Automated Verification of Group Key Agreement Protocols

Benedikt Schmidt
IMDEA Software Institute
Madrid, Spain

Ralf Sasse
Institute of Information Security
Dept. of Computer Science
ETH Zurich, Switzerland

Cas Cremers
Dept. of Computer Science
University of Oxford, UK

David Basin
Institute of Information Security
Dept. of Computer Science
ETH Zurich, Switzerland

Abstract—**We advance the state-of-the-art in automated symbolic cryptographic protocol analysis by providing the first algorithm that can handle Diffie-Hellman exponentiation, bilinear pairing, and AC-operators. Our support for AC-operators enables protocol specifications to use multisets, natural numbers, and finite maps. We implement the algorithm in the TAMARIN prover and provide the first symbolic correctness proofs for group key agreement protocols that use Diffie-Hellman or bilinear pairing, loops, and recursion, while at the same time supporting advanced security properties, such as perfect forward secrecy and eCK-security. We automatically verify a set of protocols, including the STR, group Joux, and GDH protocols, thereby demonstrating the effectiveness of our approach.**

I. INTRODUCTION

Key exchange protocols are a core building block for secure communication. They allow participants to establish a shared symmetric key, which can in turn be used with primitives such as symmetric encryption or message authentication codes, for secure communication. Most key exchange protocols are designed for two participants. However, in many scenarios, such as video conferencing and secure group communication, we would like efficient protocols to establish a shared key among an arbitrary number of parties. Moreover, the protocol should have strong security guarantees such as perfect forward secrecy. This is the problem addressed by *group key agreement* protocols, e.g. [1]–[3].

State-of-the-art group key agreement protocols are difficult to analyze automatically because they should work with arbitrarily many participants and they typically combine cryptographic operations such as Diffie-Hellman exponentiation and bilinear pairing with loops and mutable global state. The current best practice to establish security guarantees for such protocols is by pen-and-paper cryptographic proofs. This approach is extremely valuable, but given the complexity of group key agreement protocols and their advanced security guarantees, it is also time-consuming and error-prone. For example, manual analysis performed in [4] uncovered flaws in the group protocols of the CLIQUES family [5], which extends the protocols given in [3] and whose security was claimed to follow from the proofs in [3]. It would clearly be desirable to be able to apply methods and tools from Formal Methods to support, and where possible automate, reasoning in this domain.

There have been initial efforts at using symbolic methods to analyze some basic group key agreement protocols that do not use Diffie-Hellman or bilinear pairing. For example, the CORAL tool [6] was used to find several attacks on three such protocols. Similarly, the ProVerifList tool [7], a variant of ProVerif that supports unbounded lists but no equational theories, was used to prove a secrecy property of the Asokan-Ginzboorg protocol. None of these methods support the combination of AC-operators, which are needed to model group aspects, and Diffie-Hellman/bilinear pairing, which are the cryptographic primitives used to establish strong security guarantees.

Contributions: Our main contributions are twofold. First, we advance the state-of-the-art in automated symbolic analysis by providing the first analysis algorithm that can handle Diffie-Hellman exponentiation, bilinear pairing, and AC-operators. The AC-operators enable us to model protocols that rely on multisets or natural numbers, or use finite maps. For example, we use finite maps to represent the trees of unbounded depth used in group protocols. Our approach allows both verification and falsification, in which case it generates attack traces. We implement our algorithm in the TAMARIN prover, thereby enabling the analysis of protocols that were previously outside the scope of automated symbolic analysis tools.

Second, we use the resulting tool to provide the first symbolic verification results for group key agreement protocols that use Diffie-Hellman or bilinear pairing, loops, and recursion. Our results include the automated verification of identity-based protocols, tripartite group protocols, as well as the STR [1], group Joux [2], and GDH [3] protocols. Taken together, they show that our approach is effective and efficient: the analysis times are on the order of seconds to minutes.

Organization: In Section II we introduce background on group key protocols and we provide background on TAMARIN in Section III. In Section IV we present the tool extensions we developed. We explain our group protocol models and their analysis in Section V. We describe related work in Section VI and draw conclusions in Section VII. Readers mainly interested in the case studies in Section V are recommended to first read the protocol descriptions in Section II-B.

II. BACKGROUND ON GROUP PROTOCOLS

A. DH exponentiation and bilinear pairing

A *Diffie-Hellman group* $\mathbb{G} = \langle g \rangle$ is a cyclic group of prime order with generator g. We use multiplicative notation for $\mathbb{G}$ and denote the n-fold product of g with g^n. We consider groups

