Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties

Benedikt Schmidt, Simon Meier, Cas Cremers, David Basin
Institute of Information Security, ETH Zurich, Switzerland

***Abstract*—We present a general approach for the symbolic analysis of security protocols that use Diffie-Hellman exponentiation to achieve advanced security properties. We model protocols as multiset rewriting systems and security properties as first-order formulas. We analyze them using a novel constraint-solving algorithm that supports both falsification and verification, even in the presence of an unbounded number of protocol sessions. The algorithm exploits the finite variant property and builds on ideas from strand spaces and proof normal forms. We demonstrate the scope and the effectiveness of our algorithm on non-trivial case studies. For example, the algorithm successfully verifies the NAXOS protocol with respect to a symbolic version of the eCK security model.**

I. Introduction

Authenticated Key Exchange (AKE) protocols are widely used components in modern network infrastructures. They assume a Public-Key Infrastructure and use the public keys to establish shared session keys over an untrusted channel. Recent AKE protocols use Diffie-Hellman (DH) exponentiation to achieve advanced security properties, namely secrecy and authentication properties in the presence of adversaries who are significantly more powerful than the classical Dolev-Yao adversary. For example, in the eCK model [1], the adversary may corrupt random number generators and dynamically compromise long-term keys and session keys.

As witnessed by the numerous attacks on published protocols, e.g. [2]–[5], designing AKE protocols is error-prone. It is therefore desirable to formally verify them before deployment, ideally automatically and with respect to an unbounded number of sessions. In this paper, we use a symbolic model of DH exponentiation to enable automatic verification. Our model supports DH exponentiation and an abelian group of exponents. This allows the adversary to cancel out DH exponents using exponentiation with their inverse. Similar to previous work on automatic symbolic analysis [6]–[8], we do not model multiplication in the DH group and addition of exponents.

There are no existing approaches capable of automatically verifying recent AKE protocols in models combining advanced security properties, unbounded sessions, and DH exponentiation. Existing approaches either bound the number of sessions [7], [8], fail to model the required adversary capabilities [6], [9]–[11], do not consider inverses in the group of DH exponents [12]–[14], or faithfully model the adversary, but do not support DH exponentiation [15], [16]. In this paper, we give a general approach to security protocol verification, which is capable of automatically verifying AKE protocols in models as described above.

Contributions: First, we give an expressive and general security protocol model, which uses multiset rewriting to specify protocols and adversary capabilities, a guarded fragment [17] of first-order logic to specify security properties, and equational theories to model the algebraic properties of cryptographic operators.

Second, we give a novel constraint-solving algorithm for the falsification and verification of security protocols specified in our model for an unbounded number of sessions. We give a full proof of its correctness along with proofs of all theorems and assertions in this paper in the extended version [18].

Third, we implemented our algorithm in a tool, the Tamarin prover [19], and validated its effectiveness on a number of non-trivial case studies. Despite the undecidability of the verification problem, our algorithm performs well: it terminates in the vast majority of cases, and the times for falsification and verification are in the range of a few seconds. This makes Tamarin well-suited for the automated analysis of security protocols that use DH exponentiation to achieve advanced security properties.

Organization: We introduce notation in Section II and provide background on the security properties of AKE protocols in Section III. In Section IV, we define our protocol model. We present the theory underlying our constraint-solving algorithm in Section V and the algorithm in Section VI. We perform case studies in Section VII, compare with related work in Section VIII, and conclude in Section IX.

II. Notational Preliminaries

S^* denotes the set of sequences over S. For a sequence s, we write s_i for the i-th element, $|s|$ for the length of s, and $\mathit{idx}(s) = \{1, \ldots, |s|\}$ for the set of indices of s. We write $\vec{s}$ to emphasize that s is a sequence. We use $[]$ to denote the empty sequence, $[s_1, \ldots, s_k]$ to denote the sequence s where $|s| = k$, and $s \cdot s'$ to denote the concatenation of the sequences s and s'. $S^\sharp$ denotes the set of finite multisets with elements from S. We also use the superscript $\sharp$ to denote the usual operations on multisets such as $\cup^\sharp$. For a sequence s, $\mathit{mset}(s)$ denotes the corresponding multiset and $\mathit{set}(s)$ the corresponding set. We also use $\mathit{set}(m)$ for multisets m.

We write $\mathit{vars}(t)$ for the set of all variables in t, and $\mathit{fvars}(F)$ for the set of all variables that have free occurrences in a formula F. For a function f, we write $f[a \mapsto b]$ to denote the function that maps a to b and c to $f(c)$, for all $c \neq a$.

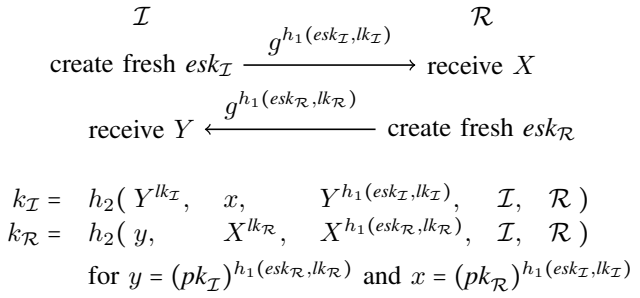

Figure 1. The NAXOS protocol.

III. Authenticated Key Exchange Protocols

We use the NAXOS protocol [1] as an example to illustrate the constructions and goals underlying recent AKE protocols. Figure 1 depicts the protocol. Each party x has a long-term private key lk_x and a corresponding public key $pk_x = g^{lk_x}$, where g is a generator of the DH group. To start a session, the initiator $\mathcal{I}$ first creates a fresh nonce $esk_{\mathcal{I}}$, also known as $\mathcal{I}$'s ephemeral (private) key. He then concatenates $esk_{\mathcal{I}}$ with $\mathcal{I}$'s long-term private key $lk_{\mathcal{I}}$, hashes the result using h_1, and sends $g^{h_1(esk_{\mathcal{I}},lk_{\mathcal{I}})}$ to the responder. The responder $\mathcal{R}$ stores the received value in a variable X, computes a similar value based on his own nonce $esk_{\mathcal{R}}$ and long-term private key $lk_{\mathcal{R}}$, and sends the result to the initiator, who stores the received value in the variable Y. Finally, both parties compute a session key ($k_{\mathcal{I}}$ and $k_{\mathcal{R}}$, respectively) whose computation includes their own long-term private keys, such that only the intended partner can compute the same key.

Note that the messages exchanged are not authenticated, as the recipients cannot verify that the expected long-term key was used in the construction of the message. The authentication is implicit and only guaranteed through ownership of the correct key. Explicit authentication (e.g., the intended partner was recently alive or agrees on some values) is commonly achieved in AKE protocols by adding a key-confirmation step, where the parties exchange a MAC of the exchanged messages that is keyed with (a variant of) the computed session key.

The key motivation behind recent AKE protocols is that they should achieve their security goals even in the presence of very strong adversaries. For example, the NAXOS protocol is designed to be secure in the eCK security model [1]. In this model, as in the standard Dolev-Yao model, the adversary has complete control over the network and can learn the long-term private keys of all dishonest agents. However, unlike in the Dolev-Yao model, he can additionally, under some restrictions, learn the long-term private key of any agent. This models (weak) *Perfect Forward Secrecy* (wPFS/PFS): even if the adversary learns the long-term private keys of all the agents, the keys of previous sessions should remain secret [20]. Additionally, this models resilience against *Key Compromise Impersonation* (KCI): even if the adversary learns the long-term private key of an agent, he should be unable to impersonate as anybody to this agent [5]. Moreover, the adversary can learn the session keys of certain sessions. This models both Key Independence (KI), where compromising one session key should not compromise other keys, and resilience against *unknown-key share attacks* (UKS), where the adversary should not be able to trick other sessions into computing the same key. Finally, the adversary can learn any agent's ephemeral keys. This models resilience against corrupted random-number generators. All these attack types are modeled in the eCK security model.

We call security properties that consider such strong adversaries *advanced security properties*. We give an example of such a property by formalizing the security of the NAXOS protocol in the eCK model in Section IV-C.

IV. Security Protocol Model

We model the execution of a security protocol in the context of an adversary as a labeled transition system, whose state consists of the adversary's knowledge, the messages on the network, information about freshly generated values, and the protocol's state. The adversary and the protocol interact by updating network messages and freshness information. Adversary capabilities and protocols are specified jointly as a set of (labeled) multiset rewriting rules. Security properties are modeled as trace properties of the transition system.

In the following, we first describe how protocols are specified and executed. Then, we define our property specification language and illustrate our protocol model with an example.

A. Protocol Specification and Execution

To model cryptographic messages, we use an order-sorted term algebra with the sort *msg* and two incomparable subsorts *fresh* and *pub* for fresh and public names. We assume there are two countably infinite sets *FN* and *PN* of *fresh* and *public names* and a countably infinite set $\mathcal{V}_s$ of variables for each sort s. We denote the union of these $\mathcal{V}_s$ by $\mathcal{V}$. We write $x{:}s$ to denote that $x \in \mathcal{V}_s$. Our approach supports a user-defined signature for modeling cryptographic operators other than DH exponentiation. To simplify its presentation, we use however a fixed signature with the function symbols

$$\Sigma_{DH} = \{\mathrm{enc}(\_,\_), \mathrm{dec}(\_,\_), \mathrm{h}(\_), \langle\_,\_\rangle, \mathrm{fst}(\_), \mathrm{snd}(\_),\\ \_\hat{}\_, \_^{-1}, \_*\_, 1\}\ ,$$

which are all of sort $msg \times \ldots \times msg \to msg$. The symbols in the first line model symmetric encryption, hashing, and pairing. Those in the second line model DH exponentiation and inversion, multiplication, and the unit in the group of exponents.

We abbreviate the set of well-sorted terms built over Σ_{DH}, *PN*, *FN*, and $\mathcal{V}$ as $\mathcal{T}$. Cryptographic *messages* are modeled by the ground terms in $\mathcal{T}$, which we abbreviate as $\mathcal{M}$. In the remainder of the paper, we use g to denote a public name that is used as a fixed generator of the DH group and $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{k}$ to denote fresh names.

(1) $\mathrm{dec}(\mathrm{enc}(m,k),k) \simeq m$ $\quad$ (6) $x * 1 \simeq x$

(2) $\mathrm{fst}(\langle x,y\rangle) \simeq x$ $\quad$ (7) $x * x^{-1} \simeq 1$

(3) $\mathrm{snd}(\langle x,y\rangle) \simeq y$ $\quad$ (8) $(x^{-1})^{-1} \simeq x$

(4) $x * (y * z) \simeq (x * y) * z$ $\quad$ (9) $(x \hat{} y) \hat{} z \simeq x \hat{} (y * z)$

(5) $x * y \simeq y * x$ $\quad$ (10) $x \hat{} 1 \simeq x$

Figure 2. Equations that constitute E_{DH}.

The equational theory E_{DH} generated by the equations in Figure 2 formalizes the semantics of the function symbols in Σ_{DH}. It consists of equations for decryption and projection (1–3), exponentiation (9–10), and the theory of abelian groups for the exponents (4–8). Equation (9) states that repeated exponentiation in a DH group corresponds to multiplication of the exponents.

As an example, consider the term $((\mathsf{g}\hat{}\mathsf{a})\hat{}\mathsf{b})\hat{}\mathsf{a}^{-1}$, which results from exponentiating g with a, followed by b, followed by a inverse. This is equal to $\mathsf{g}\hat{}((\mathsf{a}*\mathsf{b})*\mathsf{a}^{-1})$ because of (9) and can be further simplified to $\mathsf{g}\hat{}\mathsf{b}$ using (4–7).

Note that our approach supports the combination of Equations (2–10) modeling DH exponentiation and pairing with an arbitrary subterm-convergent rewriting theory for the user-defined cryptographic operators (see [18]). A rewriting theory R is subterm-convergent if it is convergent and for each rule $l \to r \in R$, r is either a proper subterm of l or is ground and in normal form with respect to R. One can therefore extend Σ_{DH} and E_{DH} with asymmetric encryption, signatures, and similar operators.

Note that our equational theory does not support protocols that perform multiplication in the DH group G. To define such protocols, an additional function symbol $\times$ denoting multiplication in G is required. The function symbol $*$ denotes multiplication in the group of exponents, which is a different operation. For example, the equality $(\mathsf{g}\hat{}\mathsf{a}\times\mathsf{g}\hat{}\mathsf{b})\hat{}\mathsf{c} = (\mathsf{g}\hat{}\mathsf{a})\hat{}\mathsf{c}\times(\mathsf{g}\hat{}\mathsf{b})\hat{}\mathsf{c}$ holds in all DH groups, but does usually not hold if we replace $\times$ by $*$. Moreover, addition of exponents must be modeled for such protocols to avoid missing attacks. Consider the example protocol that randomly choses two exponents a and b, sends these exponents, receives some exponent x, and checks if $\mathsf{g}\hat{}\mathsf{a}\times\mathsf{g}\hat{}\mathsf{b} = \mathsf{g}\hat{}\mathsf{x}$. This check succeeds if and only if $\mathsf{x} = \mathsf{a}+\mathsf{b}$.

1) Transition System State: We model the states of our transition system as finite multisets of facts. We use a fixed set of fact symbols to encode the adversary's knowledge, freshness information, and the messages on the network. The remaining fact symbols are used to represent the protocol state. Formally, we assume an unsorted signature Σ_{Fact} partitioned into *linear* and *persistent* fact symbols. We define the set of *facts* as the set $\mathcal{F}$ consisting of all facts $F(t_1,..,t_k)$ such that $t_i \in \mathcal{T}$ and $F \in \Sigma^k_{Fact}$. We denote the set of *ground facts* by $\mathcal{G}$. We say that a fact $F(t_1,..,t_k)$ is *linear* if F is linear and *persistent* if F is persistent.

