal{R}} = t \downarrow_{\mathcal{R}}$. A rewrite system $\mathcal{R}$ is called *subterm-convergent* if it is convergent and for each rule $l \to r \in \mathcal{R}$, term r is either a proper subterm of l or a ground term in normal form.

If the equational theory includes a set of equations $\mathcal{AX}$ that cannot be oriented such as associativity (A) and commutativity (C), we will use the standard notion of $\mathcal{R}, \mathcal{AX}$*-rewriting*. For a decomposition $(\mathbf{\Sigma}, \mathcal{R}, \mathcal{AX})$ that has the *finite variant property*, $\lceil t \rceil^{\mathcal{R}}$ denotes the complete set of $\mathcal{R}, \mathcal{AX}$ variants of a term $t \in \mathcal{T}_{\Sigma}(\mathcal{V})$. We use the same notation for variants of a rule and variants of a system. Equational theories with a finite variant property include Diffie-Hellman and XOR.

A labeled multiset rewrite rule is a tuple (id, l, a, r), written as $ru = id$: $[l]\text{–}[a]\text{–}\!\!\rightarrow[r]$, where $l, a, r \in \mathcal{F}^{\#}$, and $id \in \mathcal{I}$ is the unique rule identifier. The name of the rule ru is denoted by $name(ru) = id$, and sequence of premises, actions and conclusions by $prems(ru) = [l]$, $acts(ru) = [a]$ and $concs(ru) = [r]$ respectively. *Labeled multiset rewrite system* is a set of labeled multiset rewrite rules. By overloading the notation, the set of all ground instances of both labeled multiset rewrite rule ru and system P will be denoted by $ginsts(ru)$ and $ginsts(P)$ respectively. Ground instance obtained by applying a substitution σ to every fact of a rule ru is denoted by $ru\sigma$.

We partition labeled multiset rewrite system into *system* (Sys), *environment* (Env), and *interface* (IF). The system can interact with the environment using the interface

$$IF = \left\{ \begin{array}{rcl} out & : & [\mathsf{Out}_{sys}(x)]\text{–}[\mathsf{O}]\text{–}\!\!\rightarrow[\mathsf{In}_{env}(x)], \\ in & : & [\mathsf{Out}_{env}(x)]\text{–}[\mathsf{I}]\text{–}\!\!\rightarrow[\mathsf{In}_{sys}(x)] \; . \end{array} \right.$$

Example 1. *The following system (sys) aims to authenticate a user a to a server b using a password p. The environment rules are denoted by env. We will refer to this labeled multiset rewrite system throughout the paper.*

$$\begin{array}{rl} fresh^{sys}\text{:} & []\text{–}[]\text{–}\!\!\rightarrow[\mathsf{Fr}(x{:}fr)] \\ psgen^{sys}\text{:} & [\mathsf{Fr}(p)]\text{–}[\mathsf{PL}]\text{–}\!\!\rightarrow[!\mathsf{P}(p, a{:}pub, b{:}pub), \mathsf{C}(p)] \\ ureq^{sys}\text{:} & [!\mathsf{P}(p,a,b)]\text{–}[\mathsf{PL}]\text{–}\!\!\rightarrow[\mathsf{Out}_{sys}((a,b)), \mathsf{U}(p,a,b)] \\ sreq^{sys}\text{:} & [!\mathsf{P}(p,a,b), \mathsf{In}_{sys}((a,b)), \mathsf{Fr}(n)]\text{–}[\mathsf{PL}()]\text{–}\!\!\rightarrow \\ & [\mathsf{Out}_{sys}(n), \mathsf{S}(p,a,b,n)] \\ ures^{sys}\text{:} & [\mathsf{U}(p,a,b), \mathsf{In}_{sys}(n)]\text{–}[\mathsf{PL}]\text{–}\!\!\rightarrow[\mathsf{Out}_{sys}(\mathsf{enc}(n,p))] \\ sver^{sys}\text{:} & [\mathsf{S}(p,a,b,n), \mathsf{In}_{sys}(\mathsf{enc}(n,p))]\text{–}[\mathsf{PL}]\text{–}\!\!\rightarrow[] \\ chal^{sys}\text{:} & [\mathsf{C}(p), \mathsf{Fr}(f)]\text{–}[\mathsf{PG}]\text{–}\!\!\rightarrow[\mathsf{Out}_{sys}(p)] \\ recv^{env}\text{:} & [\mathsf{In}_{env}(x)]\text{–}[]\text{–}\!\!\rightarrow[!\mathsf{K}(x)] \\ send^{env}\text{:} & [!\mathsf{K}(x)]\text{–}[]\text{–}\!\!\rightarrow[\mathsf{Out}_{env}(x)] \\ dec^{env}\text{:} & [!\mathsf{K}(\mathsf{enc}(x,y)), !\mathsf{K}(z)]\text{–}[]\text{–}\!\!\rightarrow[!\mathsf{K}(\mathsf{dec}(\mathsf{enc}(x,y),z))] \\ enc^{env}\text{:} & [!\mathsf{K}(x), !\mathsf{K}(y)]\text{–}[]\text{–}\!\!\rightarrow[!\mathsf{K}(\mathsf{enc}(x,y))] \\ comp^{env}\text{:} & [!\mathsf{K}(x), !\mathsf{K}(x)]\text{–}[]\text{–}\!\!\rightarrow[] \end{array}$$

Let P be a labeled multiset rewrite system and $ru \in P$ its rule. The set of *new variables* of the k-th conclusion fact of the rule ru, is defined by

$$newvars(k, ru) = \{x \in vars(concs(ru)_k) \mid k \in idx(concs(ru)), x \notin \bigcup_{i \in idx(prems(ru))} vars(prems(ru)_i)\}.$$

For a ground rule instance $ri = ru\sigma$, we define $nvins(k, ri)$ as sequence of *instances of new variables* by applying σ to $newvars(k, ru)$ and sorting by a variable appearance.

Example 2. *Consider the rule $psgen_{sys}$ of the multiset rewrite system from Example 1, which can generate a password $x{:}fr$ for participants $z{:}pub$ and $w{:}pub$:*

$$ru = psgen_{sys}\text{:}\ [\mathsf{Fr}(p)]\text{–}[\mathsf{PL}]\text{–}\!\!\rightarrow[!\mathsf{P}(p, a{:}pub, b{:}pub), \mathsf{C}(p)] \; .$$

We have $newvars(1, ru)) = \{a, b\}$. Consider now the instance $ri = ru\sigma$ such that

$$ri = psgen_{sys}\text{:}\ [\mathsf{Fr}(p_0)]\text{–}[\mathsf{PL}]\text{–}\!\!\rightarrow[!\mathsf{P}(p_0, a_0, b_0), \mathsf{C}(p_0)] \; .$$

It holds $nvins(1, ri) = [\{^{a_0}/_{a}\}, \{^{b_0}/_{b}\}]$.

For observational equivalence, it is useful to know how a fact or rule was derived. For this reason we will use *recipes*. We define the set of *recipes* of the conclusion facts of the rule instances, denoted by ρ_f, and the set of *recipes* of the rule instances, denoted by ρ, with the following inductive definition.

R1. For a rule instance $ri = ru\sigma \in ginsts(P)$ such that $name(ru) = id$, $prems(ru) = []$, and $concs(ru) = [r_1, \ldots, r_m]$, the following holds for $k \in \{1, \ldots, m\}$.

$$\begin{array}{l} frecipe(k, ri) = id_k(nvins(k, ri), []) \in \rho_f \\ recipe(ri) = id([nvins(1, ri), \ldots, nvins(m, ri)], []) \in \rho \end{array}$$

R2. For a rule instance $ri = ru\sigma \in ginsts(P)$ such that $name(ru) = id$, $prems(ru) = [l_1, \ldots, l_n]$, and $concs(ru) = [r_1, \ldots, r_m]$, if for every (linear) premise l_i, $i \in \{1, \ldots, n\}$, there exists (an unique pair of) a rule instance $ri_i \in ginsts(P)$ and $p_i \in idx(concs(ri_i))$ such that $l_i = concs(ri_i)_{p_i}$, and $frecipe(p_i, ri_i) \in \rho_f$, then the following holds for $k \in \{1, \ldots, m\}$.

$$\begin{array}{l} frecipe(k, ri) = id_k(nvins(k, ri), \\ \qquad [frecipe(p_1, ri_1), \ldots, frecipe(p_n, ri_n)]) \in \rho_f \\ recipe(ri) = id([nvins(1, ri), \ldots, nvins(m, ri)], \\ \qquad [frecipe(p_1, ri_1), \ldots, frecipe(p_n, ri_n)]) \in \rho. \end{array}$$

The sets ρ_f and ρ are formed by first considering the rule instances with empty premises (**R1.**) and then inductively considering all other rules that can be reached from the ones that are already in the set ρ_f (**R2.**). Note that a fact can have multiple recipes, since there can be more than one way to derive it.

Example 3. *The following are the instance of the rule $ureq^{sys}$ of the multiset rewrite system from Example 1 and its recipe.*

$$\begin{aligned} ri = ureq^{sys}\colon\ & [!\mathsf{P}(p_0,a_0,b_0)] \text{-}[\mathsf{PL}]\text{-}\!> \\ & [\mathsf{Out}_{sys}(a_0,b_0), \mathsf{U}(p_0,a_0,b_0)] \\ recipe(ri) = ureq^{sys}([], [psgen_1^{sys}([\{^{a_0}/_a\}, \{^{b_0}/_b\}], \\ & [fresh_1^{sys}([\{^{p_0}/_p\}], [])])]). \end{aligned}$$

We do not want an observer (environment) to be able to distinguish systems by their internal workings because the observer cannot see the internals of the system, i.e., taken rules are hidden. This makes sense since a system corresponds to a protocol, and an environment corresponds to an adversary. The fact that the environment cannot see the rules internal to the system is expressed by the following two special recipes.

$$\begin{aligned} frecipe(1, out) &= out_1([],[z]), \\ recipe(out) &= out([],[z]). \end{aligned}$$

We see that the recipes of the premises are replaced with a new variable z, which hide how the fact $\mathsf{In}_{env}(x)$ and the rule out were derived.

Labeled transition relation $\rightarrow_P \subseteq \mathcal{G}^{\#} \times (\mathcal{G}^{\#} \times \rho) \times \mathcal{G}^{\#}$ of a multiset rewrite system P is defined as:

$$\frac{ri = id\colon [l] \text{-}[a]\text{-}\!> [r] \in_E ginsts(P) \quad lfacts(l) \subseteq^{\#} S \quad pfacts(l) \subseteq^{\#} S}{S \xrightarrow[recipe(ri)]{set(a)}_P ((S \setminus^{\#} lfacts(l)) \cup^{\#} mset(r))}$$

A multiset rewrite system with a labeled transition relation is called *labeled transition system* (LTS).

An *execution* e of the multiset rewrite system P is an alternating sequence of states and transitions:

$$e = [S_0, (l_1 \xrightarrow[rec(ri_1)]{set(a_1)} r_1), S_1, \ldots, S_{k-1}, (l_k \xrightarrow[rec(ri_k)]{set(a_k)} r_k), S_k]$$

such that $S_0 = \emptyset^{\#}$. The set of all *executions* of P is denoted by $exec_P$; the *last state* of an execution $e \in exec_P$ is denoted by $state(e)$; the *executions of state* S consists of all executions $e \in exec_P$ with S as the last state: $exec_P(S) = \{e \in exec_P \mid state(e) = S\}$; the *trace* of an execution is the sequence of its action facts: $trace(e) = [set(a_1), \ldots, set(a_k)]$; *traces* of the set of executions $R \subseteq exec_P$ are defined as $trace(R) = \{trace(e) \mid e \in R\}$.

Example 4. *The following sequence e is one possible execution of the multiset rewrite system from Example 1. It generates the pasword p_0 with the rule $fresh$, and binds it to the user a_0 and the server b_0 using the rule $psgen$. User then sends (a, b) to the server with the rule $ureq$, and the server responds with the challenge nonce n_0 using the rule $sreq$. The rule labels env and sys, and the multiset notation $\#$ are omitted for clarity. Notice the introduction of the variable z.*

$$\begin{aligned} \emptyset &\xrightarrow[fresh([\{^{p_0}/_x\}],[])]{} S_1 = \emptyset \cup \{\mathsf{Fr}(p_0)\} \\ &\xrightarrow[psgen([\{^{a_0}/_a\},\{^{b_1}/_b\}],[fresh_1([\{^{p_0}/_x\}],[])])]{} S_2 \\ &\xrightarrow[ureq([],[psgen_1([\{^{a_0}/_a\},\{^{b_1}/_b\}],[fresh_1([\{^{p_0}/_x\}],[])])])]{\mathsf{PL}} S_3 \\ &\xrightarrow[out([],[z])]{\mathsf{O}} S_4 = (S_3 \setminus \{\mathsf{Out}_{sys}((a_0,b_0))\}) \cup \{\mathsf{In}_{env}((a_0,b_0))\}, \\ &\xrightarrow[recv([],[out_1([],[z])])]{} S_5 \\ &\xrightarrow[send([],[recv_1([],[out_1([],[z])])])]{} S_6 = S_5 \cup \{\mathsf{Out}_{env}((a_0,b_0))\} \\ &\xrightarrow[in([],[send_1([],[recv_1([],[out_1([],[z])])])])]{\mathsf{I}} S_7 \\ &\xrightarrow[fresh([\{^{n_0}/_x\}],[])]{} S_8 = S_7 \cup \{\mathsf{Fr}(n_0)\} \\ &\xrightarrow[sreq([],r_0)]{\mathsf{PL}} S_9 \end{aligned}$$

$$\begin{aligned} S_2 &= (S_1 \setminus \{\mathsf{Fr}(p_0)\}) \cup \{!\mathsf{P}(p_0,a_0,b_0), \mathsf{C}(p_0)\}, \\ S_3 &= S_2 \cup \{\mathsf{Out}_{sys}((a_0,b_0)), \mathsf{U}(p_0,a_0,b_0)\}, \\ S_5 &= (S_4 \setminus \{\mathsf{In}_{env}((a_0,b_0))\}) \cup \{!\mathsf{K}((a_0,b_0))\}, \\ S_7 &= (S_6 \setminus \{\mathsf{Out}_{env}((a_0,b_0))\}) \cup \{\mathsf{In}_{sys}((a_0,b_0))\}, \\ S_9 &= (S_8 \setminus \{\mathsf{In}_{sys}((a_0,b_0)), \mathsf{Fr}(n_0)\}) \\ &\quad \cup \{\mathsf{Out}_{sys}(n_0), \mathsf{S}(p_0,a_0,b_0,n_0)\}, \\ r_0 &= [psgen_1([\{^{a_0}/_a\}, \{^{b_1}/_b\}], [fresh_1([\{^{p_0}/_x\}], [])]), \\ &\quad in_1([], [send_1([], [recv_1([], [out_1([], [z])])])]), \\ &\quad fresh_1([\{^{n_0}/_x\}], [])]. \end{aligned}$$

It holds that $trace(e') = [\mathsf{PL}, \mathsf{O}, \mathsf{I}, \mathsf{PL}]$.

2) Observational equivalence: There are many flavors of behavioral equivalence by which the systems behavior can be compared [17]. Here, we focus on the observational equivalence from [2].

Two sets of multiset rewrite rules S_A and S_B are *observational equivalent* with respect to an environment given by a set of multiset rewrite rules Env, written as $S_A \approx_{Env} S_B$, if, given the LTS defined by the rules $S_A \cup IF \cup Env$ and $S_B \cup IF \cup Env$, there exist a relation $\mathcal{R}$ containing the initial states, such that for all states $(\mathcal{S}_A, \mathcal{S}_B) \in \mathcal{R}$ the following conditions hold.

E1. If $\mathcal{S}_A \xrightarrow[r]{l} \mathcal{S}'_A$ where r is the recipe of a rule in $Env \cup IF$ then there exist $l' \in \mathcal{F}^{\#}$ and $\mathcal{S}'_B \in \mathcal{G}^{\#}$ such that $\mathcal{S}_B \xrightarrow[r]{l'} \mathcal{S}'_B$, and $(\mathcal{S}'_A, \mathcal{S}'_B) \in \mathcal{R}$.

E2. If $\mathcal{S}_A \xrightarrow[r]{l} \mathcal{S}'_A$ where r is the recipe of a rule in S_A then there exist recipes $r_1, \ldots, r_n \in \rho$ of rules in S_B, actions $l_1, \ldots, l_n \in \mathcal{F}^{\#}$, $n \geq 0$, and $\mathcal{S}'_B \in \mathcal{G}^{\#}$ such that $\mathcal{S}_B \xrightarrow[r_1]{l_1} \ldots \xrightarrow[r_n]{l_n} \mathcal{S}'_B$, and $(\mathcal{S}'_A, \mathcal{S}'_B) \in \mathcal{R}$.

Conditions **E1.** and **E2.** must also hold in the other direction.

Example 5. *Let S_A be the multiset rewrite system from Example 1. Suppose we form multiset rewrite system S_B by taking S_A, and modifying the rule $chal^{sys}$ such that*

$$chal^{sys}\colon [\mathsf{C}(p), \mathsf{Fr}(f)] \text{-}[\mathsf{PG}]\text{-}\!> [\mathsf{Out}_{sys}(f)] \ .$$

The two sets of multiset rewrite rules S_A and S_B are not observationally equivalent. Continuing the execution e from Example 4, we obtain the password p_0 using the rule $chal$, encrypt the nonce n_0 with the password p_0 using the rule enc, and verify the encryption with the rule $sver$. The last step is possible in S_A, but not in S_B. This is because in S_A, it holds that $!\mathsf{K}(\mathsf{enc}(n_0, p_0)) \in S_{16}$, but in S_B, we have $!\mathsf{K}(\mathsf{enc}(n_0, f_0)) \in S_{16}$ instead. So, the adversary (environment) can distinguish the password p_0 from a random value f_0 by the following execution.

$$
\begin{aligned}
S_9 &\xrightarrow[out([],[z])]{\mathsf{O}} S_{10} = (S_9 \setminus \{\mathsf{Out}_{sys}(n_0)\}) \cup \{\mathsf{In}_{env}(n_0)\} \\
&\xrightarrow[recv([],[out_1([],[z])])]{} S_{11} = (S_{10} \setminus \{\mathsf{In}_{env}(n_0)\}) \cup \{!\mathsf{K}(n_0)\} \\
&\xrightarrow[fresh([\{^{f_0}/_x\}],[])]{} S_{12} = S_{11} \cup \{\mathsf{Fr}(f_0)\} \\
&\xrightarrow[chal([],r_1)]{\mathsf{PG}} S_{13} = (S_{12} \setminus \{\mathsf{C}(p_0), \mathsf{Fr}(f_0)\}) \cup \{\mathsf{Out}_{sys}(p_0)\} \\
&\xrightarrow[out([],[z])]{\mathsf{O}} S_{14} = (S_{13} \setminus \{\mathsf{Out}_{sys}(p_0)\}) \cup \{\mathsf{In}_{env}(p_0)\} \\
&\xrightarrow[recv([],[out_1([],[z])])]{} S_{15} = (S_{14} \setminus \{\mathsf{In}_{env}(p_0)\}) \cup \{!\mathsf{K}(p_0)\} \\
&\xrightarrow[enc([],[recv_1([],[out_1([],[z])]),recv_1([],[out_1([],[z])])])]{} S_{16} \\
&\xrightarrow[send([],r_2)]{} S_{17} = S_{16} \cup \{\mathsf{Out}_{env}(enc(n_0, p_0))\} \\
&\xrightarrow[in([],[send_1([],r_2)])]{\mathsf{I}} S_{18} \\
&\xrightarrow[sver([],[sreq_2([],r_0),in_1([],[send_1([],r_2)])])]{\mathsf{PL}} S_{19}
\end{aligned}
$$

$$
\begin{aligned}
S_{16} &= S_{15} \cup \{!\mathsf{K}(enc(n_0, p_0))\}, \\
S_{18} &= (S_{17} \setminus \{\mathsf{Out}_{env}(enc(n_0, p_0))\}) \cup \{\mathsf{In}_{sys}(enc(n_0, p_0))\}, \\
S_{19} &= S_{18} \setminus \{\mathsf{In}_{sys}(enc(n_0, p_0)), \mathsf{S}(p_0, a_0, b_0, n_0)\}, \\
r_1 &= [psgen_2([\{^{a_0}/_a\}, \{^{b_1}/_b\}], [fresh_1([\{^{p_0}/_x\}], [])]), \\
&\qquad fresh_1([\{^{f_0}/_x\}], [])] \\
r_2 &= [enc_1([], [recv_1([], [out_1([], [z])]), \\
&\qquad recv_1([], [out_1([], [z])])])].
\end{aligned}
$$

It holds that $trace(e') = [\mathsf{PL}, \mathsf{O}, \mathsf{I}, \mathsf{PL}, \mathsf{O}, \mathsf{PG}, \mathsf{O}, \mathsf{I}, \mathsf{PL}]$. This execution e' represents the false attack in the context of off-line guessing because one can always make a distinction by succesfully executing a session after obtaining the password. Therefore, we should forbid such executions. That means the action PL must not come after the action PG. These actions will represent the off-line guessing phases later on.

A *bi-system* [1] S is a multiset rewrite system with $\mathsf{diff}(\_,\_)$ operator such that a pair of multiset rewrite systems $L(S)$ and $R(S)$ can be obtained by considering the left hand side (LHS) and the right hand side (RHS) of $\mathsf{diff}(\_,\_)$ operator respectively. This makes it possible do verify *bi-equivalence* — a rewriting modulo diff terms. Bi-equivalence is a sound approximation of observational equivalence and it is easier to automate.

Example 6. *Consider the multiset rewrite system from Example 1. Suppose that we modify the rule $chal^{sys}$ such that*

$$chal^{sys}\colon [\mathsf{C}(p), \mathsf{Fr}(f)] \text{–}[\mathsf{PG}]\text{→}[\mathsf{Out}_{sys}(\mathsf{diff}(p, f)] \ .$$

The resulting system S is now a bi-system. Furthermore, for the systems S_A and S_B from Example 5 it holds that $S_A = L(S)$ and $S_B = R(S)$.

3) TAMARIN *prover:* In the context of observational equivalence, TAMARIN uses *restricted normal dependency graphs modulo AC* to represent protocol and adversary execution with *$*$-restricted protocol rules* and *normal (message) deduction rules*. The signature Σ_{Fact} consists of an arbitrary number of protocol-specific fact symbols used to specify protocol state, the linear facts $\mathsf{Fr}(x{:}fr)$ and $\mathsf{Frl}(x{:}fr)$ for the protocol and adversary fresh names respectively, the persistent fact $\mathsf{K}(x)$ for the adversary knowledge, and the interface facts $\mathsf{Out}(x)$ and $\mathsf{In}(x)$. It can be shown that the set of traces of dependency graphs and LTS executions coincide [18].

A $*$-restricted protocol rule is a multiset rewrite rule id: $[l]\text{–}[a]\text{→}[r]$ such that (P1) l, a, r do not contain fresh names; (P2) l does not contain Out, K and Frl facts, and reducible function symbols; (P3) r does not contain K, In, Fr and Frl facts, and multiplication symbol $*$; (P4) $vars(r) \subseteq vars(l) \cup \mathcal{V}_{pub}$. *Protocol* is a finite set of protocol rules. Protocol rules are executed by honest participants (system).

Normal message deduction rules ND represent standard Dolev-Yao adversary. They consist of deconstruction rules built from the equations and used to deduce messages sent to the environment, and construction rules built from the signature and used to construct messages sent to the system. Adversary knowledge fact K is split into $\mathsf{K}^{\downarrow}$ and $\mathsf{K}^{\uparrow}$ facts. Deconstruction rules have premises with both $\mathsf{K}^{\downarrow}$ and $\mathsf{K}^{\uparrow}$ facts, and conclusion with $\mathsf{K}^{\downarrow}$, for example $[\mathsf{K}^{\downarrow}(\mathsf{enc}(x, y)), \mathsf{K}^{\uparrow}(y)]\text{–}[]\text{→}[\mathsf{K}^{\downarrow}(y)]$. Construction rules have only $\mathsf{K}^{\uparrow}$ facts, for example $[\mathsf{K}^{\uparrow}(x), \mathsf{K}^{\uparrow}(y)]\text{–}[]\text{→}[\mathsf{K}^{\uparrow}(\mathsf{enc}(x, y)]$. Moreover, ND rules are considered modulo *AC*. Normal deduction rules supported by TAMARIN can be found in [2, Figure 7].

There is a special rule for generating Fr facts: $Fresh$: $[]\text{–}[]\text{→}[\mathsf{Fr}(x{:}fr)]$. It is the only way for the facts Fr to came into existance and no two instances of the rule give the same $\mathsf{Fr}(x{:}fr)$. We have two variants of this rule: $Fresh_{Sys}$ for system and $Fresh_{Env}$ for the environment.