Linear facts model resources that can only be consumed once, whereas persistent facts model inexhaustible resources that can be consumed arbitrarily often. In the rest of the paper, we assume that Σ_{Fact} consists of an arbitrary number of protocol-specific fact symbols to describe the protocol state and the following special fact symbols. A persistent fact $\mathsf{K}(m)$ denotes that m is known to the adversary. A linear fact $\mathsf{Out}(m)$ denotes that the protocol has sent the message m, which can be received by the adversary. A linear fact $\mathsf{In}(m)$ denotes that the adversary has sent the message m, which can be received by the protocol. A linear fact $\mathsf{Fr}(n)$ denotes that the fresh name n was freshly generated.

2) Adversary, Protocol, and Freshness Rules: To specify the possible transitions by the adversary and the honest participants, we use *labeled multiset rewriting*. A *labeled multiset rewriting rule* is a triple (l,a,r) with $l,a,r \in \mathcal{F}^*$, denoted $l\text{–}[\ a\ ]\text{→}r$. We often suppress the brackets around the sequences l, a, and r when writing rules. For $ri = l\text{–}[\ a\ ]\text{→}r$, we define the *premises* as $prems(ri) = l$, the *actions* as $acts(ri) = a$, and the *conclusions* as $concs(ri) = r$. We use $ginsts(R)$ to denote the *set of ground instances* of a set of labeled multiset rewriting rules R.

There are three types of rules. A rule for fresh name generation, the message deduction rules, and the rules specifying the protocol and the adversary's capabilities. All fresh names are created with the rule FRESH = $([]\text{–}[]\text{→}\mathsf{Fr}(x{:}\mathit{fresh}))$. This is the only rule that produces Fr facts and we consider only runs with unique instances of this rule, i.e., the same fresh name is never generated twice.

We use the following set of *message deduction rules*.

$$
\begin{aligned}
MD = \{\ & \mathsf{Out}(x)\text{–}[]\text{→}\mathsf{K}(x), \quad \mathsf{K}(x)\text{–}[\ \mathsf{K}(x)\ ]\text{→}\mathsf{In}(x) && \} \\
\cup \{\ & \text{–}[]\text{→}\mathsf{K}(x{:}\mathit{pub}), \quad \mathsf{Fr}(x{:}\mathit{fresh})\text{–}[]\text{→}\mathsf{K}(x{:}\mathit{fresh}) && \} \\
\cup \{\ & \mathsf{K}(x_1),\ldots,\mathsf{K}(x_k)\text{–}[]\text{→}\mathsf{K}(f(x_1,\ldots,x_k)) \mid f \in \Sigma^k_{DH} && \}
\end{aligned}
$$

The rules in the first line allow the adversary to receive messages from the protocol and send messages to the protocol. The $\mathsf{K}(x)$ action in the second rule makes the messages sent by the adversary observable in a protocol's trace. We exploit this to specify secrecy properties. The rules in the second line allow the adversary to learn public names and freshly generated names. The remaining rules allow the adversary to apply functions from Σ_{DH} to known messages.

A *protocol rule* is a multiset rewriting rule $l\text{–}[\ a\ ]\text{→}r$ such that (**P1**) it does not contain fresh names, (**P2**) K and Out facts do not occur in l, (**P3**) K, In, and Fr facts do not occur in r, and (**P4**) $vars(r) \subseteq vars(l) \cup \mathcal{V}_{pub}$. A *protocol* is a finite set of protocol rules. Note that our formal notion of a protocol encompasses both the rules executed by the honest participants and the adversary's capabilities, like revealing long-term keys. Condition **P1** and the restriction on the usage of Fr facts from **P3**, which also hold for the message deduction rules, ensure that all fresh names originate from instances of the FRESH rule.

3) Transition Relation: The labeled transition relation $\rightarrow_P \subseteq \mathcal{G}^\sharp \times \mathcal{P}(\mathcal{G}) \times \mathcal{G}^\sharp$ for a protocol P is defined by the transition rule

$$\frac{l\text{-}[\ a\ ]\!\rightarrow r \in_{E_{DH}} \mathit{ginsts}(P \cup \mathit{MD} \cup \{\textsc{Fresh}\}) \quad \mathit{lfacts}(l) \subseteq^\sharp S \quad \mathit{pfacts}(l) \subseteq \mathit{set}(S)}{S \xrightarrow{\mathit{set}(a)}_P ((S \setminus^\sharp \mathit{lfacts}(l)) \cup^\sharp \mathit{mset}(r))}\ ,$$

where $\mathit{lfacts}(l)$ is the multiset of all linear facts in l and $\mathit{pfacts}(l)$ is the set of all persistent facts in l. This transition rule models rewriting the state with a ground instance of a protocol rule, a message deduction rule, or the FRESH rule. Since we perform multiset rewriting modulo E_{DH}, we use $\in_{E_{DH}}$ for the rule instance. As linear facts are consumed upon rewriting, we use multiset inclusion to check that all facts in $\mathit{lfacts}(l)$ occur sufficiently many times in S. For persistent facts, we only check that every fact in $\mathit{pfacts}(l)$ occurs in S. To obtain the successor state, we remove the consumed linear facts and add the generated facts. The action associated to the transition is the set of actions of the rule instance.

A *trace* is a sequence of sets of ground facts denoting the sequence of actions that happened during a protocol's execution. We model the executions of a security protocol P by its set of traces defined as

$$\begin{aligned}\mathit{traces}(P) = \{[A_1, \ldots, A_n] \mid\ & \exists S_1, \ldots, S_n \in \mathcal{G}^\sharp.\ \varnothing^\sharp \xrightarrow{A_1}_P \ldots \xrightarrow{A_n}_P S_n \\ & \wedge \forall i \neq j.\ \forall x.\ (S_{i+1} \setminus^\sharp S_i) = \{\mathsf{Fr}(x)\}^\sharp \Rightarrow \\ & (S_{j+1} \setminus^\sharp S_j) \neq \{\mathsf{Fr}(x)\}^\sharp\ \}\ .\end{aligned}$$

The second conjunct ensures that each instance of the FRESH rule is used at most once in a trace. Each consumer of a $\mathsf{Fr}(n)$ fact therefore obtains a different fresh name. Transitions labeled with $\varnothing$ are silent. We therefore define the *observable trace* $\overline{tr}$ of a trace tr as the subsequence of all non-silent actions in tr.

B. Security Properties

We use two-sorted first-order-logic to specify security properties. This logic supports quantification over both messages and timepoints. We thus introduce the sort *temp* for timepoints and write $\mathcal{V}_{\mathit{temp}}$ for the set of *temporal variables*.

A *trace atom* is either *false* $\bot$, a *term equality* $t_1 \approx t_2$, a *timepoint ordering* $i \lessdot j$, a *timepoint equality* $i \doteq j$, or an *action* $f@i$ for a fact f and a timepoint i. A *trace formula* is a first-order formula over trace atoms.

To define the semantics of trace formulas, we associate a domain $\mathbf{D}_s$ with each sort s. The domain for temporal variables is $\mathbf{D}_{\mathit{temp}} = \mathbb{Q}$ and the domains for messages are $\mathbf{D}_{\mathit{msg}} = \mathcal{M}$, $\mathbf{D}_{\mathit{fresh}} = \mathit{FN}$, and $\mathbf{D}_{\mathit{pub}} = \mathit{PN}$. We say a function θ from $\mathcal{V}$ to $\mathbb{Q} \cup \mathcal{M}$ is a *valuation* if it respects sorts, i.e., $\theta(\mathcal{V}_s) \subseteq \mathbf{D}_s$ for all sorts s. For a term t, we write $t\theta$ for the application of the homomorphic extension of θ to t.

For an equational theory E, the *satisfaction relation* $(tr, \theta) \vDash_E \varphi$ between traces tr, valuations θ, and trace formulas φ is defined as follows.

$(tr,\theta) \vDash_E f@i$	iff $\theta(i) \in \mathit{idx}(tr)$ and $f\theta \in_E tr_{\theta(i)}$
$(tr,\theta) \vDash_E i \lessdot j$	iff $\theta(i) < \theta(j)$
$(tr,\theta) \vDash_E i \doteq j$	iff $\theta(i) = \theta(j)$
$(tr,\theta) \vDash_E t_1 \approx t_2$	iff $t_1\theta =_E t_2\theta$
$(tr,\theta) \vDash_E \neg\varphi$	iff not $(tr,\theta) \vDash_E \varphi$
$(tr,\theta) \vDash_E \varphi \wedge \psi$	iff $(tr,\theta) \vDash_E \varphi$ and $(tr,\theta) \vDash_E \psi$
$(tr,\theta) \vDash_E \exists x{:}s.\ \varphi$	iff there is $u \in \mathbf{D}_s$ such that $(tr, \theta[x \mapsto u]) \vDash_E \varphi$

The semantics of the remaining logical connectives and quantifiers are defined by translation to the given fragment as usual. Overloading notation, we write $tr \vDash_E \varphi$ if $(tr,\theta) \vDash_E \varphi$ for all θ. For a set of traces *TR*, we write $\mathit{TR} \vDash_E \varphi$ if $tr \vDash_E \varphi$ for all $tr \in \mathit{TR}$. We say that a *protocol* P *satisfies* φ, written $P \vDash_{E_{DH}} \varphi$, if $\mathit{traces}(P) \vDash_{E_{DH}} \varphi$.

C. Example: Security of NAXOS in the eCK Model

We formalize the NAXOS protocol for the eCK model using the rules in Figure 3. We include two free function symbols h_1 and h_2 in Σ_{DH}. The first rule models the generation and registration of long-term asymmetric keys. An exponent lk_A is randomly chosen and stored as the long-term key of an agent A. The persistent facts $!\mathsf{Ltk}(A, lk_A)$ and $!\mathsf{Pk}(A, \mathsf{g}^{\wedge} lk_A)$ denote the association between A and his long-term private and public keys. The public key is additionally sent to the adversary.

In the rules modeling the initiator and responder, each protocol thread chooses a unique ephemeral key esk_x, which we also use to identify the thread. The first initiator rule chooses the actor I and the intended partner R, looks up I's long-term key, and sends the half-key hk_I. The fact $\mathsf{Init}_1(\mathit{esk}_I, I, R, lk_I, hk_I)$ then stores the state of thread esk_I and the fact $!\mathsf{Ephk}(\mathit{esk}_I, \mathit{esk}_I)$ is added to allow the adversary to reveal the ephemeral key esk_I (the second argument) of the thread esk_I (the first argument). The second initiator rule reads the thread's state, looks up the public key of the intended partner, and receives the half-key Y. The key k_I is then computed. The action $\mathsf{Accept}(\mathit{esk}_I, I, R, k_I)$ denotes that the thread esk_I finished with the given parameters.

To specify when two threads are intended communication partners ("matching sessions"), we include $\mathsf{Sid}(\mathit{esk}_I, sid)$ and $\mathsf{Match}(\mathit{esk}_I, sid')$ actions. A thread s' matches a thread s if there exists a sid such that $\mathsf{Sid}(s', sid)$ and $\mathsf{Match}(s, sid)$ occur in the trace. By appropriately defining the Match actions and session identifier sid, various definitions of matching can be modeled [21].

Finally, the fact $!\mathsf{Sessk}(\mathit{esk}_I, k_I)$ is added to the second initiator rule to allow revealing the session key k_I. The responder rule works analogously. The final three rules model

Generate long-term keypair:

$\mathsf{Fr}(lk_A) -[]\rightarrow !\mathsf{Ltk}(A{:}pub, lk_A), !\mathsf{Pk}(A, \mathsf{g}\,\hat{}\,lk_A), \mathsf{Out}(\mathsf{g}\,\hat{}\,lk_A)$

Initiator step 1:

$\mathsf{Fr}(esk_I), !\mathsf{Ltk}(I, lk_I)$
$-[]\rightarrow \mathsf{Init}_1(esk_I, I, R{:}pub, lk_I, hk_I), !\mathsf{Ephk}(esk_I, esk_I), \mathsf{Out}(hk_I)$

where $hk_I = \mathsf{g}\,\hat{}\,\mathsf{h}_1(esk_I, lk_I)$

Initiator step 2:

$\mathsf{Init}_1(esk_I, I, R, lk_I, hk_I), !\mathsf{Pk}(R, pk_R), \mathsf{In}(Y)$
$-[\ \mathsf{Accept}(esk_I, I, R, k_I), \mathsf{Sid}(esk_I, \langle \mathsf{Init}, I, R, hk_I, Y\rangle)$
$,\ \mathsf{Match}(esk_I, \langle \mathsf{Resp}, R, I, hk_I, Y\rangle)\ ]\rightarrow !\mathsf{Sessk}(esk_I, k_I)$

where $k_I = \mathsf{h}_2(Y\,\hat{}\,lk_I, pk_R\,\hat{}\,\mathsf{h}_1(esk_I, lk_I), Y\,\hat{}\,\mathsf{h}_1(esk_I, lk_I), I, R)$

Responder:

$\mathsf{Fr}(esk_R), !\mathsf{Ltk}(R, lk_R), !\mathsf{Pk}(I, pk_I), \mathsf{In}(X)$
$-[\ \mathsf{Accept}(esk_R, R, I, k_R), \mathsf{Sid}(esk_R, \langle \mathsf{Resp}, R, I, X, hk_R\rangle)$
$,\ \mathsf{Match}(esk_R, \langle \mathsf{Init}, I, R, X, hk_R\rangle)\ ]\rightarrow$
$!\mathsf{Sessk}(esk_R, k_R), !\mathsf{Ephk}(esk_R, esk_R), \mathsf{Out}(\mathsf{g}\,\hat{}\,\mathsf{h}_1(esk_R, lk_R))$

where $hk_R = \mathsf{g}\,\hat{}\,\mathsf{h}_1(esk_R, lk_R)$, and
$k_R = \mathsf{h}_2(pk_I\,\hat{}\,\mathsf{h}_1(esk_R, lk_R), X\,\hat{}\,lk_R, X\,\hat{}\,\mathsf{h}_1(esk_R, lk_R), I, R)$

Key Reveals for the eCK model:

$!\mathsf{Sessk}(tid, k)$	$-[\ \mathsf{SesskRev}(tid)\ ]\rightarrow$	$\mathsf{Out}(k)$
$!\mathsf{Ltk}(A, lk_A)$	$-[\ \mathsf{LtkRev}(A)\ ]\rightarrow$	$\mathsf{Out}(lk_A)$
$!\mathsf{Ephk}(tid, esk_A)$	$-[\ \mathsf{EphkRev}(tid)\ ]\rightarrow$	$\mathsf{Out}(esk_A)$

Figure 3. Multiset rewriting rules formalizing NAXOS.

that, in the eCK model, the adversary can reveal any session, long-term, or ephemeral key. We model the restrictions on key reveals as part of the security property and thus record all key reveals in the trace.