We use *many-sorted* first-order logic with sorts for messages msg and timepoints $temp$. A *trace formula* is the first order formula over the trace atoms [18, p.4.]. Given a trace formula φ, $facts(\varphi)$ denotes the set of fact symbols occurring in φ; $quant(\varphi) \subseteq \{\exists, \forall\}$ denotes the set of quantifiers that occur in φ. The semantics of trace formulas is defined by assigning a domain Dom_s with each sort s: $Dom_{temp} = \mathbb{Q}$, $Dom_{fr} = FN$, $Dom_{pub} = PN$, $Dom_{msg} = \mathcal{M}$. For an equational theory $\mathcal{E}$, the definition of satisfiability of the trace formula φ on the trace tr for a valuation θ, written as $(tr, \theta) \models_{\mathcal{E}} \varphi$, can be found in [18, p.4.]. The relation $\models_{\mathcal{E}}$ is called the *satisfaction relation*.

To specify security properties we use *guarded trace properties* [18, p.12.]. For a trace formula ψ and a set of action facts A, we define *ψ formula traces* to be the set of all traces of the

system for which the formula holds $Tr_\psi = \{tr \in ginsts(A)^* \mid tr \vDash \psi\}$; we will use the name property for both the formula ψ and its sets of traces Tr_ψ. We say that $Tr_s \subseteq ginsts(A)^*$ is a *safety property* if for all $tr \in ginsts(A)^* \setminus Tr_s$ there exists finite prefix $\hat{tr}$ of tr, such that

$$Tr_s \cap \{tr' \in ginsts(A)^* \mid \hat{tr} \text{ is a finite prefix of } tr'\} = \emptyset.$$

The trace $\hat{tr}$ is called a *bad prefix* for Tr_s. Safety properties can be further characterized by the closure. The set of *finite prefixes of a trace* $tr \in ginsts(A)^*$ is defined as

$$pref(tr) = \{\hat{tr} \in ginsts(A)^* \mid \hat{tr} \text{ is a finite prefix of } tr\}.$$

For a property $Tr \subseteq ginsts(A)^*$, we define the *set of finite prefixes* of Tr as $pref(Tr) = \bigcup_{tr \in Tr} pref(tr)$; the *closure* of a property Tr as

$$closure(Tr) = \{tr \in ginsts(A)^* \mid pref(tr) \subseteq pref(Tr)\}.$$

We say that property Tr is *prefix-closed* if $closure(Tr) = Tr$. Safety properties can be characterized by their closure (the proof of this fact can be found, for example, in [17]):

Tr is a safety property iff Tr is prefix-closed.

III. Conditional Observational Equivalence

In this section we give our definition of conditional observational equivalence for systems induced with two sets of multiset rewrite rules S_A and S_B. We model conditions as sets of "allowable" traces Tr_A and Tr_B of the two respective systems. Informally, our definition says that whenever it is possible to make progress in one system while satisfying the conditions, it must also be possible to make analogous progress in the other system, again while satisfying the equivalence conditions. Similarly to the definition in [2], a transition of the environment has to be matched by the same rule, while the transition inside the system has to be matched by a sequence of transitions in the other system.

Definition 1 (Conditional Observational Equivalence)**.** *Two sets of multiset rewrite rules S_A and S_B are **conditional observational equivalent** with respect to the traces $Tr = Tr_A \cup Tr_B$, and an environment given by a set of multiset rewrite rules Env, written as $S_A \approx^{Tr}_{Env} S_B$, if, given the LTS defined by the rules $S_A \cup IF \cup Env$ and $S_B \cup IF \cup Env$, there exist a relation $\mathcal{R}$ containing the initial states, such that for all states $(\mathcal{S}_A, \mathcal{S}_B) \in \mathcal{R}$ the following conditions hold.*

- **C1.** *If there exists a trace $tr_A \in trace(exec_{S_A}(\mathcal{S}_A)) \cap Tr_A$, a recipe $r \in \rho$ of a rule in $Env \cup IF$, and a set of action facts $l \in \mathcal{F}^\#$, such that $\mathcal{S}_A \xrightarrow[r]{l} \mathcal{S}'_A$ and $concat(tr_A, [l]) \in Tr_A$, then for every trace $tr_B \in trace(exec_{S_B}(\mathcal{S}_B)) \cap Tr_B$ there exists $\mathcal{S}'_B \in \mathcal{G}^\#$, and a set of action facts $l' \in \mathcal{F}^\#$, such that $\mathcal{S}_B \xrightarrow[r]{l'} \mathcal{S}'_B$, $concat(tr_B, [l']) \in Tr_B$, and $(\mathcal{S}'_A, \mathcal{S}'_B) \in \mathcal{R}$.*
- **C2.** *If there exists a trace $tr_A \in trace(exec_{S_A}(\mathcal{S}_A)) \cap Tr_A$, a recipe $r \in \rho$ of a rule in S_A, and a set of action facts $l \in \mathcal{F}^\#$, such that $\mathcal{S}_A \xrightarrow[r]{l} \mathcal{S}'_A$ and $concat(tr_A, [l]) \in Tr_A$, then for every trace $tr_B \in trace(exec_{S_B}(\mathcal{S}_B)) \cap Tr_B$ there exists $\mathcal{S}'_B \in \mathcal{G}^\#$, a sequence of recipes $r_1, \ldots, r_n \in \rho$, and a sequence of sets of action facts $l_1, \ldots, l_n \in \mathcal{F}^\#$, $n \geq 0$, such that $\mathcal{S}_B \xrightarrow[r_1]{l_1} \ldots \xrightarrow[r_n]{l_n} \mathcal{S}'_B$, $concat(tr_B, [l_1, \ldots, l_n]) \in Tr_B$, and $(\mathcal{S}'_A, \mathcal{S}'_B) \in \mathcal{R}$.*

The conditions must also hold in the other direction.

First, note that $\mathcal{R}$ is always non-empty since $(\emptyset, \emptyset) \in \mathcal{R}$. Even though this definition makes no assumptions on the sets of allowable traces Tr_A and Tr_B, it will be useful in practice only when Tr_A and Tr_B are safety properties (prefix-closed). The reason is that we look no further than the states which can be reached from the initial state without breaking the conditions **C1.**, **C2.** and the opposite ones. For example, if there is no way for systems to take any steps (in the initial state) without violating the conditions, then the conditional observational equivalence trivially holds.

Example 7. *Consider the bi-sytem S from Example 6, and let Tr denote the set of traces such that for every trace $tr \in Tr$ it holds that the action fact* PL *always precedes the action fact* PG*. For the execution e' from Example 5 of the system $S_A = L(S)$, it holds that $trace(e') \notin Tr$ because the last fact $tr(e')_8 = \mathsf{PL}$ comes after the fact $tr(e')_6 = \mathsf{PG}$. The last step in the execution e' breaks the condition, it is a "bad step" in that sense. By abuse of notation, we denote this kind of steps with a crossed arrow:*

$$S_{18} \xrightarrow[sver([],r_4)]{\mathsf{PL}} \!\!\!\!\!\!\rightarrow_{L(S)} \; S_{19}.$$

So, just by considering the execution e', we cannot make the claim that the systems $L(S)$ and $R(S)$ are also not conditionally observational equivalent.

Suppose that we add the equation $\mathsf{dec}(\mathsf{enc}(x, y), y) = x$ *to the system, and that the rule dec^{sys} is considered modulo the equation. Continuing from the state S_{11}, we can make the following execution e'' on the LHS, which executes the rule $ures$ to get the message* $\mathsf{dec}(\mathsf{enc}(n_0, p_0), p_0)$*, obtaines the password p_0 with the rule $chal$, decrypts the message with the password to get the second nonce n_0 using the rule dec, and compares both nonces with the rule $comp$.*

$$\begin{aligned}
S_{11} &\xrightarrow[send([],[recv_1([],[out_1([],[z])])])]{} S'_{12} = S_{11} \cup \{\mathsf{Out}_{env}(n_0)\} \\
&\xrightarrow[in([],[send_1([],recv_1([],[out_1([],[z])])))]]{\mathsf{I}} S'_{13} \\
&\xrightarrow[ures([],r'_1)]{\mathsf{PL}} S'_{14} \\
&\xrightarrow[out([],[z])]{\mathsf{O}} S'_{15} \\
&\xrightarrow[recv([],[out_1([],[z])])]{} S'_{16} \\
&\xrightarrow[fresh([\{^{f_0}/_x\}],[])]{} S'_{17} = S'_{16} \cup \{\mathsf{Fr}(f_0)\} \\
&\xrightarrow[chal([],r'_2)]{\mathsf{PG}} S'_{18} = (S'_{17} \setminus \{\mathsf{C}(p_0), \mathsf{Fr}(f_0)\}) \cup \{\mathsf{Out}_{sys}(p_0)\} \\
&\xrightarrow[out([],[z])]{\mathsf{O}} S'_{19} = (S'_{18} \setminus \{\mathsf{Out}_{sys}(p_0)\}) \cup \{\mathsf{In}_{env}(p_0)\} \\
&\xrightarrow[recv([],[out_1([],[z])])]{} S'_{20} = (S'_{19} \setminus \{\mathsf{In}_{env}(p_0)\}) \cup \{!\mathsf{K}(p_0)\}
\end{aligned}$$

$$\xrightarrow[dec([],[recv_1([],[out_1([],[z])]),recv_1([],[out_1([],[z])])])]{} S'_{21}$$
$$\xrightarrow[comp([],r'_3)]{} S'_{22} = S'_{21}$$

$$S'_{13} = (S'_{12} \setminus \{\mathsf{Out}_{env}(n_0)\}) \cup \{\mathsf{In}_{sys}(n_0)\},$$
$$S'_{14} = (S'_{13} \setminus \{\mathsf{U}(p_0, a_0, b_0), \mathsf{In}_{sys}(n_0)\}) \cup \{\mathsf{Out}_{sys}(\mathsf{enc}(n_0, p_0))\},$$
$$S'_{15} = (S'_{14} \setminus \{\mathsf{Out}_{sys}(\mathsf{enc}(n_0, p_0))\}) \cup \{\mathsf{In}_{env}(\mathsf{enc}(n_0, p_0))\},$$
$$S'_{16} = (S'_{15} \setminus \{\mathsf{In}_{env}(\mathsf{enc}(n_0, p_0))\}) \cup \{!\mathsf{K}(\mathsf{enc}(n_0, p_0))\},$$
$$S'_{21} = S'_{20} \cup \{!\mathsf{K}(n_0)\},$$

$$r'_1 = [ureq_2([], [psgen_1([\{^{a_0}/_a\}, \{^{b_1}/_b\}], [fresh_1([\{^{p_0}/_x\}], [])]), in_1([], send_1([], [recv_1([], [out_1([], [z])])]))])],$$
$$r'_2 = [psgen_2([\{^{a_0}/_a\}, \{^{b_1}/_b\}], [fresh_1([\{^{p_0}/_x\}], [])]), fresh_1([\{^{f_0}/_x\}], [])]$$
$$r'_3 = [dec_1([], [recv_1([], [out_1([], z])), recv_1([], [out_1([], [z])])]), recv_1([], [out_1([], [z])])].$$

Again, the last step is not possible on the RHS since we did not get the second nonce n_0 in the state $S''_{21} = S'_{20} \cup^{\#} \{!\mathsf{K}(\mathsf{dec}(\mathsf{enc}(n_0, p_0), f_0))\}$:

$$S''_{21} \xrightarrow[comp([],r'_2))]{} \xrightarrow[R(S)]{} S''_{22}$$

But now, $trace(e'') = [\mathsf{PL}, \mathsf{O}, \mathsf{I}, \mathsf{PL}, \mathsf{I}, \mathsf{PL}, \mathsf{O}, \mathsf{PG}, \mathsf{I}] \in Tr$, and we can conclude that the $L(S)$ and $R(S)$ are not conditionally observational equivalent. The execution e'' represents the sound off-line guessing attack.

IV. Conditional Dependency Graph Equivalence

Our goal is to automate conditional observational equivalence in a manner similar to the earlier work with observational equivalence, by using a sound approximation in a form of dependency graph equivalence [2] (bi-equivalence) which is much easier to verify. All proofs can be found in the appendix.

We first introduce the notion of *partial dependency graph* by generalizing an existing definition of a dependency graph [2] such that premises without incoming edges are allowed. Hence, partial dependency graphs correspond to fragments or "suffixes" of executions that may have some open (unresolved) premises.

The motivation for partial dependency graphs comes from bi-equivalence verification: rules can have infinitely many dependency graphs that can be simulated on the other side. This would mean that the backward constraint solving algorithm that verifies bi-equivalence by naively searching for the dependency graphs would never stop since every backward "step" that satisfies the condition would lead to a premise that needs to be resolved. The question, then, is whether at some point one can stop the search and conclude that all dependency graphs of the rule have been considered? The idea is to stop the search with a set of partial dependency graphs, such that every dependency graph of the rule is an extension of some partial dependency graph in the set. Later, we will call this set a covering.

Definition 2 (Partial Dependency Graph)**.** *Let E be an equational theory, R a set of labeled multiset rewrite protocol rules, Env an environment. We say that a pair $pdg = (I, D)$ is a **partial dependency graph** (PDG) modulo E for R, if $I \in_E (ginsts(R \cup IF \cup Env))^*$, $D \subseteq \mathbb{N}^2 \times \mathbb{N}^2$ and the following holds.*

- **PDG1** *For every edge $(i, u) \rightarrowtail (j, v) \in D$, it holds that $i < j$ and $concs(I_i)_u =_E prems(I_j)_v$.*
- **PDG2** *Every premise of pdg has at most one incoming edge.*
- **PDG3** *Every linear conclusion of pdg has at most one outgoing edge.*
- **PDG4** *The* Fresh *instances are unique.*

*The set of all partial dependency graphs of rules R modulo E is denoted by $pdgraphs_E(R)$; a premise without incoming edges is called an **open premise**; the set of all open premises of a partial dependency graph pdg is denoted by $oprems(pdg)$; the **root** of a partial dependency graph $pdg = (I, D)$ is its last node: $root(pdg) = I_{max(idx(I))}$; the set of partial dependency graphs of a rule $ru \in R$ is $pdgraphs_E(ru) = \{pdg \in_E pdgraphs(R) \mid root(pdg) \in_E ginsts(ru)\}$.*

Since every dependency graph is also a partial dependency graph, all further definitions carry over including the *restricted partial dependency graph* (RPDG). We say that partial dependency graphs are equivalent if they have the same structure — that is, the nodes are instances of the same rules. Note that the equivalence of partial dependency graphs does not depend on the exact ground terms used in rule instantiations.

Definition 3 (Partial Dependency Graph Equivalence)**.** *Let R be a set of labeled multiset rewrite protocol rules and Env an environment. For a $S = pdgraphs(R \cup Env \cup IF)$, we say that the partial dependency graphs $pdg = (I, D) \in S$ and $pdg' = (I', D') \in S$, are **equivalent**, written as $pdg \simeq_S pdg'$, if $D = D'$, $|I| = |I'|$, $idx(I) = idx(I')$, and for all $i \in idx(I)$ it holds that I_i and I'_i are ground instances of the same rules.*

Obviously, the relation $\simeq$ is an *equivalence relation* on $pdgraphs(R)$, and the *equivalence class* of the partial dependency graph pdg will be denoted by $[pdg]$, while the *quotient set* will be denoted by $pdgraphs(R)/\simeq$. We will say that $[pdg] \in pdgraphs(R)/\simeq$ is a **partial dependency graph class**. The covering set that we mentioned at the beginning will consist of partial dependency graph classes.

Figure 1 shows an example of a partial dependency graph class consisting of infinitely many equivalent partial dependency graphs, one for each $n, p, f \in Dom_{fr}$, and $a, b \in Dom_{pub}$. One of its instances represents part of the execution e'' from Example 7. We see that the rule instance $sreq^5$ has the open premise $\mathsf{In}_{sys}((a, b))$ which we can easily extend so that we get the execution e''. Again, the execution trace does not violate the phase condition and corresponds to a successfull off-line guessing attack.

Using the equivalence relation $\simeq$ we now define a mirror of a partial dependency graph. Recall that a mirror of a dependency graph of one system models a matching execution of the other system.

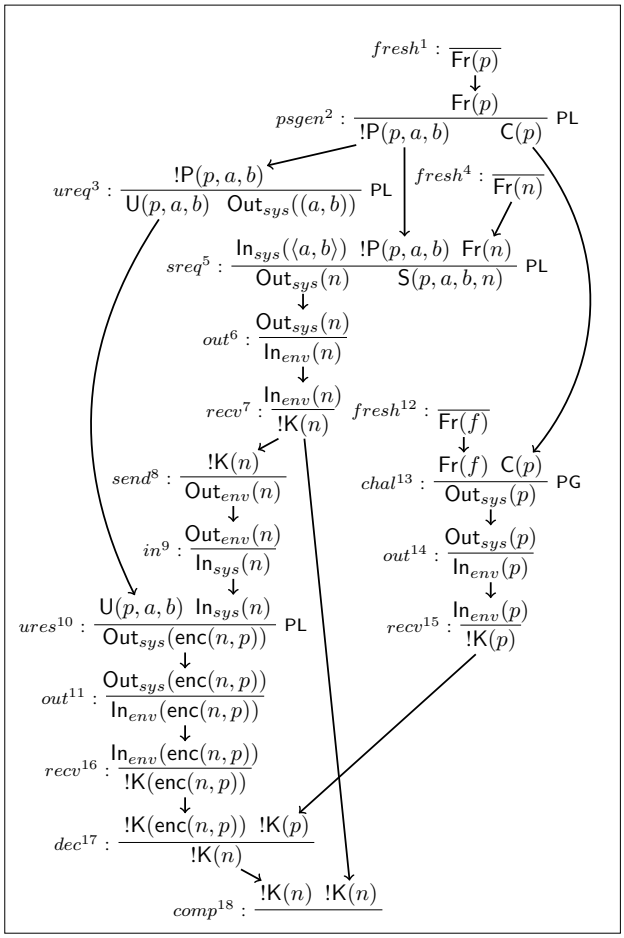

Fig. 1: Partial dependency graph class.

Definition 4 (Partial Dependency Graph Mirror)**.** *Let S be a protocol bi-system, Env an environment, $L = L(S) \cup IF \cup Env$ and $R = R(S) \cup IF \cup Env$ corresponding multiset rewrite systems, and $G = pdgraphs(L) \cup pdgraphs(R)$. **Mirror of a partial dependency graph** $pdg_L \in pdgraphs(L)$ is defined as $mirror(pdg_L) = \{pdg_R \in pdgraphs(R) \mid pdg_R \simeq_G pdg_L\}$; **mirror of a partial dependency graph class** $[pdg_L] \in pdgraphs(L)/\simeq$ is defined as $mirror([pdg_L]) = \bigcup_{pdg \in [pdg_L]} mirror(pdg)$; We define analogously in the other direction.*

Notice that, although the quotient set is defined on the union, a mirror only considers equivalent partial dependency graphs from the "other" side. This definition of a mirror is equivalent to that of Basin et al. [2] (Definition 3): the class of partial dependency graphs corresponds to the set of dependency graphs for all possible instantiations of diff-variables.

The following lemma states that the mirror of a partial dependency graph and its equivalence class coincide. This fact will be used later for proving conditional dependency graph equivalence.

Lemma 1. *Let S be a protocol bi-system, Env an environment, $L = L(S) \cup IF \cup Env$ and $R = R(S) \cup IF \cup Env$ corresponding multiset rewrite systems, and $[pdg_L] \in pdgraphs(L)/\simeq$. Then, $mirror(pdg_L) = mirror([pdg_L])$.*

We would like to extend a partial dependency graph to a possibly complete execution. For that reason, we define concatenation.

Definition 5 (Partial Dependency Graph Concatenation)**.** *For a partial dependency graph $pdg = (I, D)$, $|I| = n$, with an open premise $prems(I_i)_u \in oprems(pdg)$, and a partial dependency graph $pdg' = (I', D')$, $|I'| = m$, such that they do not share the same $Fr(x)$ instances, and $concs(I'_m)_v = prems(I_i)_u$, we define **partial dependency graph concatenation** $pdg'' = (I'', D'')$, denoted by $pdg'' = pdg' \rightarrowtail pdg$, where pdg and pdg' keep the same structure with their rule sequence numbers recalculated, such that $I'' = [I_1, \ldots, I_{i-1}, I'_1, \ldots, I'_m, I_i, \ldots, I_n]$, along with the edge between them $(m+i, v) \rightarrowtail (m+i+1, u) \in D''$. In that case, we will say that pdg' is an **partial dependency graph of an open premise** $prems(I_{m+i+1})_v$.*

For every edge of $pdg' \rightarrowtail pdg$ it holds that it is either an edge of pdg, an edge of pdg', or an edge between them, so **PDG1** is satisfied. The only incoming edge added is that of an open premise, so **PDG2** is satisfied. Since the conclusions of $root(pdg')$ have, no outgoing edges by definition, **PDG3** is satisfied. Finally, **PDG4** is required by definition.

With these definitions, we are now ready to define conditional dependency graph equivalence.

Definition 6 (Conditional Dependency Graph Equivalence)**.** *Let S be a bi-system and consider multiset rewrite systems $L = L(S) \cup IF \cup Env$ and $R = R(S) \cup IF \cup Env$. For a set of traces $Tr = Tr_A \cup Tr_B$ we say that L and R are **conditional dependency graph equivalent** written as $L(S) \sim^{Tr}_{DG,Env} R(S)$, if $dg \in dgraphs(L \cup R)$ such that $trace(dg) \in Tr$, implies $mirror(dg) \neq \emptyset$, and for all $dg' \in mirror(dg)$ it holds that $trace(dg') \in Tr$.*

The following theorem states that conditional dependency graph equivalence is a sound approximation of conditional observational equivalence.

Theorem 1. *Let S be a bi-system and $Tr = Tr_L \cup Tr_R$ be a set of traces. If $L(S) \sim^{Tr}_{DG,Env} R(S)$ then $L(S) \approx^{Tr}_{Env} R(S)$.*

We will use this result to automate verification of conditional observational equivalence for a certain kind of simple conditions.

V. Automating Conditional Observational Equivalence

Now we turn our attention to the conditions of the observational equivalence. We will consider only properties that are invariant to mirroring — that is, the properties that hold for a dependency graph if and only if they hold for its mirrors, if they exist.

Definition 7 (Trivially Mirrored Property)**.** *Let S be a bi-system, Env an environment, $L = L(S) \cup IF \cup Env$ and $R = R(S) \cup IF \cup Env$ corresponding multiset rewrite systems. We say that $Tr = Tr_A \cup Tr_B$ is a **trivially mirrored property** if for all $dg \in dgraphs(L \cup R)$, such that $trace(dg) \in Tr$ and $mirror(dg) \neq \emptyset$, implies $trace(mirror(dg)) \subseteq Tr$.*

Many useful conditions are trivially mirrored, and in general we can have an automated procedure where this assumption is discharged with a verified proof. In this paper, however, we restrict ourselves to properties for which we can conclude from the syntax of the formulas alone that they are trivially mirrored. These properties are called Type-0 properties, and they rely only on the structure of the dependency graph, and not on the values (terms) in rule instantiations. Since mirrors have the same structure, Type-0 properties will clearly be trivially mirrored.

Definition 8 (Type-0 Property)**.** *A guarded trace property ψ, such that $facts(\psi) \subseteq \mathcal{F}^0$ and $\exists \notin quant(\psi)$, is called **Type-0 property**.*

Type-0 properties suffice to express some useful conditions such as protocol phases, which are later used in the off-line guessing:

$$\forall\, \#i\, \#j.\ \mathsf{PhaseLearn}()@i \wedge \mathsf{PhaseGuess}()@j \implies \#i < \#j;$$

rules that should be executed only once:

$$\forall\, \#i\, \#j.\ \mathsf{Unique}()@i \wedge \mathsf{Unique}()@j \implies \#i = \#j;$$

or rules that should never be executed:

$$\forall\, \#i.\ \mathsf{Bad}()@i \implies \bot.$$

On the other hand, they cannot capture equality checks because values must be taken into account:

$$\forall\, x\ y\ \#i.\ \mathsf{Eq}(x, y)@i \implies x = y.$$

The following lemma and corollary state that the Type-0 properties are trivially mirrored under-approximated safety properties.

Lemma 2. *A Type-0 property is trivially mirrored.*

Corollary 1. *A Type-0 property is prefix-closed.*

Now we focus on termination in the case where unbounded number of sessions and infinite execution "loops" are possible. Informally, we can terminate when only independent open premises remain — the attacker can choose and fill in arbitrary values for these premises without affecting subsequent execution.