We formalize security in the eCK model by the formula in Figure 4, which is a one-to-one mapping of the original definition of eCK security given in [1]. Intuitively, the formula states that if the adversary knows the session key of a thread esk_I, then he must have performed forbidden key reveals. The left-hand side of the implication states that the key k is known and the right-hand side disjunction states the restrictions on key reveals. We describe each disjunct in the comment above it. Further motivation and variants of these restrictions can be found in [1], [21]. Note that the eCK model formalizes weak Perfect Forward Secrecy (weak PFS), as it only allows for a long-term key reveal of the intended partner if there is a matching session. To obtain a variant of eCK formalizing PFS, we can replace the last line with

$$\vee\ (\exists i_5.\ \mathsf{LtkRev}(B)@i_5 \wedge i_5 \ll i_1))\Big)\ \Big).$$

This allows the adversary to reveal the long-term key of the intended partner after the test thread is finished or if there is a matching session. The NAXOS protocol does not satisfy this property, as reported in Table I in SectionVII.

Note that some protocols require modeling inequality conditions, e.g., the TS1-2004 [22] protocol, which assumes that an agent never executes a session with himself. We

$\forall i_1\, i_2\, s\, A\, B\, k.\ (\mathsf{Accept}(s, A, B, k)@i_1 \wedge \mathsf{K}(k)@i_2) \Rightarrow$

// If the session key of the test thread s is known, then
// s must be "not clean". Hence either there is a
// session key reveal for s,

$\Big((\exists i_3.\ \mathsf{SesskRev}(s)@i_3)$

// or a session key reveal for a matching session,

$\vee\ (\exists s'\, i_3\, i_4\, sid.\ (\mathsf{Sid}(s', sid)@i_3 \wedge \mathsf{Match}(s, sid)@i_4)$
$\wedge\ (\exists i_5.\ \mathsf{SesskRev}(s')@i_5))$

// or if a matching session exists,

$\vee\ \Big(\exists s'\, i_3\, i_4\, sid.\ (\mathsf{Sid}(s', sid)@i_3 \wedge \mathsf{Match}(s, sid)@i_4)$

// both lk_A and esk_A, or both lk_B and esk_B are revealed,

$\wedge\ \big((\exists i_5\, i_6.\ \mathsf{LtkRev}(A)@i_5 \wedge \mathsf{EphkRev}(s)@i_6)$
$\vee\ (\exists i_5\, i_6.\ \mathsf{LtkRev}(B)@i_5 \wedge \mathsf{EphkRev}(s')@i_6)\big)\Big)$

// or if no matching session exists,

$\vee\ \Big(\neg\big(\exists s'\, i_3\, i_4\, sid.\ (\mathsf{Sid}(s', sid)@i_3 \wedge \mathsf{Match}(s, sid)@i_4)\big)$

// either both lk_A and esk_A, or lk_B are revealed.

$\wedge\ \big((\exists i_5\, i_6.\ \mathsf{LtkRev}(A)@i_5 \wedge \mathsf{EphkRev}(s)@i_6)$
$\vee\ (\exists i_5.\ \mathsf{LtkRev}(B)@i_5)\big)\Big)\ \Big)$

Figure 4. eCK security definition.

model inequality conditions in two steps. First, we include $\mathsf{Neq}(s, t)$ facts in the actions of rules that require the terms s and t to be unequal. Second, we replace the considered security property φ with $(\neg(\exists i\, x.\ \mathsf{Neq}(x, x)@i)) \Rightarrow \varphi$ to restrict the analysis to traces where all inequality conditions hold. This filtering construction also works for enforcing other restrictions on traces, e.g., the uniqueness of certain actions.

V. Normal Dependency Graphs

For symbolic attack-search algorithms, there are several drawbacks to the multiset rewriting semantics given in the previous section. First, incrementally constructing attacks is difficult with (action-)traces, as they contain neither the history of past states nor the causal dependencies between steps. Second, symbolic reasoning modulo E_{DH} is difficult because E_{DH} contains cancellation equations. For example, if the adversary knows $t = na * x$ for a nonce *na*, we cannot conclude that *na* has been used in the construction of t, as x could be equal to na^{-1}. Third, the message deduction rules allow for redundant steps such as first encrypting a cleartext and then decrypting the resulting ciphertext. For search algorithms, it is useful to impose normal-form conditions on message deduction to avoid exploring such redundant steps.

We take the following approach. First, we define dependency graphs. They consist of the sequence of rewriting rule instances corresponding to a protocol execution and their causal dependencies, similar to strand spaces [23]. Afterwards, we show that we can use dependency graphs modulo AC, an equational theory without cancellation equations, instead of dependency graphs modulo E_{DH}.

Finally, we define normal message deductions and the corresponding normal dependency graphs. We also show that normal dependency graphs are weakly trace equivalent to the multiset rewriting semantics.

A. Dependency Graphs

We use dependency graphs to represent protocol executions together with their causal dependencies. A dependency graph consists of nodes labeled with rule instances and dependencies between the nodes. We first present an example of a dependency graph and then give its formal definition.

Example 1 (Dependency Graph)**.** Consider the protocol

$$P = \{\ [\mathsf{Fr}(x), \mathsf{Fr}(k)] -[]\rightarrow [\mathsf{St}(x,k), \mathsf{Out}(\mathsf{enc}(x,k)), \mathsf{Key}(k)]$$
$$,\ [\mathsf{St}(x,k), \mathsf{In}(\langle x, x\rangle)] -[\ \mathsf{Fin}(x,k)\ ]\rightarrow []$$
$$,\ [\mathsf{Key}(k)] -[\ \mathsf{Rev}(k)\ ]\rightarrow [\mathsf{Out}(k)]\ \}\ .$$

Figure 5 shows a dependency graph for an execution of P. We use inference rule notation with the actions on the right for rule instances. Nodes 1 and 2 are rule instances that create fresh names. Node 3 is an instance of the first protocol rule. Node 4 is an instance of the key reveal rule. Nodes 5–9 are instances of message deduction rules and denote that the adversary receives a ciphertext and its key, decrypts the ciphertext, pairs the resulting cleartext with itself, and sends the result to an instance of the second protocol rule, Node 10. The edges denote causal dependencies: an edge from a conclusion of node i to a premise of node j denotes that the corresponding fact is generated by i and consumed by j. Since this is a dependency graph modulo E_{DH}, it is sufficient that each pair of generated and consumed facts is equal modulo E_{DH}.

Formally, let E be an equational theory and R be a set of multiset rewriting rules. We say that $dg = (I, D)$ is a *dependency graph modulo E for R* if $I \in (\mathit{ginsts}(R \cup \{\text{FRESH}\}))^*$, $D \subseteq \mathbb{N}^2 \times \mathbb{N}^2$, and dg satisfies the conditions **DG1–4** listed below. To state these conditions, we introduce the following definitions. We call $\mathit{idx}(I)$ the *nodes* and D the *edges* of dg. We write $(i,u) \nrightarrow (j,v)$ for the edge $((i,u),(j,v))$. Let $I = [l_1 -[\ a_1\ ]\rightarrow r_1, \ldots, l_n -[\ a_n\ ]\rightarrow r_n]$. The *trace* of dg is $\mathit{trace}(dg) = [\mathit{set}(a_1), \ldots, \mathit{set}(a_n)]$. A *conclusion* of dg is a pair (i,u) such that i is a node of dg and $u \in \mathit{idx}(r_i)$. The corresponding *conclusion fact* is $(r_i)_u$. A *premise* of dg is a pair (i,u) such that i is a node of dg and $u \in \mathit{idx}(l_i)$. The corresponding *premise fact* is $(l_i)_u$. A conclusion or premise is *linear* if its fact is linear.

- **DG1** For every edge $(i,u) \nrightarrow (j,v) \in D$, it holds that $i < j$ and the conclusion fact of (i,u) is equal modulo E to the premise fact of (j,v).
- **DG2** Every premise of dg has exactly one incoming edge.
- **DG3** Every linear conclusion of dg has at most one outgoing edge.
- **DG4** The FRESH rule instances in I are unique.

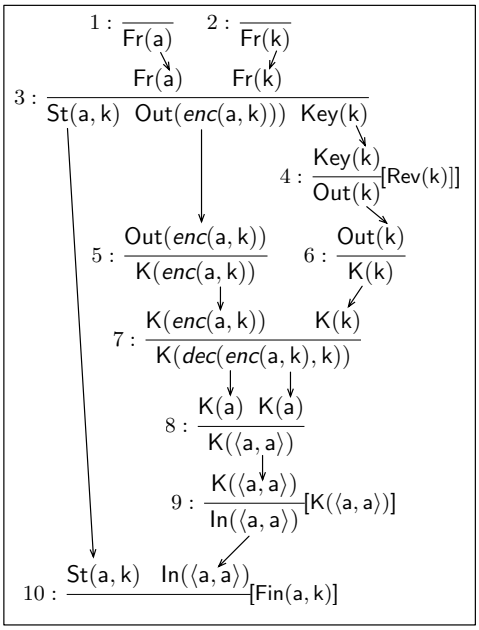

Figure 5. Dependency graph modulo E_{DH}.

We denote the set of all dependency graphs modulo E for R by $\mathit{dgraphs}_E(R)$.

Note that, for all protocols P, the multiset rewriting semantics given in Section IV and the dependency graphs modulo E_{DH} for $P \cup \mathit{MD}$ have the same set of traces, i.e., $\mathit{traces}(P) =_{E_{DH}} \{\mathit{trace}(dg) \mid dg \in \mathit{dgraphs}_{E_{DH}}(P \cup \mathit{MD})\}$.

B. Dependency Graphs modulo AC

We now switch to a semantics based on dependency graphs modulo AC. We use standard notions from order-sorted rewriting [24] and proceed in two steps.

First, we define AC as the equational theory generated by Equations (4–5) from Figure 2 and DH as the rewriting system obtained by orienting Equations (1–3,9–10) from Figure 2 and all equations from Figure 6 from left to right. $DH \uplus AC$ is an equational presentation of E_{DH} and DH is AC-convergent and AC-coherent. We can therefore define $t{\downarrow_{DH}}$ as the normal form of t with respect to DH,AC-rewriting and have $t =_{E_{DH}} s$ iff $t{\downarrow_{DH}} =_{AC} s{\downarrow_{DH}}$. We say that t is $\downarrow_{DH}$-normal if $t =_{AC} t{\downarrow_{DH}}$. We say a dependency graph $dg = (I, D)$ is $\downarrow_{DH}$-normal if all rule instances in I are $\downarrow_{DH}$-normal.

Second, E_{DH} has the finite variant property [25] for this presentation, which allows us to perform symbolic reasoning about normalization. More precisely, for all terms t, there is a finite set of substitutions $\{\tau_1, \ldots, \tau_k\}$ such that for all substitutions σ, there is an $i \in \{1, \ldots, k\}$ and a substitution σ' with $(t\sigma){\downarrow_{DH}} =_{AC} ((t\tau_i){\downarrow_{DH}})\sigma'$ and $(x\sigma){\downarrow_{DH}} =_{AC} x\tau_i\sigma'$ for all $x \in \mathit{vars}(t)$. We call $\{(t\tau_i{\downarrow_{DH}}, \tau_i) \mid 1 \le i \le k\}$ a *complete set of DH,AC-variants of t*. For a given term t, we use folding variant narrowing [24] to compute such a set,

$(1)\ (x^{-1}*y)^{-1} \simeq x*y^{-1}$ $(6)\ 1^{-1} \simeq 1$

$(2)\ x^{-1}*y^{-1} \simeq (x*y)^{-1}$ $(7)\ x*1 \simeq x$

$(3)\ x*(x*y)^{-1} \simeq y^{-1}$ $(8)\ (x^{-1})^{-1} \simeq x$

$(4)\ x^{-1}*(y^{-1}*z) \simeq (x*y)^{-1}*z$ $(9)\ x*(x^{-1}*y) \simeq y$

$(5)\ (x*y)^{-1}*(y*z) \simeq x^{-1}*z$ $(10)\ x*x^{-1} \simeq 1$

Figure 6. Lankford's presentation of the abelian group axioms

which we denote by $\lceil t \rceil^{DH}$. Overloading notation, we also denote $\{s \mid (s,\tau) \in \lceil t \rceil^{DH}\}$ by $\lceil t \rceil^{DH}$.

It is straightforward to extend these notions to multiset rewriting rules by considering rules as terms and the required new function symbols as free. We can then show that $\mathit{dgraphs}_{E_{DH}}(R)\!\downarrow_{DH} \subseteq_{AC} \mathit{dgraphs}_{AC}(\lceil R \rceil^{DH})$ for all sets of multiset rewriting rules R. If we restrict the right hand side to $\downarrow_{DH}$-normal dependency graphs, then the two sets are equal modulo AC.

Example 2. To normalize the graph $dg = (I, D)$ in Figure 5 with respect to $\downarrow_{DH}$, it suffices to replace I_7 with $ri = \mathsf{K}(\mathrm{enc}(\mathsf{a},\mathsf{k})), \mathsf{K}(\mathsf{k}) \mathrel{-[]\!\!\rightarrow} \mathsf{K}(\mathsf{a})$, calling the result dg'. Since ri is not an instance of the decryption rule $rdec = \mathsf{K}(x), \mathsf{K}(y) \mathrel{-[]\!\!\rightarrow} \mathsf{K}(\mathrm{dec}(x,y))$ or any other rule in $P \cup \mathit{MD}$, dg' is not in $\mathit{dgraphs}_{AC}(P \cup \mathit{MD})$. However, ri is an instance of $\mathsf{K}(\mathrm{enc}(x,y)), \mathsf{K}(y) \mathrel{-[]\!\!\rightarrow} \mathsf{K}(x)$, which is a DH,AC-variant of $rdec$ and therefore in $\lceil \mathit{MD} \rceil^{DH}$. Hence, $dg' \in \mathit{dgraphs}_{AC}(\lceil P \cup \mathit{MD} \rceil^{DH})$.