Given a partial dependency graph class $[pdg] = [(I, D)]$, we say that an open premise $prems([I]_i)_u = K^{\uparrow}(x)$ is a *simple premise* if $x \in Dom_{msg}$. Simple premises $prems([I]_i)_u$ and $prems([I]_j)_v$ are *dependent premises* if for every $pdg' = (I', D') \in [pdg]$ with $prems(I'_i)_u = K^{\uparrow}(s)$ and $prems(I'_j)_v = K^{\uparrow}(t)$, it holds that $i \neq j$ or $u \neq v$, and $t = s$. Otherwise, we say that they are *independent premises*. A simple premise is an *independent* if it is independent with every other open premise. A Partial dependency graph class is *trivial* if all its open premises are simple premises that are pairwise independent. Independent premise p has the property that every partial dependency graph of p has non-empty mirrors.

As mentioned at the beginning, a covering allows us to stop searching for all dependency graphs of a rule, since every dependency will be a "trivial extension" of some partial dependency graph from the covering. The definition is the following.

Definition 9 (Covering)**.** *Let P be a set of multiset rewrite protocol rules, Env an environment, and $G = P \cup IF \cup Env$. The set of restricted trivial partial dependency graph classes $Cov_S \subseteq pdgraphs(G)/\simeq$ is the **covering** for the set of dependency graphs $S \subseteq pdgraphs(G)$, if for every dependency graph $dg \in S$, there exists a partial dependency graph class $[pdg] \in C$ and the dependency graphs $dg_{p_1}, \ldots, dg_{p_n} \in dgraphs(G)$ of open premises $p_1, \ldots, p_n \in oprems(pdg)$, such that the following holds:*

$$dg = dg_{p_n} \rightarrowtail (dg_{p_{n-1}} \rightarrowtail (\cdots \rightarrowtail (dg_{p_1} \rightarrowtail pdg) \cdots)).$$

A covering is a set because there may be more then one way to get to the trivial partial dependency graph classes. For example, given the system

$$\begin{cases} ru_0: & [\mathsf{In}(\mathsf{x})]\text{–}[]\text{–}\!\!\rightarrow[\mathsf{B}(x)], \\ ru_1: & [\mathsf{B}(\mathsf{x})]\text{–}[]\text{–}\!\!\rightarrow[\mathsf{A}(x)], \\ ru_2: & [\mathsf{In}(x)]\text{–}[]\text{–}\!\!\rightarrow[\mathsf{A}(x)], \\ ru_3: & [\mathsf{A}(x)]\text{–}[]\text{–}\!\!\rightarrow[], \end{cases}$$

the covering for the set of dependency graphs of the rule ru_3 consists of two trivial partial dependency graph clases.

Finally, with a covering that has non-empty mirrors, as given in the next definition, and Type-0 properties, we can prove the conditional dependency graph equivalence.

Definition 10 (Partially Mirrored Rule)**.** *Let S be a bi-system, Env an environment, $L = L(S) \cup IF \cup Env$ and $R = R(S) \cup IF \cup Env$ corresponding multiset rewrite systems. Rule $r \in L \cup R$ is **partially mirrored** if there exists a covering $Cov_{dgraphs(r)}$ of the set of dependency graphs $dgraphs(r)$ of a rule r, such that, for every $[pdg] \in Cov_{dgraphs(r)}$, $mirror([pdg]) \neq \emptyset$ is trivial.*

Lemma 3. *Let S be a bi-system, ND a set of normal deduction rules, $L = L(S) \cup IF \cup Fresh_{sys} \cup ND$ and $R = R(S) \cup IF \cup Fresh_{sys} \cup ND$ corresponding multiset rewrite systems, and ψ a Type-0 property. If the rule $r \in L \cup R$ is partially mirrored, then $L(S) \sim^{Tr_\psi}_{RPDG} R(S)$.*

The above lemma directly translates into a termination condition when constraint solving in TAMARIN is used to verify conditional observational equivalence.

Also, it is important to note that the proof depends on the assumption that ψ is a Type-0 property, and it is not obvious how to extend the lemma (and hence the termination criteria) to arbitrary safety properties. For example, a partial dependency

graph dg with a single open premise can have completions that do and completions that do not satisfy ψ. The same is, of course, true for mirrors of dg and there is no obvious way to establish a correspondence between completions of dg that satisfy the property ψ and their mirrors.

VI. Off-line guessing

In this section, we use conditional observational equivalence to model off-line guessing resistance. Before going into technical details, we first illustrate the definition, requirements, and constraints of off-line guessing resistance in multiset rewriting.

Informally, a protocol that uses secrets (most often *weak secrets*, such as passwords) is resistant to such attacks if, after eavesdropping or participating in the protocol, the attacker cannot later use the obtained information to verify guesses (without interacting with the protocol again). More formally, off-line guessing resistance can be described as the following two-phase game. Starting from the *learning phase*, an adversary interacts with the protocol in a standard way by instantiating its rules and thereby learning the messages exchanged during protocol execution. When it reaches the *guessing phase*, the adversary can no longer interact with the protocol, but it is given a challenge: the weak secret and a fresh value. We now say that the protocol is resistant to off-line guessing if the adversary cannot distinguish the weak secret from a fresh value.

Note that this informal definition is analogous to the definition used by the ProVerif [1], which is based on the Corin et al. [19]. While ProVerif uses observational equivalence with built-in phases (stages), we use conditional observational equivalence, where the condition is the first-order formula

$$\forall\, \#i\ \#j.\ \mathsf{PhaseLearn}()@i \wedge \mathsf{PhaseGuess}()@j \implies \#i < \#j.$$

Of course, we annotate all protocol rules with the action $\mathsf{PhaseLearn}()$, and the challenge itself with the action $\mathsf{PhaseGuess}()$. Note that phases are necessary when verifying off-line guessing resistance with more than one session. Otherwise, an adversary can distinguish the secret from the fresh value by instantiating a new session of the protocol.

In order to verify off-line guessing resistance in protocols that use symmetric encryption with keys derived from weak secrets (like EAP-EKE), we need to introduce an additional primitive wsenc — weak symmetric encryption. The current symmetric encryption scheme in Tamarin is too strong — it is impossible for the adversary to apply the following deconstruction rule to decrypt a message with a wrong key

$$[\mathsf{K}^{\downarrow}(\mathsf{senc}(x,y)), \mathsf{K}^{\uparrow}(y)] \text{-}[]\!\!\rightarrow\!\![\mathsf{K}^{\downarrow}(x)]\ .$$

In a sense, symmetric encryption includes integrity protection. This gives the adversary the ability to verify guesses at secrets, given a ciphertext where those secrets are used as keys: Decryption succeeds (the rule can be applied) if the challenge is the weak secret, and fails (the rule cannot be applied) if the challenge is a fresh value.

Since we are dealing with weak secrets, the security assumptions are in a sense reversed — instead of using possibly *weak* data with a strong key, here it is necessary to protect *strong* (high entropy) data with a weak key. More precisely, data encrypted with a weak key must be indistinguishable from the random value. It imposes a constraint on what can and cannot be encrypted with a weak key; roughly speaking, we cannot use data that has a known *structure*. This is a technical problem, since we need to add reasoning rules for the kinds of payload data to be encrypted with weak keys. In this paper, we only consider Diffie-Hellman public keys $`g`^x$ as payload data and this is sufficient to analyze the EAP-EKE protocol.

To allow decryption of a message with a possibly wrong key, we add the standard symmetric encryption equation

$$\mathsf{wsdec}(\mathsf{wsenc}(x,y),y) = x \tag{1}$$

to the equational theory and extend the adversary rules with the following rule

$$[\mathsf{K}^{\downarrow}(\mathsf{wsenc}(x,y)), \mathsf{K}^{\uparrow}(z)] \text{-}[]\!\!\rightarrow\!\![\mathsf{K}^{\downarrow}(\mathsf{wsdec}(\mathsf{wsenc}(x,y),z))]$$

When this rule is applied in execution, the normalized conclusion will always be either x or $\mathsf{wsdec}(\mathsf{wsenc}(x,y),z)$ corresponding to the decryption with the correct or incorrect key. However, if the adversary now tries to encrypt both terms using the challenge, it will obtain different terms on two sides ($\mathsf{wsenc}(x,y)$ and $\mathsf{wsenc}(\mathsf{wsdec}(\mathsf{wsenc}(x,y),z),z)$) leading to a false attack. To address this we need to allow the adversary to "undo" the decryption by adding the following equation to the theory

$$\mathsf{wsenc}(\mathsf{wsdec}(x,y),y) = x. \tag{2}$$

Now, both wsenc and wsenc become reducible function symbols, and we cannot use them in rule premises (to encrypt/decrypt with pattern matching). However, we believe this is necessary to faithfully model weak symmetric encryption.

Although the equations (1) and (2) can be user-specified in Tamarin, their corresponding adversary deconstruction rules cannot be used to describe decryption with a possibly wrong key: $wsdec(wsdec(x,y),z)$; for this we need the previously specified adversary rule.

Finally, since the plaintext encrypted with weak symmetric encryption in EAP-EKE is a Diffie-Hellman public key, we need to add assumptions that such values are indistinguishable from random values (as long as the exponents are indistinguishable from random values)[1]. For that purpose we need to add adversary rules that exponentiate the terms corresponding to decryptions with wrong keys.

$$\frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(\mathsf{wsdec}(\mathsf{wsenc}(x\hat{}y,z),w)) \qquad \mathsf{K}^{\uparrow}_{\mathsf{e}}(e)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(\mathsf{wsdec}(\mathsf{wsenc}(x\hat{}y,z),w)\hat{}e)}\ .$$

[1]Note that, in practice, this assumption does not hold for all Diffie-Hellman groups.

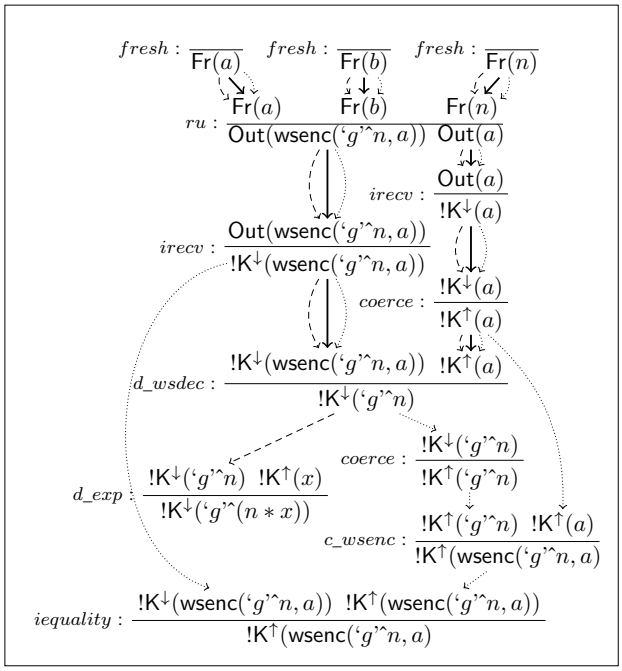

Fig. 2: Dependency graph dg_{wsenc} (⟶), partial dependency graph class $[dg_{exp}]$ (- - - -⟶), and dependency graph dg_{iequ} (⋯⋯⟶).

A. Modelling Weak Symmetric Encryption

All the equations and rules mentioned will now be introduced more formally by using the following protocol as a motivation. Starting from the existing TAMARIN's equations and rules, we iteratively verify the protocol for the off-line guessing resistance, introducing new rules and equations when necessary.

Let ND be a set of normal deduction rules [2, Figure 7], IF the interface, and $Fresh$ the fresh system rules. Consider the bi-system

$$S = \left\{ ru\colon \frac{\mathsf{Fr}(a{:}fr) \quad \mathsf{Fr}(b{:}fr) \quad \mathsf{Fr}(n{:}fr)}{\mathsf{Out}(\mathsf{wsenc}(\text{`}g\text{'}\hat{}n, a{:}fr)) \quad \mathsf{Out}(\mathsf{diff}(a{:}fr, b{:}fr))} \right\},$$

with the weak secret $a{:}fr$, where the signature $\Sigma_g = \{\mathsf{wsenc}/2, \mathsf{wsdec}/2\}$ and the set of equations $E_g = \{\mathsf{wsdec}(\mathsf{wsenc}(x,y),y) = x\}$ form the equational presentation $\mathcal{E}_g = (\Sigma_g, E_g)$. Let $\mathcal{R}_g$ be the rewrite system of E_g, and $L = L(S) \cup IF \cup Fresh \cup ND$, $R = R(S) \cup IF \cup Fresh \cup ND$ corresponding multiset rewrite systems. Both systems correspond to a simple protocol that encrypts a fresh Diffie-Hellman public key with a fresh key using weak symmetric encryption. Together with the ciphertext, the protocol outputs a challenge — encryption key on the LHS and the new fresh value on the RHS.