C. Normal Dependency Graphs

We first define the class of $*$-restricted protocols, which do not multiply exponents. We then define rules for normal message deduction and the corresponding normal dependency graphs, which are weakly trace equivalent to the multiset rewriting semantics for $*$-restricted protocols.

1) $*$-restricted Protocols: The following restriction ensures that protocols do not multiply exponents and do not introduce products by other means. A protocol P is *$*$-restricted* if, for each of its rules $l \mathrel{-[\,a\,]\!\!\rightarrow} r$, (a) l does not contain the function symbols $*$, $\hat{}$, $^{-1}$, fst, snd, and dec, and (b) r does not contain the function symbol $*$.

In general, condition (a) prevents protocol rules from pattern matching on reducible function symbols. Condition (b) prevents protocols from directly using multiplication, although repeated exponentiation is still allowed. Note that these restrictions are similar to those of previous work such as [6], [7] and are not a restriction in practice. Protocols that use multiplication in the group of exponents can usually be specified by using repeated exponentiation. Moreover, protocols that use multiplication in the DH group, such as MQV [26], cannot be specified anyway since $*$ denotes multiplication in the group of exponents.

For $*$-restricted protocols, products that occur in positions that can be extracted by the adversary can always be constructed by the adversary himself from their components.

2) Normal Message Deduction: Message deduction steps in dependency graphs modulo AC use rules from $\lceil \mathit{MD} \rceil^{DH}$. These rules still allow redundant steps. We now eliminate some of them by tagging the rules to limit their applicability.

We first partition $\lceil \mathit{MD} \rceil^{DH}$ into five subsets: communication rules for sending and receiving messages, multiplication rules consisting of all DH,AC-variants of the rule for multiplication, construction rules that apply a function symbol to arguments, deconstruction rules that extract a subterm from an argument, and the remaining exponentiation rules, which are all DH,AC-variants of the rule for exponentiation and are neither construction nor deconstruction rules.

We use tags to forbid two types of redundancies. First, we forbid using a deconstruction rule to deconstruct the result of a construction rule. This is analogous to restrictions for normal natural deduction proofs, adapted to the setting of message deduction [27]. Second, we forbid repeated exponentiation, which can always be replaced by a single exponentiation with the product of all exponents. In particular, we use the *deconstruction tags* $\downarrow$ and $\uparrow$ and the *exponentiation tags* exp and noexp. We use $\downarrow$ to tag K-facts where deconstruction is allowed and $\uparrow$ where deconstruction is forbidden. We use exp to tag K-facts that can be used as the base of an exponentiation and noexp where this is forbidden.

We obtain the *normal message deduction rules ND* shown in Figure 9 as follows. First, we add the CoerceRule rule to switch from message deconstruction to message construction, preserving the exponentiation tag. Second, we replace the multiplication rules by l-ary construction rules for multiplication. For $*$-restricted protocols, these rules are sufficient to reason about products. Third, we use exponentiation tags as follows. The construction rule for exponentiation, the deconstruction rules for exponentiation, and the exponentiation rules use exp for the first premise (the base), a variable for the second premise, and noexp for the conclusion. The remaining rules use variables for the premises and exp for the conclusion. Finally, we use deconstruction tags as follows. We use $\uparrow$ for the conclusion of construction rules and $\downarrow$ for the first premise of deconstruction and exponentiation rules. This ensures that a deconstruction or exponentiation rule can never use the conclusion of a construction rule as its first premise. For the remaining premises of rules, we use $\uparrow$. For the remaining conclusions of rules, we use $\downarrow$. Note that all conclusions of exponentiation rules, including the ones that we do not show, are of the form $\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(t \hat{} s)$ and can therefore only be used by CoerceRule.

Example 3. Figure 7 shows five message deduction subgraphs. In (a), the adversary decrypts a message that he earlier encrypted himself. Instead of performing these deductions, the adversary can directly use the conclusion $\mathsf{K}(\mathsf{a})$ that is used by the encryption. The deduction from (a) is not possible with the normal message deduction rules *ND* because the

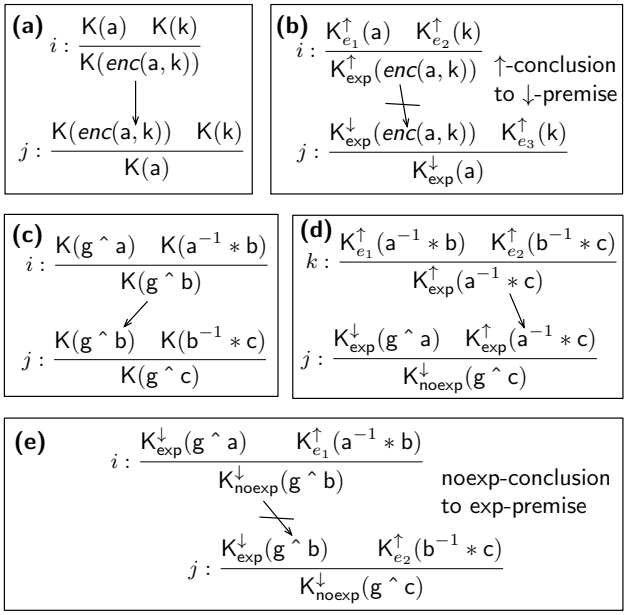

Figure 7. Message deduction subgraphs for encryption. We use $\nrightarrow$ for edges that are invalid because the source and target are not equal. We use $i, j, k \in \mathbb{N}$ and the exponentiation tags e_1, e_2, and e_3.

↑-tags and ↓-tags prevent applying a deconstruction rule to the conclusion of a construction rule, as depicted in (b). In (c), the adversary performs a redundant step that involves repeated exponentiation. Note that an unbounded number of steps that add a new exponent and remove the previously added exponent can be inserted inbetween the two rules. This deduction is impossible with the normal message deduction rules because a conclusion with a noexp-tag cannot be used with a premise that requires an exp-tag, as depicted in (e). We can replace the repeated exponentiation with one multiplication and one exponentiation as depicted in (d).

3) Normal Dependency Graphs: We now define normal dependency graphs. They use the normal message deduction rules and enforce further normal-form conditions. To state the conditions, we define the *input components of a term* t as $inp(t)$, such that $inp(t^{-1}) = inp(t)$, $inp(\langle t_1, t_2\rangle) = inp(t_1) \cup inp(t_2)$, $inp(t_1 * t_2) = inp(t_1) \cup inp(t_2)$, and $inp(t) = \{t\}$ otherwise. Intuitively, $inp(t)$ consists of the maximal subterms of t that are not products, pairs, or inverses.

Formally, a *normal dependency graph for a protocol* P is a dependency graph dg such that $dg \in dgraphs_{AC}(\lceil P \rceil^{DH} \cup ND)$ and the following conditions are satisfied.

N1 The dependency graph dg is $\downarrow_{DH}$-normal.
N2 No instance of CO ERCE deduces a pair or an inverse.
N3 There is no multiplication rule that has a premise fact of the form $\mathsf{K}^{\uparrow}_{e}(t * s)$.
N4 All conclusion facts $\mathsf{K}^{d}_{e}(t * s)$ are conclusions of a multiplication rule.
N5 If there are two conclusions c and c' with conclusion facts $\mathsf{K}^{d}_{e}(m)$ and $\mathsf{K}^{d}_{e'}(m')$ such that $m =_{AC} m'$, then $c = c'$.

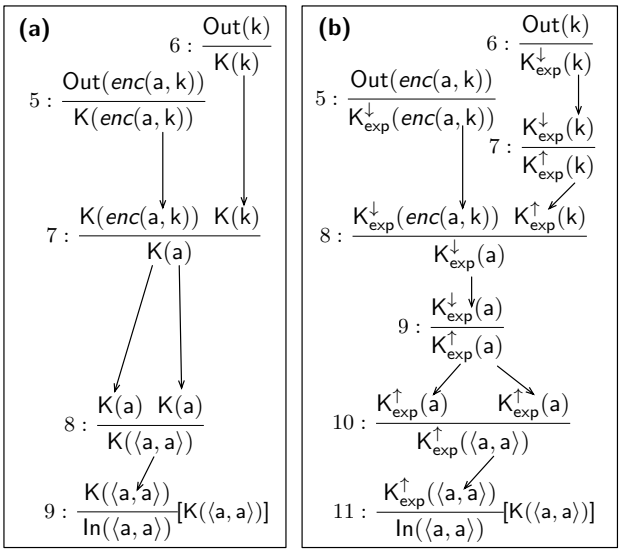

Figure 8. Examples of message deduction subgraphs of (a) a dependency graph modulo AC and (b) a normal dependency graph.

N6 If there is a conclusion $(i, 1)$ with fact $\mathsf{K}^{\downarrow}_{e}(m)$ and a conclusion $(j, 1)$ with fact $\mathsf{K}^{\uparrow}_{e'}(m')$ such that $m =_{AC} m'$, then $i < j$ and j is an instance of CO ERCE or the construction rule for pairing or the one for inversion.
N7 For all nodes $\mathsf{K}^{\downarrow}_{\mathsf{exp}}(s_1), \mathsf{K}^{\uparrow}_{e}(t_1) \dashrightarrow \mathsf{K}^{\downarrow}_{\mathsf{noexp}}(s_2 \,\hat{}\, t_2)$ such that s_2 is of sort *pub*, $inp(t_2) \notin inp(t_1)$.

We denote the set of all normal dependency graphs of P by $ndgraphs(P)$.

N1 ensures that all rule instances are $\downarrow_{DH}$-normal. **N2** ensures that pairs and inverses are always completely deconstructed. **N3** and **N4** formalize that the adversary constructs all products directly by multiplying their components. This does not restrict the adversary since we limit ourselves to $*$-restricted protocols. **N5** and **N6** ensure a restricted form of message uniqueness. **N7** forbids instances of exponentiation rules that can be replaced by instances of the construction rule for exponentiation where the base is a public name.

Note that normal dependency graphs allow exactly the same executions as our multiset rewriting semantics.

Lemma 1. *For all $*$-restricted protocols P,*

$$\overline{traces}(P)\downarrow_{DH} =_{AC} \{\overline{trace}(dg) \mid dg \in ndgraphs(P)\} .$$

Example 4. Figure 8 shows two message deduction subgraphs. Subfigure (a) shows the message deduction subgraph of a dependency graph modulo AC and (b) shows the message deduction subgraph of the corresponding normal dependency graph. We obtain (b) from (a) by renumbering the nodes and adding the required tags and CO ERCE nodes.

Example 5. Consider the exponentiation-construction node $i : \mathsf{K}^{\downarrow}_{\mathsf{exp}}(\mathsf{g}\hat{}\mathsf{a}), \mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathsf{a}^{-1} * \mathsf{b}) \dashrightarrow [] \dashrightarrow \mathsf{K}^{\downarrow}_{\mathsf{noexp}}(\mathsf{g}\hat{}\mathsf{b})$. The conclusion of i either has no outgoing edge or a single edge to a CO ERCE node j. In the first case, the node i can be removed.

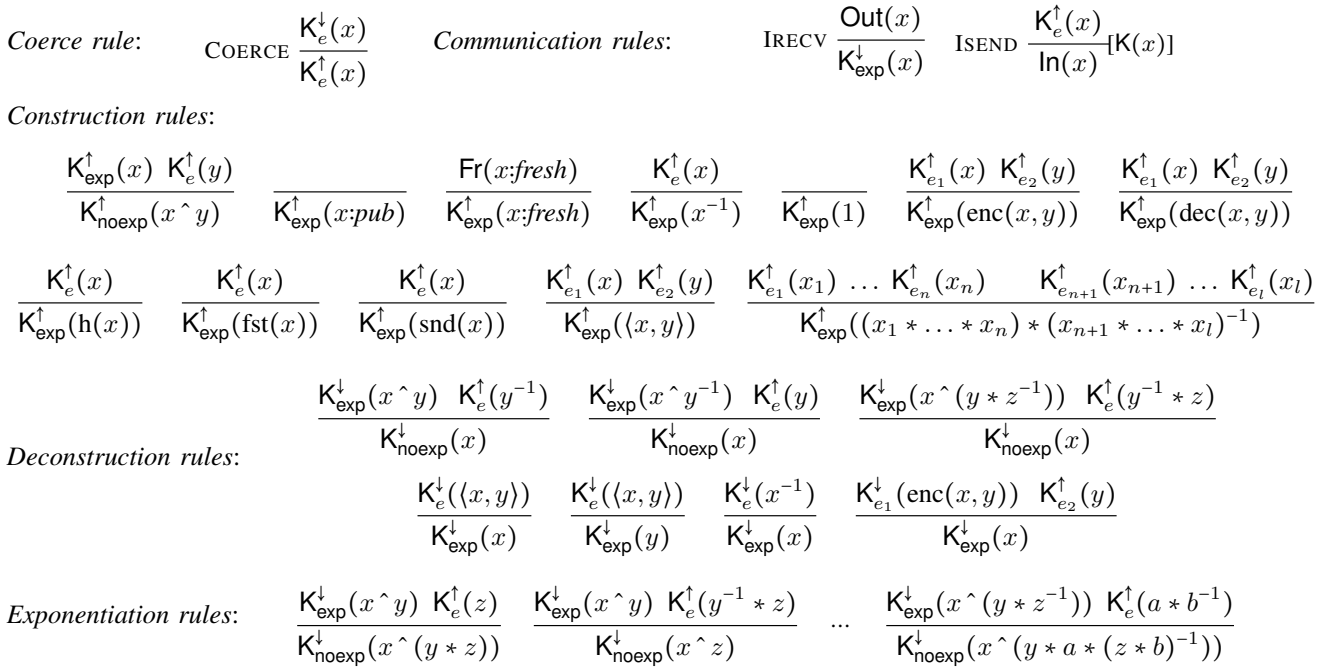

Figure 9. Normal message deduction rules *ND*. Rules containing variables e or e_i denote all variants where these are replaced by noexp or exp. Rules containing n and l denote all variants for $n \geq 1$ and $l \geq 2$. There are 42 exponentiation rules computed from the DH,AC-variants of the exponentiation rule.