It does not hold that $L(S) \sim_{DG,ND} R(S)$ because the dependency graph dg_{wsenc}, in Figure 2 has no mirror. The rule d_wsdec, on the LHS, can be used to decrypt $\mathsf{wsenc}(\text{`}g\text{'}\hat{}n, a)$ since we have the key a, but no such rule can be used on RHS. This attack is sound if we assume that the adversary knows the key used to encrypt the message beforehand — it can distinguish between a and b from the start. But in off-line guessing, the attack is not sound since the adversary cannot distinguish a from b, so it must be able to decrypt the message on the RHS also. For this purpose we define the following rule.

$$d\_wsdec_g\colon \frac{\mathsf{K}^{\downarrow}_{\mathsf{e}_1}(\mathsf{wsenc}(x,y)) \qquad \mathsf{K}^{\uparrow}_{\mathsf{e}_2}(z)}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(\mathsf{wsdec}(\mathsf{wsenc}(x,y),z))},$$

where $\mathsf{e}_i \in \{\mathsf{exp}, \mathsf{noexp}\}, i \in \{1,2\}$ (see [18, p. 7]). If $y = z$ then the rule would give us $\mathsf{K}^{\downarrow}_{\mathsf{exp}}(\mathsf{wsdec}(\mathsf{wsenc}(x,y),z)) \downarrow_{\mathcal{R}_g} = \mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)$ since all terms have to be normalized; otherwise we get $\mathsf{K}^{\downarrow}_{\mathsf{exp}}(\mathsf{wsdec}(\mathsf{wsenc}(x,y),z))$.

It still does not hold that $L(S) \sim_{DG,ND} R(S)$ because the trivial partial dependency graph class $[dg_{exp}]$ in Figure 2 has no mirror. The reason is that we cannot apply the rule d_exp on the RHS since term $\mathsf{wsdec}(\mathsf{wsenc}(\text{`}g\text{'}\hat{}n, a), b)$ is not an exponentiation. In practice, this would mean that we can distinguish between the public key $\text{`}g\text{'}\hat{}n$ and a value $\mathsf{wsdec}(\mathsf{wsenc}(\text{`}g\text{'}\hat{}n, a), b)$. But, assuming we use the safe Diffie-Hellman groups [20, Section 7.1], no such distinction is possible. This brings us to the definition of the following exponentiation rule.

$$d\_exp_g\colon \frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(\mathsf{wsdec}(\mathsf{wsenc}(x\hat{}y,z),w)) \qquad \mathsf{K}^{\uparrow}_{\mathsf{e}}(e)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(\mathsf{wsdec}(\mathsf{wsenc}(x\hat{}y,z),w)\hat{}e)}.$$

Since we are using restricted normal dependency graphs, we should comment on why the condition N7) [2, Definition 5] is not violated. The condition states that for all nodes $[\mathsf{K}^{\downarrow}_{\mathsf{exp}}(s_1), \mathsf{K}^{\uparrow}_{\mathsf{e}}(t_1)] \text{-}[]\text{-}\!\!\rightarrow [\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(s_2\hat{}t_2)]$, $s_2 \in PN$ implies $inp(t_2) \not\subseteq inp(t_1)$. For $z = w$, we have the existing exponentiation rule $[\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\hat{}y), \mathsf{K}^{\uparrow}_{\mathsf{e}}(e)] \text{-}[]\text{-}\!\!\rightarrow [\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x\hat{}(y * e))]$, otherwise $\mathsf{wsdec}(\mathsf{wsenc}(x\hat{}y,z),w) \notin PN$ so the condition trivially holds.

We still have a dependency graph dg_{iequ} from the Figure 2 without any mirror. This happens because the rule c_wsenc on the RHS looks like

$$c\_wsenc_R\colon \frac{\mathsf{K}^{\uparrow}(\mathsf{wsdec}(\mathsf{wsenc}(\text{`}g\text{'}\hat{}n, a), b)) \qquad \mathsf{K}^{\uparrow}(b)}{\mathsf{K}^{\uparrow}(\mathsf{wsenc}(\mathsf{wsdec}(\mathsf{wsenc}(\text{`}g\text{'}\hat{}n, a), b), b))}.$$

Since the conclusion $\mathsf{K}^{\uparrow}(\mathsf{wsenc}(\mathsf{wsdec}(\mathsf{wsenc}(\text{`}g\text{'}\hat{}n, a), b), b))$ is already in normal form, the rule $iequality$ cannot be applied. To do so, we use the equation $\mathsf{wsenc}(\mathsf{wsdec}(x,y),y) = x$ to ensure that the same rule normalizes to $\mathsf{K}^{\uparrow}(\mathsf{wsenc}(\text{`}g\text{'}\hat{}n, a)))$. Finally, we define the set of equations

$$E'_g = \left\{ \begin{matrix} \mathsf{wsdec}(\mathsf{wsenc}(x,y),y) = x, \\ \mathsf{wsenc}(\mathsf{wsdec}(x,y),y) = x \end{matrix} \right\},$$

and call $\mathcal{E}'_g = (\Sigma_g, E'_g)$ the *weak symmetric encryption scheme*. It is obvious that the corresponding rewrite system $\mathcal{R}'_g$ is subterm-convergent. Also, to faithfully model EAP-EKE which uses encryption schemes both with and without integrity protection, we need to use both TAMARIN's "standard" and weak symmetric encryption in our formal model.

Note that our off-line guessing adversary is sound with respect to the Diffie-Hellman rules used in the EAP-EKE protocol,

since for these rules we prevented the adversary from performing false attacks. On the other hand, our off-line guessing adversary is not sound with respect to all the normal deduction rules from [2, Figure 7]. For example, if we modify the previous bi-system S to use the fact $\mathsf{Out}(\mathsf{wsenc}(\mathsf{inv}(n{:}fr), a{:}fr))$, the adversary could perform the false off-line guessing attack by first decrypting n with the weak secret a, and then using the inverse rule $[\mathsf{K}^{\downarrow}(\mathsf{inv}(n))] \dashv\!\!\!\rightarrow [\mathsf{K}^{\downarrow}(n)]$ to obtain n which is possible on the LHS but not on the RHS.

On the implementation side of TAMARIN, we use the approach analogous to the current XOR in equivalence mode [16] and generate all the mentioned rule instances at runtime.

Off-line guessing resistance can now be defined as follows.

Definition 11. *Let S be a bi-system such that for every rule $ru \in S$ holds that $\mathsf{PL}() \in set(acts(ru))$, and there exists a rule $ru' \in S$ such that $\mathsf{G}(w{:}fr) \in set(concs(ru'))$, where w is the weak secret. Given the rule*

$$Challenge\colon [\mathsf{G}(w{:}fr), \mathsf{Fr}(w'{:}fr)] \dashv\!\!\!\rightarrow [\mathsf{PG}] \dashv\!\!\!\rightarrow [\mathsf{Out}(\mathsf{diff}(w, w'))],$$

*and a system $Q = IF \cup Fresh_{sys} \cup ND' \cup \{Challenge\}$, let $L = L(S) \cup Q$ and $R = R(S) \cup Q$. We say that a bi-system S is **off-line guessing resistant** against w if $L(S) \sim^{Tr_\psi}_{RPDG} R(S)$,*

$$\psi \equiv \forall\ \#i\ \#j.\ \mathsf{PL}()@i \wedge \mathsf{PG}()@j \implies \#i < \#j.$$

The fact $\mathsf{PL}()$ denotes the learning phase, while the fact $\mathsf{PG}()$ denotes the guessing phase. The learning phase must be present in every rule except the $Challenge$ in which the guessing phase occurs. The fact G allows us to move from the learning phase to the guessing phase at any time (rule), the other direction is not allowed.

VII. CASE STUDIES

In this section, we present the results of case studies and discuss the challenges faced when attempting to analzye EAP-EKE with an unbounded number of client and server sessions. The protocols were analyzed and the proofs verified using a version of the TAMARIN prover modified to support conditional observational equivalence with Type-0 properties as described in this paper. As mentioned in the introduction, TAMARIN already supports restrictions in observational equivalence mode, but we do not use these "legacy" restrictions in the case studies.

A. Encrypted Key Exchange (EKE) off-line guessing

Encrypted Key Exchange (EKE) is a Password Authenticated Key Exchange (PAKE) protocol that combines asymmetric and symmetric cryptography to mutually authenticate two agents sharing a common weak secret — password, and derive a cryptographically strong shared secret — session key. It was initially described by Bellovin and Merritt [21]. There are several variants of the protocol, some of which have been successfully attacked [22]. Here we focus on EAP-EKE — the RFC variant of the EKE protocol [20], which is based on the original exponential exchange (Diffie-Hellman) variant (DH-EKE) and is currently considered secure. Both EAP-EKE and DH-EKE are almost equivalent, with EAP-EKE having the additional two messages for cryptographic primitive negotiation.

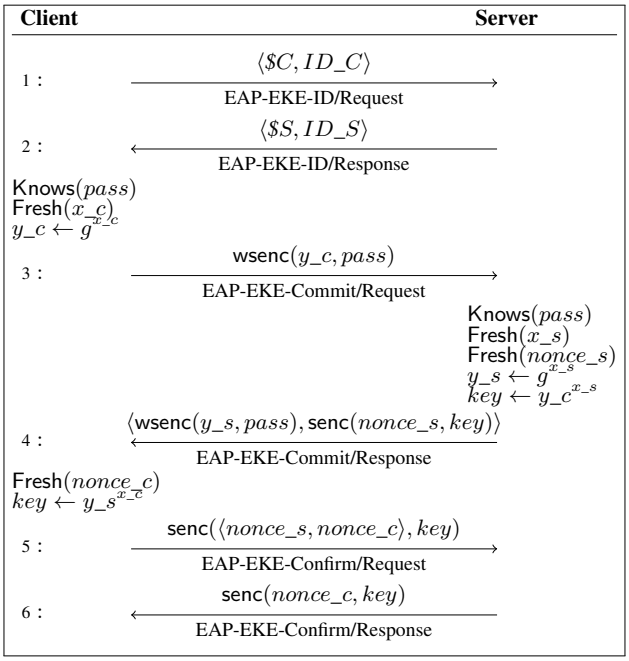

Fig. 3: EAP-EKE message exchange

The protocol is informally described in Figure 3. The EAP-EKE-ID messages are used to exchange identities; EAP-EKE-Commit to exchange ephemeral public keys y_c and y_s, where the server also sends the challenge $nonce\_s$; EAP-EKE-Confirm to complete mutual authentication using the $nonce\_s$ and $nonce\_c$, and generate the session key.

We make some simplifications compared to the RFC EAP-EKE. In particular, we do not include the entire EAP-EKE-ID payload used to negotiate the group, encryption algorithms, pseudo-random functions, and keyed message digest algorithms. Moreover, in our model, the client initiates the connection instead of the server.

We verified EAP-EKE to be off-line guessing resistant according to the Definition 11 while allowing an *unbounded number of client and two server sessions per client*; the verification took approximately eight hours on a commodity desktop computer. The setup and details are explained in the appendix.

The reason why we could not verify with an unbounded number of server sessions is the following. First, in our TAMARIN EAP-EKE model, the rule ServerCommitResponse, which is used to receive EAP-EKE-Commit/Request message, can receive anything from the environment: Since the function symbol wsenc is reducible with respect to the equation $\mathsf{wsenc}(\mathsf{wsdec}(x, y)y) = x$, and because the $*$-restricted protocol does not allow reducible function symbols in the rule premises, we must have $\mathsf{In}(client\_commit\_request)$, and then use $y\_c = \mathsf{wsdec}(client\_commit\_request, pass)$ to get the client's pub-

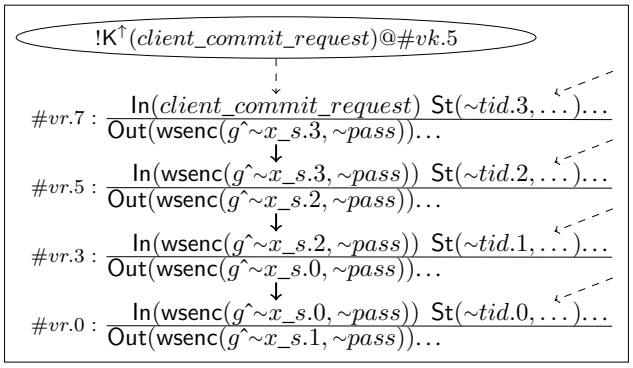

Fig. 4: Infinite regression of $ServerCommitResponse$

lic key. Second, the fact $\mathsf{In}(client\_commit\_request)$ has several variants, including the variant $\mathsf{In}(\mathsf{wsenc}(g\hat{}\sim x\_s, \sim pass))$. Since the same rule instance ServerCommitResponse also has $\mathsf{Out}(\mathsf{wsenc}(g\hat{}\sim x\_s, \sim pass))$ as a conclusion, we get an infinite regression as shown in Figure 4. Timepoints $\#vr.0$, $\#vr.3$, $\#vr.5$, $\#vr.7$ represent different instances of the same rule ServerCommitResponse, each with its own thread identifier tid.