In the second case, the nodes i and j can be replaced by $j : \mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathsf{g}), \mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathsf{b}) \dashrightarrow \mathsf{K}^{\uparrow}_{\mathsf{noexp}}(\mathsf{g} \mathbin{\hat{}} \mathsf{b})$, keeping all the outgoing edges of j. This replacement is possible because g is deducible and conditions **N4** and **N3** ensure that b is deducible whenever $\mathsf{a}^{-1} * \mathsf{b}$ is.

4) Properties of Normal Dependency Graphs: We prove two properties of normal dependency graphs that are crucial for our search algorithm. The first property states that every $\mathsf{K}^{\downarrow}_{e}(t)$-premise is deduced using a chain of deconstruction rules from a received message. We use here the *extended set of deconstruction rules* ND^{destr} that consists of the deconstruction and exponentiation rules from Figure 9. To define the second property, we partition the construction rules into the *implicit construction rules* $ND^{c\text{-}impl}$ consisting of the pair, inversion, and multiplication construction rules and the *explicit construction rules* $ND^{c\text{-}expl}$ consisting of the remaining construction rules and the COERCE rule. Since all messages that are products, pairs, and inverses must be constructed with implicit construction rules, we can show that if a $K^{\uparrow}_{e}(t)$ conclusion was deduced, then every message in $inp(t)$ must have been previously deduced.

Let $dg = (I, D)$ be a normal dependency graph for P. Its *deconstruction chain relation* $\dashrightarrow_{dg}$ is the smallest relation such that $c \dashrightarrow_{dg} p$ if c is a $\mathsf{K}^{\downarrow}$-conclusion in dg and (a) $c \twoheadrightarrow p \in D$ or (b) there is a premise (j, u) such that $c \twoheadrightarrow (j, u) \in D$ and $(j, 1) \dashrightarrow_{dg} p$. Our search algorithm exploits the following lemma to reason about the possible origins of $\mathsf{K}^{\downarrow}$-premises.

Lemma 2 (Deconstruction Chain)**.** *For every premise p with fact $\mathsf{K}^{\downarrow}_{e}(t)$ of dg, there is a node i in dg such that $I_i \in$ ginsts(IRECV) and $(i, 1) \dashrightarrow_{dg} p$.*

The *implicit construction dependency relation* $\twoheadrightarrow_{dg}$ *of* dg is the smallest relation such that $c \twoheadrightarrow_{dg} p$ if there is a premise (j, u) with $I_j \in ginsts(ND^{c\text{-}impl})$ such that (a) $c \twoheadrightarrow (j, u) \in D$ and $(j, 1) \twoheadrightarrow p \in D$ or (b) $c \twoheadrightarrow_{dg} (j, u)$ and $(j, 1) \twoheadrightarrow p \in D$. Our algorithm uses the following lemma to keep the construction of pairs, inverses, and products implicit in the search.

Lemma 3 (Implicit Construction)**.** *For every premise p in dg with fact $\mathsf{K}^{\uparrow}_{e}(t)$ and every message $m \in_{AC} inp(t)$ with $m \neq_{AC} t$, there is a conclusion $(i, 1)$ in dg with fact $\mathsf{K}^{\uparrow}_{e'}(m')$ such that $I_i \in ginsts(ND^{c\text{-}expl})$, $m' =_{AC} m$, and $(i, 1) \twoheadrightarrow_{dg} p$.*

Example 6. For the normal message deduction subgraph dg in Figure 8b, $(5, 1) \dashrightarrow_{dg} (9, 1)$, $(8, 1) \dashrightarrow_{dg} (9, 1)$, and $(6, 1) \dashrightarrow_{dg} (7, 1)$, but not $(6, 1) \dashrightarrow_{dg} (9, 1)$ because $(8, 2)$ is not a $\mathsf{K}^{\downarrow}$-premise. We have $(9, 1) \twoheadrightarrow_{dg} (11, 1)$, but not $(10, 1) \twoheadrightarrow_{dg} (11, 1)$ because $\twoheadrightarrow_{dg}$ requires at least one *inner* implicit construction node.

VI. AUTOMATED PROTOCOL ANALYSIS

In this section, we give an algorithm for determining whether $P \vDash_{E_{DH}} \varphi$ for a $*$-restricted protocol P and a guarded trace property φ. Guarded trace properties are an expressive subset of trace formulas. Our algorithm uses constraint solving to perform a complete search for counter-examples to $P \vDash_{E_{DH}} \varphi$, i.e., it attempts a proof by contradiction. This problem is undecidable and our algorithm does

Trace formula reduction rules:

$\mathbf{S}_{\approx}$: $\Gamma \rightsquigarrow_P \|_{\sigma \in unify_{AC}(t_1,t_2)}(\Gamma\sigma)$ if $(t_1 \approx t_2) \in \Gamma$ and $t_1 \neq_{AC} t_2$

$\mathbf{S}_{\doteq}$: $\Gamma \rightsquigarrow_P \Gamma\{i/j\}$ if $(i \doteq j) \in \Gamma$ and $i \neq j$

$\mathbf{S}_{@}$: $\Gamma \rightsquigarrow_P \|_{ri \in \lceil P \rceil^{DH} \cup \{\text{ISEND}\}} \|_{f' \in acts(ri)}(i : ri,\ f \approx f', \Gamma)$ if $(f@i) \in \Gamma$ and $(f@i) \notin_{AC} as(\Gamma)$

$\mathbf{S}_{\bot}$: $\Gamma \rightsquigarrow_P \bot$ if $\bot \in \Gamma$

$\mathbf{S}_{\neg,\approx}$: $\Gamma \rightsquigarrow_P \bot$ if $\neg(t \approx t) \in_{AC} \Gamma$

$\mathbf{S}_{\neg,\doteq}$: $\Gamma \rightsquigarrow_P \bot$ if $\neg(i \doteq i) \in \Gamma$

$\mathbf{S}_{\neg,@}$: $\Gamma \rightsquigarrow_P \bot$ if $\neg(f@i) \in \Gamma$ and $(f@i) \in as(\Gamma)$

$\mathbf{S}_{\neg,\lessdot}$: $\Gamma \rightsquigarrow_P (i \lessdot j, \Gamma) \parallel (\Gamma\{i/j\})$ if $\neg(j \lessdot i) \in \Gamma$ and neither $i \lessdot_\Gamma j$ nor $i = j$

$\mathbf{S}_{\vee}$: $\Gamma \rightsquigarrow_P (\phi_1, \Gamma) \parallel (\phi_2, \Gamma)$ if $(\phi_1 \vee \phi_2) \in_{AC} \Gamma$ and $\{\phi_1, \phi_2\} \cap_{AC} \Gamma = \varnothing$

$\mathbf{S}_{\wedge}$: $\Gamma \rightsquigarrow_P (\phi_1, \phi_2, \Gamma)$ if $(\phi_1 \wedge \phi_2) \in_{AC} \Gamma$ and not $\{\phi_1, \phi_2\} \subseteq_{AC} \Gamma$

$\mathbf{S}_{\exists}$: $\Gamma \rightsquigarrow_P (\phi\{y/x\}, \Gamma)$ if $(\exists x{:}s.\ \phi) \in \Gamma$, $\phi\{w/x\} \notin_{AC} \Gamma$ for every term w of sort s, and $y{:}s$ fresh

$\mathbf{S}_{\forall}$: $\Gamma \rightsquigarrow_P (\psi\sigma, \Gamma)$ if $(\forall \vec{x}.\ \neg(f@i) \vee \psi) \in \Gamma$, $dom(\sigma) = set(\vec{x})$, $(f@i)\sigma \in_{AC} as(\Gamma)$, and $\psi\sigma \notin_{AC} \Gamma$

Graph constraint reduction rules:

$\mathbf{U}_{lbl}$: $\Gamma \rightsquigarrow_P (ri \approx ri', \Gamma)$ if $\{i : ri, i : ri'\} \subseteq \Gamma$ and $ri \neq_{AC} ri'$

$\mathbf{DG1}_1$: $\Gamma \rightsquigarrow_P \bot$ if $i \lessdot_\Gamma i$

$\mathbf{DG1}_2$: $\Gamma \rightsquigarrow_P (f \approx f', \Gamma)$ if $c \nrightarrow p \in \Gamma$, $(c, f) \in cs(\Gamma)$, $(p, f') \in ps(\Gamma)$, and $f \neq_{AC} f'$

$\mathbf{DG2}_1$: $\Gamma \rightsquigarrow_P (\text{if } u = v \text{ then } \Gamma\{i/j\} \text{ else } \bot)$ if $\{(i, v) \nrightarrow p, (j, u) \nrightarrow p\} \subseteq \Gamma$ and $i \neq j$

$\mathbf{DG2}_{2,P}$: $\Gamma \rightsquigarrow_P \|_{ri \in \lceil P \rceil^{DH} \cup \{\text{ISEND},\text{FRESH}\}} \|_{u \in idx(concs(ri))}(i : ri, (i, u) \nrightarrow p, \Gamma)$
if p is an open f-premise in Γ, f is not a $\mathsf{K}^{\uparrow}$- or $\mathsf{K}^{\downarrow}$-fact, and i fresh

$\mathbf{DG3}$: $\Gamma \rightsquigarrow_P (\text{if } u = v \text{ then } \Gamma\{i/j\} \text{ else } \bot)$ if $\{c \nrightarrow (i, v), c \nrightarrow (j, u)\} \subseteq \Gamma$, c linear in Γ, and $i \neq j$,

$\mathbf{DG4}$: $\Gamma \rightsquigarrow_P \Gamma\{i/j\}$ if $\{i : \dashv[]\rightarrow \mathsf{Fr}(m), j : \dashv[]\rightarrow \mathsf{Fr}(m)\} \subseteq_{AC} \Gamma$ and $i \neq j$

$\mathbf{N1}$: $\Gamma \rightsquigarrow_P \bot$ if $(i : ri) \in \Gamma$ and ri not $\downarrow_{DH}$-normal

$\mathbf{N5,6}$: $\Gamma \rightsquigarrow_P \Gamma\{i/j\}$ if $\{((i,1), \mathsf{K}^{d}_{e}(t)), ((j,1), \mathsf{K}^{d'}_{e'}(t))\} \subseteq_{AC} cs(\Gamma)$, $i \neq j$, *and*
$d = d'$ or $\{i, j\} \cap \{k \mid \exists ri \in insts(\{\text{PAIR}\uparrow, \text{INV}\uparrow, \text{COERCE}\}).\ (k : ri) \in \Gamma\} = \varnothing$

$\mathbf{N6}$: $\Gamma \rightsquigarrow_P (i \lessdot j, \Gamma)$ if $((j, v), \mathsf{K}^{\uparrow}_{e'}(t)) \in ps(\Gamma)$, $m \in_{AC} inp(t)$, $((i, u), \mathsf{K}^{\downarrow}_{e}(m)) \in cs(\Gamma)$, and not $i \lessdot_\Gamma j$

$\mathbf{N7}$: $\Gamma \rightsquigarrow_P \bot$ if $(i : \mathsf{K}^{\downarrow}_{\mathsf{exp}}(s_1), \mathsf{K}^{\uparrow}_{e}(t_1) \dashv[]\rightarrow \mathsf{K}^{\downarrow}_{\mathsf{noexp}}(s_2 \hat{}\ t_2)) \in \Gamma$, s_2 is of sort *pub*, and $inp(t_2) \subseteq inp(t_1)$

Message deduction constraint reduction rules:

$\mathbf{DG2}_{2,\uparrow i}$: $\Gamma \rightsquigarrow_P \|_{(l \dashv[]\rightarrow \mathsf{K}^{\uparrow}_{e}(t)) \in ND^{c\text{-}expl}} (i : (l \dashv[]\rightarrow \mathsf{K}^{\uparrow}_{e}(t)), t \approx m, (i, 1) \twoheadrightarrow p, \Gamma)$
if p is an open implicit m-construction in Γ, m non-trivial, and i fresh

$\mathbf{DG2}_{2,\uparrow e}$: $\Gamma \rightsquigarrow_P \|_{ri \in ND^{c\text{-}expl}} (i : ri, (i, 1) \nrightarrow p, \Gamma)$
if p is an open $\mathsf{K}^{\uparrow}_{e}(m)$-premise in Γ, $\{m\} = inp(m)$, m non-trivial, and i fresh

$\mathbf{DG2}_{2,\downarrow}$: $\Gamma \rightsquigarrow_P (i : \mathsf{Out}(y) \dashv[]\rightarrow \mathsf{K}^{\downarrow}_{\mathsf{exp}}(y), (i, 1) \dashrightarrow p, \Gamma)$ if p is an open $\mathsf{K}^{\downarrow}_{e}(m)$-premise in Γ and y, i fresh

$\mathbf{DG2}_{\dashrightarrow}$: $(c \dashrightarrow p, \Gamma) \rightsquigarrow_P (c \nrightarrow p, \Gamma) \parallel \|_{ri \in ND^{destr}} (i : ri, c \nrightarrow (i, 1), (i, 1) \dashrightarrow p, \Gamma)$
if $(c, \mathsf{K}^{\downarrow}_{e}(m)) \in cs(\Gamma)$, $m \notin \mathcal{V}_{msg}$, and i fresh

We assume that the multiset rewriting rules in $\lceil P \rceil^{DH}$, $ND^{c\text{-}expl}$, and ND^{destr} are renamed apart from Γ. We write $\Gamma\{a/b\}$ for the substitution of all occurrences of b with a in Γ. We write $\Gamma \rightsquigarrow_P \Gamma_1 \parallel \ldots \parallel \Gamma_n$ for $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_n\}$, which denotes an n-fold case distinction. We overload notation and write $\bot$ for the empty set of constraint systems.

Figure 10. Rules defining the constraint-reduction relation $\rightsquigarrow_P$, explained in Sections VI-C and VI-D.