Now, normally we would solve this kind of problem with induction:

lemma Start_Before_Loop [use_induction, diff_reuse]:
"$\forall\ x\ y\ \#j.\ \mathsf{Loop}(x,y)@j \implies (\exists\ \#i.\ \mathsf{Start}(x)@i \wedge i < j)$"

where $\mathsf{Loop}(\mathsf{x},\mathsf{y}) \in acts(\mathsf{ServerCommitResponse})$, and $\mathsf{Start}(x) \in acts(\mathsf{ClientCommitRequest})$. But in this case, the lemma does not hold because we cannot distinguish the client's EAP-EKE-Commit/Request message from a fresh value. We cannot use tagging like $\mathsf{wsenc}(\langle 1, g\hat{}\sim x\_s\rangle, \sim pass)$, since an adversary could easily distinguish an ordered pair from a fresh value. Moreover, as discussed earlier, the message cannot use any known structure whatsoever, as this would render the protocol susceptible to an off-line guessing attacks.

Note that these "looping" runs are not a feature of our model, but are in principle possible in the real deployment of the EAP-EKE protocol since the server can, in fact, not verify that the received messages decrypt to anything meaningful. This seems to be a fundamental limitation of the method of verifying observational equivalence based on coverings with partial dependency graphs — there are protocols like EAP-EKE whose dependency graphs do not admit a finite covering with partial dependency graphs satisfying the conditions of Lemma 3. As a consequence, we must use a bounded number of server sessions, in our case we limit the number of server sessions to *two per client*. This limitation is implemented in a setup rule that generates a shared password for a client and a server.

Along with the off-line guessing resistance, we also verified several trace properties in an unbounded number of sessions including the injective_agreement and perfect_forward_secrecy. These include aliveness, weak_agreement, noninjective_agreement,

B. LAK'06 and CH'07 game-based unlinkability

LAK'06 [23] is a lightweight RFID protocol with mutual authentication and tag untraceability. CH'07 [24] is RFID mutual authentication scheme that also provides tag untraceability. Both protocols were verified by TAMARIN [16] using the UK_1, UK_2 and UK_3 untraceability definitions, and their models are currently available in the TAMARIN source code repository. A limitation in verifying game-based unlinkability UK_1 was inability to model the learning and guessing phase (weak phases), and, as a consequence, these models did not consider readers during the learning phase. We take CH'07 and LAK'06 and verify UK_1 for an unbounded number of tags and readers and two pairs of tags and readers, respectively. LAK'06 took 28.2 minutes, while CH'07 took 18.9 minutes to verify.

VIII. RELATED WORK

There are many flavors of equivalence properties in the symbolic model of protocol analysis [6, 7], but we are not aware of other attempts to combine trace properties and observational equivalence properties in that model. In the computational model of cryptography, such constructions are more common since one needs indistinguishability for stating basic concepts such as key secrecy. For example, Datta et al. [25] develop a computational logic for protocols and support formulas that combine trace and indistinguishability properties. Corin and den Hartog [26] and Barthe et al. [27] develop models for reasoning about cryptographic proofs and reductions that combine reasoning about deterministic events in traces and probabilistic equivalence expressions.

Many researchers considered off-line guessing attacks in the symbolic model, and almost all of them have used some EKE variant as the main example [28, 29, 3, 19]. Lowe [28] was the first to analyze guessing attacks on protocols in [28] — he uses CSP (Communicating Sequential Processes) process algebra and the FDR model checker to analyze an asymmetric variant of EKE (PK-EKE) that uses public key encryption instead of Diffie-Hellman. FDR is constrained to bounded number of sessions and the verification was done with one initiator and responder.

Finally, the PROVERIF tool was used by Blanchet et al. [1] to verify several EKE variants, including the DH-EKE in unbounded number of client and server sessions. However, PROVERIF has rather limited support for Diffie-Hellman exponentiation with only one equation for commutative exponentiation: $\mathsf{exp}(\mathsf{exp}(g,x),y) = \mathsf{exp}(\mathsf{exp}(g,y),x)$. There is an extension with a richer support [30], but it lacks observational equivalence. It is worth mentioning that PROVERIF is orders of magnitude faster than our analysis using TAMARIN; verifying DH-EKE in less than two seconds, as opposed to nearly eight hours. Apart from mentioned PROVERIF analysis, we are not aware of any other work that verifies EKE in unbounded number of server and/or client sessions.

Besides the multiset rewriting setting, our approach is unique by using conditions to implement protocol phases — other approaches implement multi-phase experiments using a global

synchronization mechanism. Also, we are the first to verify EAP-EKE using an equational theory for the Abelian group of Diffie-Hellman exponents.

IX. Conclusions and Future Work

In this paper we have proposed combining trace and equivalence properties into *conditional observational equivalence* with the goal of being able to analyze a greater space of protocols and properties. We give the definitions in the multiset rewriting setting and extend the method of Basin et al. [2] to support conditions in observational equivalence properties. Using conditional observational equivalence, we give the first analysis of off-line guessing attacks in the multiset rewriting framework. The case study looks at the EAP-EKE password authenticated key exchange protocol and verifies off-line guessing resistance (along with a number of trace properties) in a setting with unbounded number of clients and two servers per client.

There are several limitations of our method that we wish to address in future work. First and foremost, we would like to extend the class of allowed conditions for observational equivalence. Instead of considering only Type-0 properties, we wish to allow the user to prove (using Tamarin of course) that an arbitrary property is a trivially mirrored property and use it as a condition of equivalence. We also want to extend the verification procedure to a class of safety properties that are not trivially mirrored, but this requires different approach for termination since our current method (Lemma 3 to be precise) does not easily generalize to any safety property. For example, a trivial dependency graph could easily be extended with a prefix that satisfies an equality check, while its mirror does not.

Finally, we would like to perform a more fine-grained analysis of off-line guessing resistance that it is closer to the properties required in the computational model. In particular, we wish to quantify the level of resistance and say, for example, that an active attacker can validate at most one password guess per protocol execution.

Acknowledgment

This work has been supported by the European Union through the European Regional Development Fund, under the grant KK.01.1.1.01.0009 (DATACROSS).

References

[1] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *Journal of Logic and Algebraic Programming*, vol. 75, no. 1, pp. 3–51, Feb.–Mar. 2008.

[2] D. Basin, J. Dreier, and R. Sasse, "Automated symbolic proofs of observational equivalence," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1144–1155.

[3] S. Santiago, S. Escobar, C. Meadows, and J. Meseguer, "A formal definition of protocol indistinguishability and its verification using maude-npa," in *Security and Trust Management*, S. Mauw and C. D. Jensen, Eds. Cham: Springer International Publishing, 2014, pp. 162–177.

[4] R. Chadha, Ş. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocols," in *Programming Languages and Systems*, H. Seidl, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 108–127.

[5] V. Cheval, "Apte: An algorithm for proving trace equivalence," in *Tools and Algorithms for the Construction and Analysis of Systems*, E. Ábrahám and K. Havelund, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 587–592.

[6] V. Cheval, S. Kremer, and I. Rakotonirina, *The Hitchhiker's Guide to Decidability and Complexity of Equivalence Properties in Security Protocols*. Cham: Springer International Publishing, 2020, pp. 127–145.

[7] S. Delaune and L. Hirschi, "A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols," *Journal of Logical and Algebraic Methods in Programming*, vol. 87, pp. 127–144, 2017.

[8] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 696–701.

[9] S. Delaune, S. Kremer, and M. Ryan, "Verifying privacy-type properties of electronic voting protocols," *J. Comput. Secur.*, vol. 17, no. 4, p. 435–487, Dec. 2009.

[10] V. Cortier, D. Galindo, and M. Turuani, "A formal analysis of the neuchatel e-voting protocol," in *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, 2018, pp. 430–442.

[11] M. Backes, M. Maffei, and D. Unruh, "Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol," in *IEEE Symposium on Security and Privacy, Proceedings of SSP'08*, January 2008, pp. 202–215.

[12] J. Whitefield, L. Chen, R. Sasse, S. Schneider, H. Treharne, and S. Wesemeyer, "A symbolic analysis of ecc-based direct anonymous attestation," in *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, 2019, pp. 127–141.

[13] S. Wesemeyer, C. J. Newton, H. Treharne, L. Chen, R. Sasse, and J. Whitefield, "Formal analysis and implementation of a tpm 2.0-based direct anonymous attestation scheme," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 784–798.

[14] G. Girol, L. Hirschi, R. Sasse, D. Jackson, C. Cremers, and D. Basin, "A spectral analysis of noise: A comprehensive, automated, formal analysis of diffie-hellman protocols," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1857–1874.

[15] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, "A formal analysis of 5g authentication," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1383–1396.

[16] J. Dreier, L. Hirschi, S. Radomirovic, and R. Sasse, "Automated unbounded verification of stateful cryptographic protocols with exclusive or," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, 2018, pp. 359–373.

[17] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[18] B. Schmidt, S. Meier, C. Cremers, and D. Basin, "Automated analysis of diffie-hellman protocols and advanced security properties," in *2012 IEEE 25th Computer Security Foundations Symposium*, June 2012, pp. 78–94.

[19] R. Corin, J. Doumen, and S. Etalle, "Analysing password protocol security against off-line dictionary attacks," *Electronic Notes in Theoretical Computer Science*, vol. 121, pp. 47–63, 01 2004.

[20] Y. Sheffer, G. Zorn, H. Tschofenig, and S. Fluhrer, "An eap authentication method based on the encrypted key exchange

(eke) protocol," Internet Requests for Comments, RFC Editor, RFC 6124, February 2011.
[21] S. M. Bellovin and M. Merritt, "Encrypted key exchange: password-based protocols secure against dictionary attacks," in *Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, 1992, pp. 72–84.
[22] S. Patel, "Number theoretic attacks on secure password schemes," in *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*, 1997, pp. 236–247.
[23] S. Lee, T. Asano, and K. Kim, "RFID mutual authentication scheme based on synchronized secret information," in *Symposium on Cryptography and Information Security (SCIS), Hiroshima, Japan*, January 2006.
[24] H.-Y. Chien and C.-W. Huang, "A lightweight RFID protocol using substring," in *Embedded and Ubiquitous Computing*, T.-W. Kuo, E. Sha, M. Guo, L. T. Yang, and Z. Shao, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 422–431.
[25] A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani, "Probabilistic polynomial-time semantics for a protocol security logic," in *Automata, Languages and Programming*, L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 16–29.
[26] R. Corin and J. den Hartog, "A probabilistic hoare-style logic for game-based cryptographic proofs," in *Automata, Languages and Programming*, M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 252–263.
[27] G. Barthe, M. Daubignard, B. Kapron, and Y. Lakhnech, "Computational indistinguishability logic," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 375–386.
[28] G. Lowe, "Analysing protocols subject to guessing attacks," *Journal of Computer Security*, vol. 12, 03 2003.
[29] R. Chadha, V. Cheval, c. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocols," *ACM Trans. Comput. Logic*, vol. 17, no. 4, Sep. 2016.
[30] M. Arapinis, E. Ritter, and M. D. Ryan, "StatVerif: Verification of stateful processes," in *2011 IEEE 24th Computer Security Foundations Symposium*, 2011, pp. 33–47.
[31] G. Lowe, "Casper: a compiler for the analysis of security protocols," in *Proceedings 10th Computer Security Foundations Workshop*, 1997, pp. 18–30.
[32] "Tamarin: off-line guessing implementation," https://github.com/UAlke8xBA42z5/tamarin-prover/tree/off-line-guessing, 2022.

X. Appendix

A. Lemmas and Theorems

Lemma 1. *Let S be a protocol bi-system, Env an environment, $L = L(S) \cup IF \cup Env$ and $R = R(S) \cup IF \cup Env$ corresponding multiset rewrite systems, and $[pdg_L] \in pdgraphs(L)/\simeq$. Then, $mirror(pdg_L) = mirror([pdg_L])$.*

Proof. Suppose $pdg_L^M \in mirror(pdg_L)$ which means $pdg_L^M \in \{pdg_R \in pdgraphs_E(R) \mid pdg_R \simeq pdg_L\}$. Since for every $pdg'_L \in [pdg_L]$ holds $pdg'_L \simeq pdg_L$, we get $pdg_L^M \in mirror(pdg_L) = mirror(pdg'_L)$. It follows that $pdg_L^M \in mirror([pdg_L])$.

Suppose now $pdg_L^M \in mirror([pdg_L])$ which means $pdg_L^M \in \bigcup_{pdg'_L \in [pdg_L]} mirror(pdg'_L)$. It follows that there exists $pdg'_L \in [pdg_L]$ such that $pdg_L^M \in mirror(pdg'_L)$. Now, $mirror(pdg'_L) = \{pdg_R \in pdgraphs_E(R) \mid pdg_R \simeq pdg'_L\}$, implies $pdg_L^M \simeq pdg'_L$, and since $pdg'_L \in [pdg_L]$, it holds that $pdg_L^M \simeq pdg_L$. By the assumption $pdg_L^M \in pdgraphs_R(R)$ and the fact $pdg_L^M \simeq pdg_L$, it follows $pdg_L^M \in mirror(pdg_L)$. □

Theorem 1. *Let S be a bi-system and $Tr = Tr_L \cup Tr_R$ be a set of traces. If $L(S) \sim^{Tr}_{DG,Env} R(S)$ then $L(S) \approx^{Tr}_{Env} R(S)$.*

Proof. This proof is similar to the proof of the Theorem 1 from [2]. Suppose $L(S) \sim^{Tr}_{DG,Env} R(S)$ for a multiset rewrite systems $L = L(S) \cup IF \cup Env$ and $R = R(S) \cup IF \cup Env$. We must prove that there exists a relation $\mathcal{R}$ with the initial states $(\emptyset, \emptyset) \in \mathcal{R}$, such that conditions **C1.** and **C2.** are satisfied in both directions. Let $\mathcal{R}$ be defined as

$$\begin{aligned}\mathcal{R} =& \{(\mathcal{S}_A, \mathcal{S}_B) \mid \mathcal{S}_A = state(dg_L), \mathcal{S}_B = state(dg_R),\\ & dg_R \in mirror(dg_L), dg_L \in dgraphs(L),\\ & trace(dg_L) \in Tr_L\}\\ \cup\ & \{(\mathcal{S}_A, \mathcal{S}_B) \mid \mathcal{S}_A = state(dg_L), \mathcal{S}_B = state(dg_R),\\ & dg_L \in mirror(dg_R), dg_R \in dgraphs(R),\\ & trace(dg_R) \in Tr_R\},\end{aligned}$$

and $(\mathcal{S}_A, \mathcal{S}_B) \in \mathcal{R}$.

C1. Suppose there exists a trace $tr_A \in trace(exec_{S_A}(\mathcal{S}_A)) \cap Tr_A$, a recipe $r \in \rho$ of a rule in $Env \cup IF$, and a set of action facts $l \in \mathcal{F}^{\#}$, such that $\mathcal{S}_A \xrightarrow[r]{l} \mathcal{S}'_A$ and $concat(tr_A, [l]) \in Tr_A$, where $r = recipe(ru)$ and ru: $[p] \text{-}[l] \text{-}\!\!\rightarrow [c] \in Env \cup IF$. By the definition of $\mathcal{R}$ it holds that

$$\begin{aligned}&\mathcal{S}_A = state(dg_L), \mathcal{S}_B = state(dg_R),\\ &dg_R \in mirror(dg_L), dg_L \in dgraphs(L),\\ &trace(dg_L) \in Tr_L.\end{aligned}$$

Since $\mathcal{S}_A \xrightarrow[r]{l} \mathcal{S}'_A$, there exists partial dependency graph $pdg_{ru} = ([ru\text{: } [p] \text{-}[l] \text{-}\!\!\rightarrow [c]], \emptyset) \in pdgraphs(L)$ such that $dg'_L = dg_L \rightarrowtail pdg_{ru} \in dgraphs(L)$. Because $concat(tr_A, [l]) \in Tr_A$, it follows that $trace(dg'_L) \in Tr_A$. Now, $dg'_L \in dgraphs(L)$ and $L(S) \sim^{Tr}_{DG,Env} R(S)$ together imply $mirror(dg'_L) \neq \emptyset$, and for all $dg'_R \in mirror(dg'_L)$, it holds that $trace(dg'_R) \in Tr_B$. Since $dg'_R \in mirror(dg'_L)$ implies $dg'_L \simeq dg'_R$, the right side has same structure with all possible instances, so there must exist the partial dependency graph $pdg'_{ru} = ([ru\text{: } [p'] \text{-}[l'] \text{-}\!\!\rightarrow [c']], \emptyset) \in pdgraphs(R)$ such that $dg'_R = dg_R \rightarrowtail pdg'_{ru} \in dgraphs(R)$ and $concat(tr_B, [l']) \in Tr_B$. Thus, by applying the same rule ru, we get the same recipe r. By defining $\mathcal{S}'_B = state(dg'_R)$ we have $\mathcal{S}_B \xrightarrow[r]{l'} \mathcal{S}'_B$, and with

$$\begin{aligned}&\mathcal{S}'_A = state(dg'_L), \mathcal{S}'_B = state(dg'_R),\\ &dg'_R \in mirror(dg'_L), dg'_L \in dgraphs(L),\\ &trace(dg'_L) \in Tr_L.\end{aligned}$$

it follows $(\mathcal{S}'_A, \mathcal{S}'_B) \in \mathcal{R}$. Analogously can be shown in the other direction.

C2. This proof is analogous to the **C1.** because bi-equivalence requires that every rule must be simulated by itself modulo

diff terms. The only difference is that here we consider protocol rules.

□

Lemma 2. *A Type-0 property is trivially mirrored.*

Proof. Suppose $dg \in dgraphs(L \cup R)$, $trace(dg) \in Tr$ and $mirror(dg) \neq \emptyset$. Since $facts(\psi) \subseteq \mathcal{F}^0$, fact $F \in \mathcal{F}^0$ does not contain any variables, so the diff operator cannot make any difference on the other side. It follows that for every $dg' \in mirror(dg)$, $trace(dg') \vDash \psi$, so $trace(mirror(dg)) \subseteq Tr$. □

Corollary 1. *A Type-0 property is prefix-closed.*

Proof. Let ψ be a Type-0 property. Since $\exists \notin quant(\psi)$, ψ only contains universal quantifiers, which implies that it is a safety property. □

Lemma 3. *Let S be a bi-system, ND a set of normal deduction rules, $L = L(S) \cup IF \cup Fresh_{sys} \cup ND$ and $R = R(S) \cup IF \cup Fresh_{sys} \cup ND$ corresponding multiset rewrite systems, and ψ a Type-0 property. If the rule $r \in L \cup R$ is partially mirrored, then $L(S) \sim^{Tr_\psi}_{RPDG} R(S)$.*

Proof. Suppose that $r \in L \cup R$ is a partially mirrored rule. It follows that there exist covering $Cov_{dgraphs(r)}$ of the set of dependency graphs $dgraphs(r)$ of a rule r, such that, for every $[pdg_c] \in Cov_{dgraphs(r)}$, $mirror([pdg_c]) \neq \emptyset$ is trivial. Let $dg \in dgraphs(r)$, $trace(dg) \in Tr_\psi$. Since $Cov_{dgraphs(r)}$ is the covering of $dgraphs(r)$ and $dg \in dgraphs(r)$, there exist $pdg \in [pdg_c] \in Cov_{dgraphs(r)}$ and dependency graphs $dg_{p_1}, \ldots, dg_{p_n} \in dgraphs(L \cup R)$ of open premises $p_1, \ldots, p_n \in oprems(pdg)$, such that

$$dg = dg_{p_n} \rightarrowtail (dg_{p_{n-1}} \rightarrowtail (\cdots \rightarrowtail (dg_{p_1} \rightarrowtail pdg) \cdots)).$$

Assuming that ψ is a Type-0 property, $trace(dg) \in Tr_\psi$ and Corollary 1, we have $trace(dg_{p_i}) \in Tr_\psi, i \in 1, \ldots, n$, and $trace(pdg) \in Tr_\psi$. Also, assuming that $[pdg]$ is trivial and the Lemma 2, it holds $dg^M_{p_i} = mirror(dg_{p_i}) \neq \emptyset$, $trace(mirror(dg_{p_i})) \subseteq Tr_\psi$, $i \in \{1, \ldots, n\}$. With the Lemma 1 and the assumption we have $pdg^M = mirror(pdg) = mirror([pdg]) = mirror([pdg_c]) \neq \emptyset$. This together with the Lemma 2 gives $trace(mirror(pdg)) \subseteq Tr_\psi$. Now, the following holds.

$$\begin{aligned} &mirror(dg) \\ &= mirror(dg_{p_n} \rightarrowtail (dg_{p_{n-1}} \rightarrowtail (\cdots \rightarrowtail (dg_{p_1} \rightarrowtail pdg) \cdots))) \\ &= dg^M_{p_n} \rightarrowtail (dg^M_{p_{n-1}} \rightarrowtail (\cdots \rightarrowtail (dg^M_{p_1} \rightarrowtail pdg^M) \cdots)) \\ &\neq \emptyset \end{aligned}$$

□

B. EAP-EKE off-line guessing model

Our modified TAMARIN tool is available in the Github repository [32], where models and proofs are stored under the directory `examples/csf22-conditional-equivalence`. Analysis was done on the AMD Ryzen 5 2600X (6/12 cores) @ 3.6GHz with 23GB of available RAM running on the Linux kernel version 5.11.16-arch1-1 and took almost eight hours.

As mentioned earlier, we model EAP-EKE with an unbounded number of client sessions and two server sessions per client. This is expressed with the following rule. Note unbounded number of passwords.

```
rule GeneratePassword:
  [ Fr(~pass) ]
--[]->
  [ !ClientPassword($C, $S, ~pass)
   , ServerPassword($S, $C, ~pass)
   , ServerPassword($S, $C, ~pass) ]
```

Since EAP-EKE does not use tagging, protocol messages `senc(nonce_s,key)`, `senc(<nonce_s,nonce_p>, key)` and `senc(nonce_p,key)` from the Figure 3 look very similar and can cause rules `ClientConfirmRequest` and `ServerConfirmResponse` to loop on themselves or each other. Moreover, `nonce_s` and `nonce_p` also cause partial deconstructions in the mentioned rules. We use the following lemma to resolve both of this issues.

```
lemma StartBeforeLoopNonces [sources,
    diff_reuse, reuse]:
" ( All x y #j. ClientConfirmRequestLoopNonces
    (x,y)@j ==>
    (Ex #i. ServerCommitResponseStartNonce(x)
       @i & i < j )
  )
& ( All x y #j.
    ServerConfirmResponseLoopNonces(x,y)@j ==>
    (Ex #i. ClientConfirmRequestStartNonce(x)
       @i & i < j)
  ) "
```

Following is the functional correctness lemma.

```
lemma Executable: exists-trace
" Ex C S nc ns #i #j.
   CommitClient(C, S, 'initiator', nc, ns)@i &
   CommitServer(S, C, 'responder', nc, ns)@j "
```

Appart from the restriction that enforces protocol phases:

```
restriction learnBeforeGuess:
" All #i #j. PhaseLearning()@i & PhaseGuessing
    ()@j ==> (#i < #j) "
```

we use restriction to enforce only one password guess:

```
restriction uniqueGuess:
" All #i #j. UniqueGuess()@i & UniqueGuess()@j
     ==> (#i = #j) "
```

The last one is important because it prevents a trivial false attack where the adversary satisfies `Rule_Equality` with the same password on the LHS, but no such thing is possible with the fake password on the RHS since its always new.

There are three rules in conditional observational equivalence that have no mirror: `Rule_Destrd_0_fst`, `Rule_Destrd_0_snd`, and `Rule_Destrd_0_sdec`. The analysis of these rules loops when using the default "smart" heuristic for diff proofs because of the delayed equation case splits which would otherwise lead to a contradiction. For this reason we use an oracle that puts more priority on the case splits.