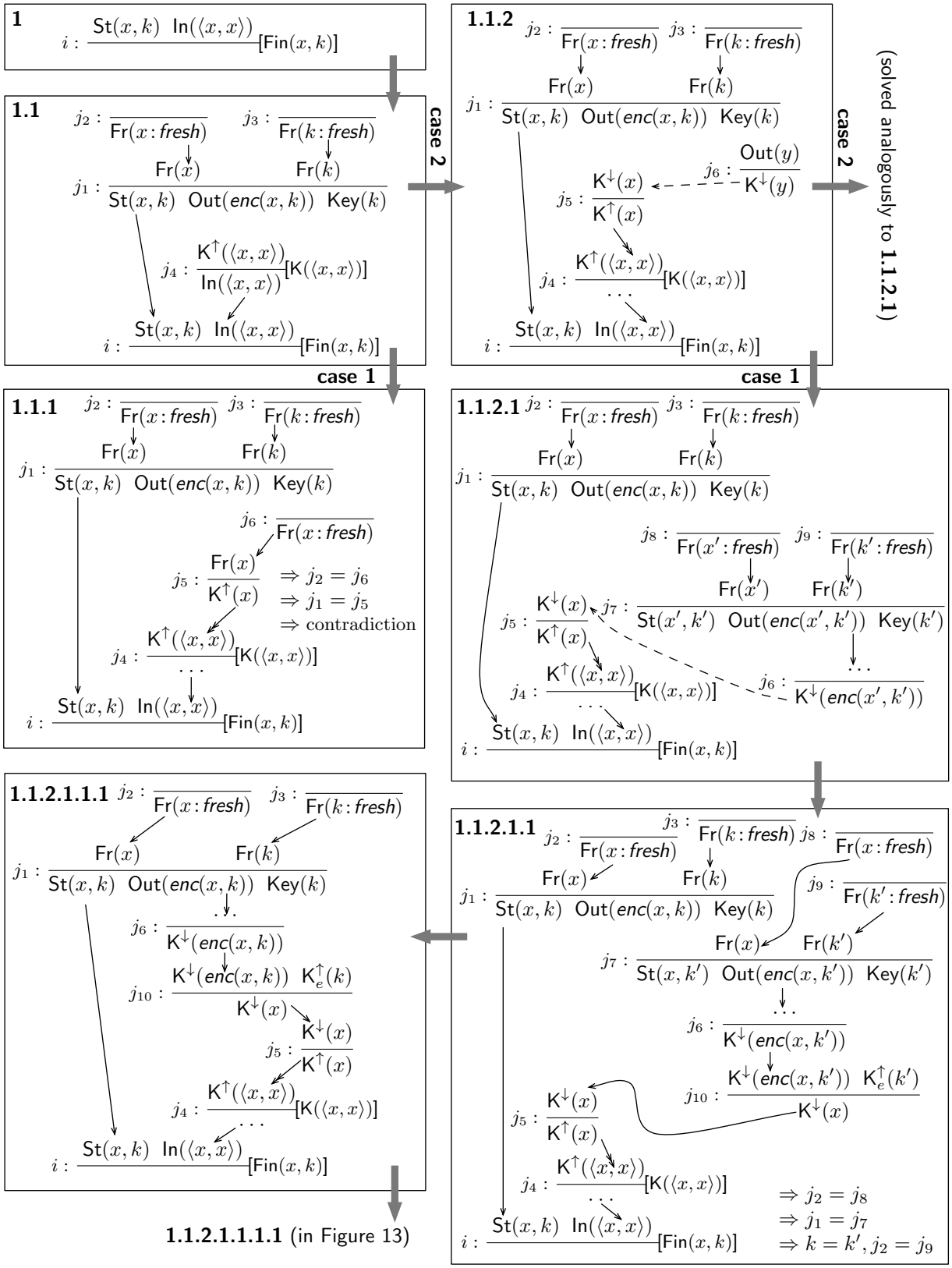

Figure 11. Constraint systems constructed by our algorithm, as explained in Example 7, when verifying $\forall x\, k\, i.\mathsf{Fin}(x,k)@i \Rightarrow \exists j.\mathsf{Rev}(k)@j$ for the protocol from Example 1. The large gray arrows denote constraint-reduction steps and "..." at either end of an edge refers to the fact at the other end. In every constraint system, variables with the same name are of the same sort and variables that have no sort annotation are of sort *msg*.

not always terminate. Nevertheless, it often finds a counter-example (an attack) or succeeds in unbounded verification.

In the following, we define guarded trace properties and constraints. Afterwards we give our constraint-solving algorithm and several examples.

A. Guarded Trace Properties

In the remainder of this section, let f range over facts and i, j over temporal variables. A trace formula φ is in *negation normal form* if it is built such that negation is only applied to trace atoms and all other logical connectives are $\wedge$, $\vee$, $\forall$, or $\exists$. Such a trace formula φ is a *guarded trace formula* if all its quantifiers are of the form $\exists \vec{x}.(f@i) \wedge \psi$ or $\forall \vec{x}.\neg(f@i) \vee \psi$ for an action $f@i$, a guarded trace formula ψ, and $x \in \mathit{vars}(f@i) \cap (\mathcal{V}_{\mathit{msg}} \cup \mathcal{V}_{\mathit{temp}})$ for every $x \in \vec{x}$. A guarded trace formula φ is a *guarded trace property* if it is closed and $t \in \mathcal{V} \cup \mathit{PN}$ holds for all terms t occurring in φ.

Note that we restrict both universal and existential quantification and, as a result, the set of guarded trace properties is closed under negation. This, together with the support for quantifier alternations and the explicit comparison of timepoints, makes guarded trace properties well-suited for specifying advanced security properties. In our case studies, it was possible to automatically convert the specified security properties, including the eCK model from Figure 4, to guarded trace properties. The conversion first rewrites the given formula to negation normal form and pushes quantifiers inward. Then, it replaces each body φ of a universal quantifier that is not a disjunction with $\varphi \vee \bot$. The rewriting for existential quantifiers is analogous.

All terms in a guarded trace property must be either variables or public names. This is not a limitation in practice since the terms required to express a security property can be added to the actions of a protocol's rewriting rules. Together with the requirement of guarding all quantified variables, this ensures that guarded trace properties are invariant under $\downarrow_{DH}$-normalization of traces. Combined with Lemma 1, this allows us to switch from verification in a multiset rewriting semantics modulo E_{DH} to verification in a dependency graph semantics modulo AC.

Theorem 1. *For every $*$-restricted protocol P and every guarded trace property φ,*

$$P \vDash_{E_{DH}} \varphi \quad \textit{iff} \quad \{\mathit{trace}(dg) \mid dg \in \mathit{ndgraphs}(P)\} \vDash_{AC} \varphi\ .$$

B. Syntax and Semantics of Constraints

In the remainder of this section, let ri range over multiset rewriting rule instances, u and v over natural numbers, and φ over guarded trace formulas. A *graph constraint* is either a *node* $i : ri$, an *edge* $(i,u) \rightarrowtail (j,v)$, a *deconstruction chain* $(i,u) \dashrightarrow (j,v)$, or an *implicit construction* $(i,u) \twoheadrightarrow (j,v)$. A *constraint* is a graph constraint or a guarded trace formula.

A *structure* is a tuple (dg,θ) of a dependency graph $dg = (I,D)$ and a valuation θ. We denote the application of the homomorphic extension of θ to a rule instance ri by $ri\,\theta$. We define when the structure (dg,θ) *satisfies* a constraint γ, written $(dg,\theta) \Vdash \gamma$, as follows.

$$\begin{aligned}
&(dg,\theta) \Vdash i : ri && \text{iff } \theta(i) \in \mathit{idx}(I) \text{ and } ri\,\theta =_{AC} I_{\theta(i)}\\
&(dg,\theta) \Vdash (i,u) \rightarrowtail (j,v) && \text{iff } (\theta(i),u) \rightarrowtail (\theta(j),v) \in D\\
&(dg,\theta) \Vdash (i,u) \dashrightarrow (j,v) && \text{iff } (\theta(i),u) \dashrightarrow_{dg} (\theta(j),v)\\
&(dg,\theta) \Vdash (i,u) \twoheadrightarrow (j,v) && \text{iff } (\theta(i),u) \twoheadrightarrow_{dg} (\theta(j),v)\\
&(dg,\theta) \Vdash \varphi && \text{iff } (\mathit{trace}(dg),\theta) \vDash_{AC} \varphi
\end{aligned}$$

A *constraint system* Γ is a finite set of constraints. The structure (dg,θ) satisfies Γ, written $(dg,\theta) \Vdash \Gamma$, if (dg,θ) satisfies each constraint in Γ. We say that (dg,θ) is a *P-model of* Γ, if dg is a normal dependency graph for P and $(dg,\theta) \Vdash \Gamma$. A *$P$-solution* of Γ is a normal dependency graph dg for P such that there is a valuation θ with $(dg,\theta) \Vdash \Gamma$. Note that the free variables of a constraint system are therefore existentially quantified.

C. Constraint-Solving Algorithm

Let P be a $*$-restricted protocol and φ a guarded trace property. Exploiting Theorem 1, our algorithm searches for a counter-example to $P \vDash_{E_{DH}} \varphi$ by trying to construct a P-solution to the constraint system $\{\hat{\varphi}\}$, where $\hat{\varphi}$ is $\neg\varphi$ rewritten into negation normal form. Our algorithm is based on the constraint-reduction relation $\rightsquigarrow_P$ between constraint systems and sets of constraint systems. We use sets of constraint systems to represent case distinctions.

Intuitively, $\rightsquigarrow_P$ refines constraint systems and our algorithm works by refining the initial constraint system $\{\hat{\varphi}\}$ until it either encounters a solved system or all systems contain (trivially) contradictory constraints. In the following, we first define $\rightsquigarrow_P$ and then state our algorithm. Afterwards, we give examples that explain and illustrate both the constraint-reduction rules defining $\rightsquigarrow_P$ and our algorithm.

The rules defining the *constraint-reduction relation* $\rightsquigarrow_P$ are given in Figure 10. There are two types of constraint-reduction rules: (1) *simplification rules* that remove contradictory constraint systems or refine constraint systems by simplifying constraints and (2) *case distinction rules* that refine constraint systems by adding further constraints. The design choices underlying our rules are motivated by their use in our algorithm. It requires them to be sound and complete and we must be able to extract a P-solution from every solved constraint system, i.e., every system that is irreducible with respect to $\rightsquigarrow_P$. The rule names refer to the form of guarded trace formulas that they solve or the property of normal dependency graphs that they ensure. There are no rules for ensuring the properties **N2–4**, as they are maintained as invariants by our algorithm.

The formal definition of our constraint-reduction rules in Figure 10 relies on the following additional conventions and definitions. We extend the equality $\approx$ over terms to facts and rule instances by interpreting the constructors for facts and

rule instances as free function symbols. We write PAIR$\uparrow$ for the construction rule for pairs and INV$\uparrow$ for the construction rule for inverses. Moreover, for a constraint system Γ, its *actions* $as(\Gamma)$, *premises* $ps(\Gamma)$, and *conclusions* $cs(\Gamma)$ are defined as follows.

$$as(\Gamma) = \{f@i \mid \exists r\, a.\ (i : l \dashv[\ a\ ]\vdash r) \in \Gamma \wedge f \in a\}$$
$$ps(\Gamma) = \{((i,u), l_u) \mid \exists r\, a.\ (i : l \dashv[\ a\ ]\vdash r) \in \Gamma \wedge u \in idx(l)\}$$
$$cs(\Gamma) = \{((i,v), r_v) \mid \exists l\, a.\ (i : l \dashv[\ a\ ]\vdash r) \in \Gamma \wedge v \in idx(r)\}$$

A conclusion c is *linear in* Γ if there is a linear fact f such that $(c, f) \in cs(\Gamma)$. We say that p is an *open f-premise in* Γ if $(p, f) \in ps(\Gamma)$ and p has no incoming edges or deconstruction chains in Γ. We say that p is an *open implicit m-construction in* Γ if there is a premise $(p, \mathsf{K}^{\uparrow}_{e}(t))$ of Γ with $m \in inp(t) \setminus \{t\}$ and there is no implicit construction $c \twoheadrightarrow p$ in Γ that starts from a $\mathsf{K}^{\uparrow}_{e'}(m)$-conclusion. A term m is *trivial* if $m \in \mathcal{V}_{msg} \cup \mathcal{V}_{pub} \cup PN \cup \{1\}$, where the term 1 denotes the unit in the group of exponents. The *temporal order of* Γ is

$$\begin{aligned}(\lessdot_\Gamma) = \{(i,j) \mid (i \lessdot j) \in \Gamma \vee\ & \exists u\, v.\ ((i,u) \rightarrowtail (j,v)) \in \Gamma \\ & \vee ((i,u) \twoheadrightarrow (j,v)) \in \Gamma \\ & \vee ((i,u) \dashrightarrow (j,v)) \in \Gamma\}^+ .\end{aligned}$$

We call $\mathsf{K}^{\uparrow}$- and $\mathsf{K}^{\downarrow}$-premises *message deduction constraints*.

We give our constraint-solving algorithm in Figure 12. It uses a set of constraint systems as its state $\mathbf{\Omega}$. It starts with the state $\{\{\hat{\varphi}\}\}$. Afterwards, in lines 4–6, it repeatedly applies constraint-reduction steps as long as the state is non-empty and does not contain a solved constraint system. To formalize the loop condition, we use $solved(\mathbf{\Omega})$ to denote the set of solved constraint systems in $\mathbf{\Omega}$. For automated protocol analysis, we use a heuristic (explained in [18]) to make the choice in line 5. Upon termination of the while-loop, the algorithm has either found a solved constraint system (an attack) or it proved that $\{\{\hat{\varphi}\}\}$ has no P-solution and therefore $P \vDash_{E_{DH}} \varphi$ holds. The following two theorems justify the correctness of our algorithm.

Theorem 2. *The constraint-reduction relation $\rightsquigarrow_P$ is sound and complete; i.e., for every $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_n\}$, the set of P-solutions of Γ is equal to the union of the sets of P-solutions of all Γ_i, with $1 \le i \le n$.*

Theorem 3. *We can construct a P-solution from every solved system in the state $\mathbf{\Omega}$ of our constraint-solving algorithm.*

The correctness of Theorem 3 relies on the properties of solved constraint systems as well as invariants maintained by our constraint-solving algorithm, as explained in [18].

D. Extended Examples

We now give two examples that illustrate our constraint-reduction rules. Our emphasis is on the rules and their application, rather than the heuristics used in our algorithm.

1: **function** SOLVE($P \vDash_{E_{DH}} \varphi$)
2: $\quad\hat{\varphi} \leftarrow \neg\varphi$ rewritten into negation normal form
3: $\quad\mathbf{\Omega} \leftarrow \{\{\hat{\varphi}\}\}$
4: $\quad$**while** $\mathbf{\Omega} \neq \varnothing$ and $solved(\mathbf{\Omega}) = \varnothing$ **do**
5: $\quad\quad$choose $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_k\}$ such that $\Gamma \in \mathbf{\Omega}$
6: $\quad\quad\mathbf{\Omega} \leftarrow (\mathbf{\Omega} \setminus \{\Gamma\}) \cup \{\Gamma_1, \ldots, \Gamma_k\}$
7: $\quad$**if** $solved(\mathbf{\Omega}) \neq \varnothing$
8: $\quad\quad$**then return** "attack(s) found: ", $solved(\mathbf{\Omega})$
9: $\quad\quad$**else return** "verification successful"

Figure 12. Pseudocode of our constraint solving algorithm.

Example 7. Consider the protocol P from Example 1 and the formula $\forall x\ k\ i.\ \mathsf{Fin}(x,k)@i \Rightarrow \exists j.\ \mathsf{Rev}(k)@j$, which can be rewritten to the guarded trace property $\varphi = \forall x\ k\ i.\ \neg(\mathsf{Fin}(x,k)@i) \vee (\exists j.\ \mathsf{Rev}(k)@j \wedge \neg(\bot))$. We explain how to use our constraint-reduction rules to show that $P \vDash_{E_{DH}} \varphi$, i.e., to show that $\{\exists x\ k\ i.\ \psi\}$ has no P-solutions, for $\psi = \mathsf{Fin}(x,k)@i \wedge (\forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot)$.

By repeated application of the rules $\mathtt{s}_{\exists}$ and $\mathtt{s}_{\wedge}$, we replace the existentially quantified variables x, k, and i with fresh free variables and split the conjunction in ψ. The resulting constraint system is

$$\{\exists x\ k\ i.\ \psi,\ \exists k\ i.\ \psi,\ \exists i.\ \psi,\ \psi,\ \mathsf{Fin}(x,k)@i,\ \forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot\}\ .$$

Its P-solutions are the normal dependency graphs for P that contain a node with a $\mathsf{Fin}(x,k)$ action, but no node with a $\mathsf{Rev}(k)$ action. Note that the formulas $\exists x\ k\ i.\ \psi$, $\exists k\ i.\ \psi$, $\exists i.\ \psi$, and ψ are solved in the sense that no reduction rule applies to them anymore.

The only rule applicable to the constraint system in the above state is $\mathtt{s}_{@}$, which enumerates all rewriting rules that could give rise to an action. We therefore apply $\mathtt{s}_{@}$ to the $\mathsf{Fin}(x,k)$ action of node i and solve the introduced equalities using $\mathtt{s}_{\approx}$. There is only one resulting constraint system, depicted in Figure 11 as System **1**. Here we interpret a constraint system as a partial, symbolic dependency graph annotated with restrictions on its trace, stated as guarded trace formulas. We do not depict the trace restriction $\forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot$, as it is included in all constraint systems. We also omit the exp tags of K-facts because the protocol does not use exponentiation.

The large gray arrows in Figure 11 denote the constraint-reduction steps performed. Multiple successors result from case distinctions. Constraint systems with no successors are contradictory for the reason given in the figure. In some cases, we contract multiple reduction steps for ease of presentation.

System **1** has multiple open non-K-premises. We solve these by repeatedly applying $\mathtt{DG2}_{2,P}$ until no more open non-K-premises remain. Intuitively, rule $\mathtt{DG2}_{2,P}$ solves open non-K-premises by enumerating all possible rewriting rules that have a unifying conclusion. The unification is entailed by the constraint $(i,u) \rightarrowtail p$ and the rules $\mathtt{DG1}_2$ and $\mathtt{s}_{\approx}$, which ensure the equality of facts connected by an edge. After

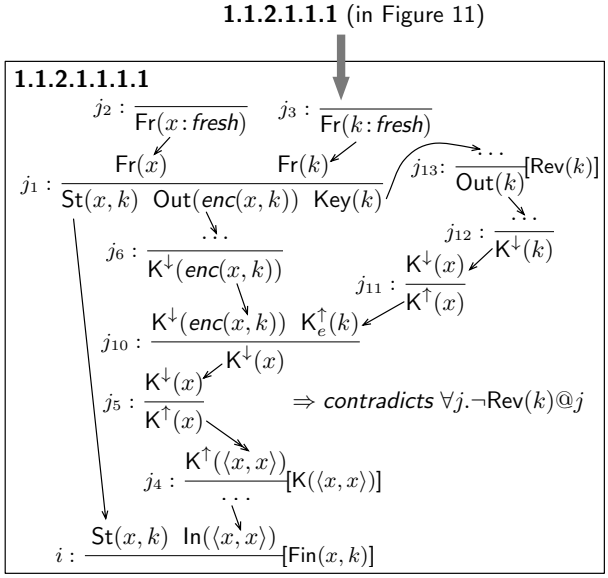

Figure 13. Example constraint system **1.1.2.1.1.1.1**.

solving all non-K-premises, the only resulting constraint system is System **1.1**.

There is only one remaining open premise in System **1.1**: $\mathsf{K}^{\uparrow}(\langle x,x \rangle)$. It is solved using rule $\mathtt{DG2}_{2,\uparrow i}$, which exploits Lemma 3 to search directly for the possible origin of the required input component x:*fresh*. This rule improves the efficiency of our algorithm as it allows reasoning modulo pairing, inversion, and multiplication. Note that the rules $\mathtt{DG2}_{2,\uparrow e}$ and $\mathtt{DG2}_{2,\uparrow i}$ exclude solving $\mathsf{K}^{\uparrow}(x)$ premises where x is a variable of sort *msg* or *pub*, as such facts can always be constructed by the adversary. These rules may however become applicable once x is instantiated with another term in a later step. There are two multiset rewriting rules in *ND* whose conclusions are possible origins of x:*fresh*.

The case for the rule $(\mathsf{Fr}(x\!:\!\mathit{fresh}) \text{–}[]\!\!\rightarrow \mathsf{K}^{\updownarrow}_{\mathsf{exp}}(x\!:\!\mathit{fresh}))$ is shown in System **1.1.1**. Note that FRESH instances must be unique and linear facts have at most one outgoing edge. This is ensured by the rules **DG4** and **DG3**, which together with the rule $\mathtt{U}_{\mathit{lbl}}$ give rise to the chain of implications depicted in System **1.1.1**. The contradiction follows because the rewriting rule instances of nodes j_1 and j_5 are not unifiable.

The case for the COERCE rule is shown in System **1.1.2**. Note that solving $\mathsf{K}^{\downarrow}$-premises by enumerating the rules with unifying conclusions leads to non-termination, as the $\mathsf{K}^{\downarrow}$-premises of deconstruction rules are larger than their conclusions. This justifies the existence of rule $\mathtt{DG2}_{2,\downarrow}$, which solves the $\mathsf{K}^{\downarrow}(x)$-premise by exploiting Lemma 2 to introduce a deconstruction chain starting from an IRECV instance. Rule $\mathtt{DG2}_{2,P}$ then solves the open premise $\mathsf{Out}(y)$ of j_6, i.e., it enumerates all protocol rules that send messages. There are two such rules in protocol P.

System **1.1.2.1** shows the case for the first protocol rule, which is renamed apart from System **1.1.2** since node j_7 might be a different rewriting rule instance than node j_1. The deconstruction chain from j_6 to j_5 is refined by rule $\mathtt{DG2}_{\dashrightarrow}$. This rule embodies the case distinction that a deconstruction chain is either just an edge or an edge to an instance of a rule from ND^{destr} and another chain starting from the conclusion of this instance. We disallow refining chains that start from a message variable, as this would lead to non-termination. Constraint systems containing such a chain are often pruned using rule **N6**, as explained in Example 8. After using rule $\mathtt{DG2}_{\dashrightarrow}$ to refine the deconstruction chain in System **1.1.2.1** twice, we obtain System **1.1.2.1.1** where x and x' are identified and the deconstruction chain has been replaced with the edges to and from the decryption j_{10}. This is the only way to refine this chain starting from $\mathsf{K}^{\downarrow}(\mathrm{enc}(x',k'))$, as all other deconstruction rules lead to cases with edges between non-unifiable facts. Again the uniqueness of FRESH instances leads to a chain of implications and results in System **1.1.2.1.1.1**.

The only remaining open premise in this system is the $\mathsf{K}^{\uparrow}(k)$-premise of the decryption j_{10}. The constraint-reduction steps required to solve this premise are similar to the ones used to solve the $\mathsf{K}^{\uparrow}(x)$-premise in System **1.1**. System **1.1.2.1.1.1.1** in Figure 13 is the only one of the resulting constraint systems that is not trivially contradictory due to the uniqueness of FRESH instances. In fact, this system could be instantiated to a normal dependency graph, if it were not for the trace restriction $\forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot$. Since System **1.1.2.1.1.1.1** contains node j_{13} with a $\mathsf{Rev}(k)$ action, we can use $\mathtt{S}_{\neg,@}$ and $\mathtt{S}_{\bot}$ to derive a contradiction. We first derive $\bot$ by applying $\mathtt{S}_{\neg,@}$ to the trace restriction $\forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot$, instantiating j with j_{13}. Then, we apply rule $\mathtt{S}_{\bot}$. In general, we can always saturate constraint systems under universally quantified guarded trace formulas. This works because all trace formulas in a constraint system are guarded and the number of trace formulas derivable from a constraint system using just $\mathtt{S}_{\neg,@}$ is finite.

System **1.1.2.2** is also contradictory, which we show in [18]. Thus, we terminate without finding an attack and, as our search is complete, we therefore have a proof that $P \vDash_{E_{DH}} \varphi$.

The above example provides intuition for all rules except **N1**, **N5,6**, **N6**, and **N7**, which enforce normal-form message deduction. We explain them in the following paragraphs.

Rule **N1** ensures that all rule instances in node constraints are in $\downarrow_{DH}$-normal form, i.e., it allows us to prune constraint systems containing DH-reducible terms. Intuitively, this rule prevents inconsistent instantiations of variables occuring in the variants of a multiset rewriting rule. Consider for example the multiset rewriting rule $\mathsf{In}(x) \text{–}[]\!\!\rightarrow \mathsf{Out}(\mathrm{fst}(x))$. The corresponding DH,AC-variants are $\mathsf{In}(x) \text{–}[]\!\!\rightarrow \mathsf{Out}(\mathrm{fst}(x))$ and $\mathsf{In}(\langle y,z \rangle) \text{–}[]\!\!\rightarrow \mathsf{Out}(y)$. The rule **N1** allows pruning constraint systems where a node is labeled with the first variant and x is instantiated with a pair, as such constraint systems contradict the implicit assumption of the first rule variant, i.e., $\mathrm{fst}(x)$

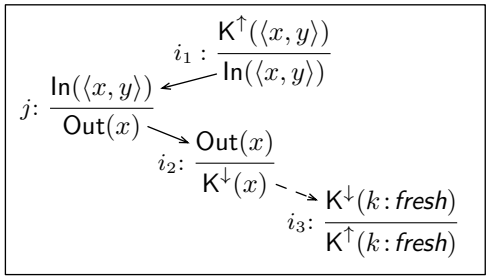

Figure 14. A contradictory constraint system where the adversary uses an instance of the protocol rule $\mathsf{In}(\langle x, y, \rangle) \text{–}[]\text{→} \mathsf{Out}(x)$ to forward to himself the message x, which he deduced himself.

is not DH-reducible. Pruning constraint systems with DH-reducible terms is crucial when reasoning about the variants of DH exponentiations. Rule **N7** provides further support for reasoning about DH exponentatiations, as it prunes constraint systems containing instances of deconstruction rules that can be replaced by instances of the construction rule for exponentiation.

Rule **N5,6** ensures that each $\mathsf{K}^{\uparrow}$- and $\mathsf{K}^{\downarrow}$-fact is derived, and therefore solved, at most once. Moreover, it ensures that every $\mathsf{K}^{\uparrow}$-premise deriving the same message as a $\mathsf{K}^{\downarrow}$-conclusion occurs after the $\mathsf{K}^{\downarrow}$-conclusion. This is required for Theorem 3 and allows pruning some constraint systems.

Rule **N6** prunes constraint systems where the adversary forwards a message via the protocol to himself. Such constraint systems occur when unfolding a deconstruction chain until it starts from a message variable whose content is received from the adversary. This is best seen in an example.

Example 8. Consider solving the premise $\mathsf{K}^{\uparrow}(k{:}\mathit{fresh})$ in the context of a protocol that contains the multiset rewriting rule $\mathsf{In}(\langle x, y, \rangle) \text{–}[]\text{→} \mathsf{Out}(x)$. One of the cases that we must consider is captured by the constraint system depicted in Figure 14. It states that the adversary might deduce k using a deconstruction chain starting from the message x sent by node j. Note that we suppress the exponentiation tags, as they are irrelevant for this example. This constraint system is contradictory because, in all its solutions, the adversary must deduce $\mathsf{K}^{\uparrow}(x)$ before $\mathsf{K}^{\uparrow}(\langle x, y\rangle)$, which implies that he cannot deduce $\mathsf{K}^{\downarrow}(x)$ afterwards, as required by node i_3. We show that this constraint system is contradictory using rule $\mathbf{DG1}_1$ after we used rule **N6** to derive $i_3 \lessdot i_1$.

VII. Case Studies

We implemented our constraint-solving algorithm in a tool, called the Tamarin prover. It provides both a command-line interface and a graphical user interface, which allows to interactively inspect and construct attacks and proofs. We evaluated our algorithm on numerous protocols. Table I lists the results, run on a laptop with an Intel i7 Quad-Core processor. The tool and all models are available at [19].

We modeled the Signed Diffie-Hellman (SIG-DH) protocol, the STS protocol and two fixes [2], the UM [28], KEA+ [4], and NAXOS [1] protocols, and the TS1, TS2, and TS3 protocols [22] and their updated versions [29]. We also modeled NIST's KAS1 and KAS2 protocols [30] and the related DH2 protocol by Chatterjee et. al. [31]. For each protocol, we formalized its intended and related security models and analyzed them using Tamarin. For example, to verify Key Independence (KI) for STS, we model that the adversary can reveal certain session keys. Additionally, the adversary can register public keys for himself, even if those keys have been previously registered for another identity. In this example, we find the UKS attack reported in [2]. The first fix from [2] requires a Proof-of-Possession (PoP) of the private key for registering a public key. The second fix includes the identities of the participants in the signatures. We model and successfully verify both fixes. For NIST's KAS1 and KAS2 protocols [30], our analysis confirms both the security proof and the informal statements made in [31].

	Protocol	Security Model	Result	Time [s]
1.	DH2 [31]	weakened eCK [31]	proof	6.7
2.	KAS1 [30]	KI+KCI [31]	proof	0.3
3.	KAS2 [30]	weakened eCK [31]	proof	2.9
4.	KAS2 [30]	eCK	attack	0.4
5.	KEA+ [4]	KI+KCI	proof	0.5
6.	KEA+ [4]	KI+KCI+wPFS	attack	0.6
7.	NAXOS [1]	eCK	proof	5.2
8.	NAXOS [1]	eCK+PFS	attack on PFS	4.8
9.	SIG-DH	PFS	proof	0.4
10.	SIG-DH	eCK	attack	0.6
11.	STS-MAC [2]	KI, reg-PK	UKS-attack	2.7
12.	STS-MAC-fix1 [2]	KI, reg-PK (with PoP)	proof	8.6
13.	STS-MAC-fix2 [2]	KI, reg-PK	proof	1.9
14.	TS1-2004 [22]	KI	UKS-attack	0.2
15.	TS1-2008 [29]	KI	proof	0.2
16.	TS2-2004 [22]	KI+wPFS	attack on wPFS	0.4
17.	TS2-2008 [29]	KI+wPFS	proof	0.7
18.	TS3-2004/08 [22], [29]	KI+wPFS	non-termination	-
19.	UM [28]	wPFS	proof	0.7
20.	UM [28]	PFS	attack	0.4

Table I
Results of case studies

Our results indicate that, in general, Tamarin is effective and efficient. For example, it requires 5.2 seconds to verify NAXOS in the eCK model (see Figure 3).

In general, there are two sources of non-termination of our algorithm. First, if the protocol contains loops (e.g., a rule like $\mathsf{A}(x) \text{–}[]\text{→} \mathsf{A}(\mathsf{h}(x))$), then the reduction rule $\mathbf{DG2}_{2,P}$ can be applied infinitely often during backwards search. Reasoning about such protocols requires support for loop invariants, as used in program verification. Second, if the protocol can serve as a generic message deduction oracle, then our normal-form conditions may fail to eliminate sufficiently many redundant steps. This explains the non-termination for the TS3-2004/08 protocols. It remains future work to develop normal-form conditions that improve reasoning about such protocols.

VIII. Related work

Corin et. al. [32] and Armando et. al. [33] use linear temporal logics and constraint solving for security protocol

verification for a bounded number of sessions. Chevalier et. al. [7] and Shmatikov [8] prove that secrecy is decidable for a bounded number of sessions for DH theories similar to ours. Meadows et. al. [34] and Kapur et. al. [35] present unification algorithms for a DH theory similar to ours. In [35], Kapur et. al. show the undecidability of unification modulo a DH theory that also allows addition of exponents.

[12]–[14] support verification for an unbounded number of sessions, but do not consider inverses. Blanchet et. al. [12] extend ProVerif [36] to handle the property that $(x\hat{}y)\hat{}z \simeq (x\hat{}z)\hat{}y$. Goubault-Larrecq [13] accounts for this property using a Horn-theory approach and resolution modulo AC. Escobar et. al. [14] use Maude-NPAand equational unification to analyze secrecy properties of DH protocols. Since Maude-NPA supports user-specified equational theories, the verification problem with respect to our DH theory can be specified. It is however unclear if Maude-NPA can achieve unbounded verification for such a theory. In the free term algebra, Basin and Cremers [15] present models and tool support for compromising adversaries, based on Scyther [37].

Küsters and Truderung [6] give a transformation that, given a Horn theory modeling secrecy and simple authentication properties modulo a DH theory with inverses, produces a Horn theory in the free algebra, which they analyze using ProVerif. Their reduction is similar to our reduction from E_{DH} to AC, but works only for Horn clauses with ground exponents. As stated in [6], stronger security properties often violate this restriction. Since our approach allows for non-ground exponents, we can also find attacks where the adversary sends products, e.g., a protocol that receives a message x and leaks a secret if $\mathsf{g}\hat{}(\mathsf{a}*\mathsf{b}*\mathsf{x}^{-1}) = \mathsf{g}$.

Lynch and Meadows [9] and Mödersheim [10] give reductions for DH reasoning without inverses to reasoning modulo a simpler equational theory for a restricted class of protocols. Both require that all exponents used by a protocol remain secret forever. This excludes modeling ephemeral key reveals and thus verifying recent AKE protocols. Ngo et. al. [11] propose a method for the automated construction of computational proofs for a restricted class of DH-based protocols.

IX. Conclusion

We gave a novel constraint-solving algorithm and demonstrated its effectiveness in non-trivial case studies. Our algorithm exploits a special representation of message deduction that satisfies a "deconstruction chain" property, inspired by the theory underlying Athena [38] and Scyther [37], [39]. This property is key for achieving unbounded verification. It enables a backwards exploration of the interleavings of protocol steps guided by the solving of message deduction constraints. We constructed this representation by imposing normal-form conditions on the use of the (finite) variants of the message deduction rules. This construction and our algorithm should also work for other theories with the finite variant property, e.g., theories for XOR and blind-signatures.

Although we were motivated by the verification of AKE protocols, neither our protocol model nor our constraint-solving algorithm are tailored to them. We expect both our model and algorithm to be applicable to a wide range of security protocol verification problems. Due to our multiset-rewriting model, our approach is especially promising for verifying protocols that exploit internal state, which are often hard to analyze using the Horn-theory approach. We plan to investigate such stateful protocols in future work together with support for loop invariants.

References

[1] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *Provable Security*, ser. LNCS, vol. 4784. Springer, 2007, pp. 1–16.

[2] S. Blake-Wilson and A. Menezes, "Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol," in *Proceedings of the 2nd International Workshop on Practice and Theory in Public Key Cryptography*. Springer, 1999, pp. 154–170.

[3] C. Cremers, "Session-StateReveal is stronger than eCK's EphemeralKeyReveal: Using automatic analysis to attack the NAXOS protocol," *International Journal of Applied Cryptography (IJACT)*, vol. 2, pp. 83–99, 2010.

[4] K. Lauter and A. Mityagin, "Security analysis of KEA authenticated key exchange protocol," in *PKC 2006*, ser. LNCS, vol. 3958. Springer, 2006, pp. 378–394.

[5] M. Just and S. Vaudenay, "Authenticated multi-party key agreement," in *ASIACRYPT 1996*, ser. LNCS, vol. 1163. Springer, 1996, pp. 36–49.

[6] R. Küsters and T. Truderung, "Using ProVerif to analyze protocols with Diffie-Hellman exponentiation," in *Proceedings of the 22nd IEEE Computer Security Foundations Symposium*. IEEE Computer Society, 2009, pp. 157–171.

[7] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani, "Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents," in *FSTTCS 2003*, ser. LNCS, vol. 2914. Springer, 2003, pp. 124–135.

[8] V. Shmatikov, "Decidable analysis of cryptographic protocols with products and modular exponentiation," in *Programming Languages and Systems*, ser. LNCS. Springer, 2004, vol. 2986, pp. 355–369.

[9] C. Lynch and C. Meadows, "Sound approximations to Diffie-Hellman using rewrite rules," in *Information and Communications Security*, ser. LNCS. Springer, 2004, vol. 3269, pp. 65–70.

[10] S. Mödersheim, "Diffie-Hellman without difficulty," in *Proceedings of the 8th International Workshop on Formal Aspects of Security & Trust (FAST)*, 2011.

[11] L. Ngo, C. Boyd, and J. Nieto, "Automated proofs for Diffie-Hellman-based key exchanges," in *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*, 2011, pp. 51 –65.

[12] 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, 2008.

[13] J. Goubault-Larrecq, M. Roger, and K. Verma, "Abstraction and resolution modulo AC: How to verify Diffie-Hellman-like protocols automatically," *Journal of Logic and Algebraic Programming*, vol. 64, no. 2, pp. 219–251, 2005.

[14] S. Escobar, C. Meadows, and J. Meseguer, "State space reduction in the Maude-NRL protocol analyzer," in *Computer Security - ESORICS 2008*, ser. LNCS. Springer, 2008, vol. 5283, pp. 548–562.

[15] D. Basin and C. Cremers, "Modeling and analyzing security in the presence of compromising adversaries," in *Computer Security - ESORICS 2010*, ser. LNCS, vol. 6345. Springer, 2010, pp. 340–356.

[16] ——, "Degrees of security: Protocol guarantees in the face of compromising adversaries," in *19th EACSL Annual Conference on Computer Science Logic (CSL)*, ser. LNCS, vol. 6247. Springer-Verlag, 2010, pp. 1–18.

[17] H. Andréka, I. Németi, and J. van Benthem, "Modal languages and bounded fragments of predicate logic," *Journal of Philosophical Logic*, vol. 27, pp. 217–274, 1998.

[18] B. Schmidt, S. Meier, C. Cremers, and D. Basin, "Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties (Extended Version)," April 2012, available http://www.infsec.ethz.ch/research/software#TAMARIN.

[19] S. Meier and B. Schmidt, "The TAMARIN prover: source code and case studies," April 2012, available http://hackage.haskell.org/package/tamarin-prover-0.4.0.0.

[20] H. Krawczyk, "HMQV: A high-performance secure Diffie-Hellman protocol," in *Advances in Cryptology–CRYPTO 2005*, ser. LNCS, vol. 3621. Springer, 2005, pp. 546–566.

[21] C. Cremers, "Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ACM, 2011, pp. 80–91.

[22] I. R. Jeong, J. Katz, and D. H. Lee, "One-round protocols for two-party authenticated key exchange," in *Applied Cryptography and Network Security*. Springer, 2004, pp. 220–232.

[23] F. J. Thayer, J. C. Herzog, and J. D. Guttman, "Strand spaces: Proving security protocols correct," *Journal of Computer Security*, vol. 7, no. 1, 1999.

[24] S. Escobar, R. Sasse, and J. Meseguer, "Folding variant narrowing and optimal variant termination," *Rewriting Logic and Its Applications*, pp. 52–68, 2010.

[25] H. Comon-Lundh and S. Delaune, "The finite variant property: How to get rid of some algebraic properties," *Term Rewriting and Applications*, pp. 294–307, 2005.

[26] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, "An efficient protocol for authenticated key agreement," *Designs, Codes and Cryptography*, vol. 28, pp. 119–134, 2003.

[27] E. Clarke, S. Jha, and W. Marrero, "Verifying security protocols with Brutus," *ACM Trans. Softw. Eng. Methodol.*, vol. 9, no. 4, pp. 443–487, 2000.

[28] S. Blake-Wilson and A. Menezes, "Authenticated Diffie-Hellman key agreement protocols," in *Selected Areas in Cryptography*, ser. LNCS. Springer, 1999, vol. 1556, pp. 630–630.

[29] I. R. Jeong, J. Katz, and D. H. Lee, "One-round protocols for two-party authenticated key exchange (full)," 2008, http://www.cs.umd.edu/~jkatz/papers/1round_AKE.pdf.

[30] *SP 800-56B, Special Publication 800-56B, Recommendation for Pair-Wise Key Establishment Schemes using Integer Factorization Cryptography*, National Institute of Standards and Technology, August 2009.

[31] S. Chatterjee, A. Menezes, and B. Ustaoglu, "A generic variant of NIST's KAS2 key agreement protocol," in *Proceedings of the 16th Australasian conference on Information security and privacy (ACISP'11)*. Springer, 2011, pp. 353–370.

[32] R. Corin, A. Saptawijaya, and S. Etalle, "A logic for constraint-based security protocol analysis," in *Proceedings of IEEE Symposium on Security and Privacy, S&P 2006*, 2006, pp. 155–168.

[33] A. Armando, R. Carbone, and L. Compagna, "LTL model checking for security protocols," in *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*, 2007, pp. 385 – 396.

[34] C. Meadows and P. Narendran, "A unification algorithm for the Group Diffie-Hellman protocol," in *Proc. of WITS 2002*, 2002.

[35] D. Kapur, P. Narendran, and L. Wang, "An E-unification algorithm for analyzing protocols that use modular exponentiation," in *Rewriting Techniques and Applications*. Springer, 2003, pp. 165–179.

[36] B. Blanchet, "An efficient cryptographic protocol verifier based on Prolog rules," in *Proceedings of the 14th IEEE Computer Security Foundations Workshop*. IEEE, 2001, pp. 82–96.

[37] C. Cremers, "The Scyther Tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification (CAV)*, ser. LNCS, vol. 5123. Springer, 2008, pp. 414–418.

[38] D. Song, S. Berezin, and A. Perrig, "Athena: A novel approach to efficient automatic security protocol analysis," *Journal of Computer Security*, vol. 9, pp. 47–74, 2001.

[39] S. Meier, C. Cremers, and D. A. Basin, "Strong invariants for the efficient construction of machine-checked protocol security proofs," in *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*. IEEE Computer Society, 2010, pp. 231–245.