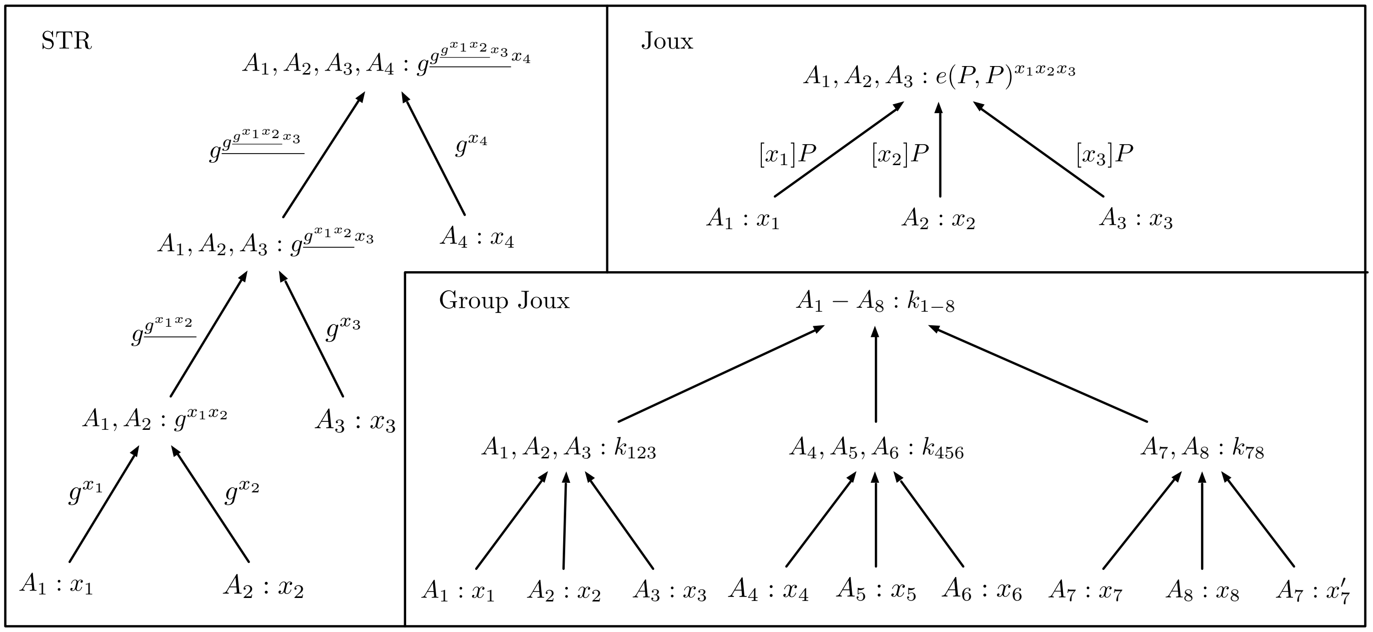

Fig. 1. The STR, Joux, and Group Joux protocols.

with a hard *computational Diffie-Hellman problem*, i.e., given g^n and g^m for random n and m, it is hard to compute g^{nm}.

A *bilinear group* is a triple $(\mathbb{G}, \mathbb{G}_T, \hat{e})$ such that $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of prime order and $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a non-degenerate bilinear map. We use additive notation for $\mathbb{G}$ and denote the n-fold sum of $P \in \mathbb{G}$ with $[n]P$. We use multiplicative notation for $\mathbb{G}_T$ and denote the n-fold product of $g \in \mathbb{G}_T$ with g^n. Since $\hat{e}$ is bilinear and non-degenerate, $\hat{e}([n]P, [m]P) = \hat{e}(P,P)^{nm}$ and $\hat{e}(P,Q)$ is a generator of $\mathbb{G}_T$ if P and Q are generators of $\mathbb{G}$.

B. Group key agreement

Group key agreement protocols allow a group of participants to agree on a shared key. Unlike with key transport protocols, the participants need *not* rely on a central server. Every member contributes to the group key and thus is assured of the key's freshness. We consider protocols that can have an arbitrary number of participants. In the following, we call a randomly sampled private value (typically x or y) used in a particular protocol session an *ephemeral secret key* and the associated public value an *ephemeral public key* (typically g^x or g^y for DH-based protocols).

STR: As a first example of a group key agreement protocol consider the *STR* protocol [1,8]. In STR, the standard Diffie-Hellman (DH) exchange is repeatedly applied for subgroups with an additional member in each iteration. The members comprise the leaves of a maximally unbalanced tree, as depicted in Figure 1. Then DH is applied to the two participants at the lowest leaves, and at all other levels DH is applied between an owner of the previous subgroup key and the next participant. In the figure, x_i represents the ephemeral secret key of participant A_i, leaves represent participants that send their ephemeral public key g^{x_i}, and inner nodes represent subgroup keys. For example, participant A_1 generates x_1, sends g^{x_1} and receives g^{x_2}, which allows A_1 and A_2 to compute the first subgroup key $g^{x_1x_2}$. Next, participant A_3 generates x_3 and sends g^{x_3}, while participant A_1 sends the exponentiated shared subgroup key, $g^{\underline{g^{x_1x_2}}}$. Here, we underline the term $g^{x_1x_2}$ to denote that it is the result of converting a group element to an integer, which is required to use it as exponent. After exchanging these messages, all three participants can compute the subgroup key $g^{\underline{g^{x_1x_2}}x_3}$. Finally, participant A_4 generates x_4 and sends g^{x_4}. Participant A_1 sends $g^{\underline{g^{\underline{g^{x_1x_2}}x_3}}}$ and all four participants can compute the group key $g^{\underline{g^{\underline{g^{x_1x_2}}x_3}}x_4}$. We will say that a protocol *has subgroup keys* when all subgroups have their own key.

Joux: The *tripartite Joux* protocol is a three-party variant of Diffie-Hellman based on bilinear pairing. We consider this as a special case of group key agreement for just 3 participants. The Joux protocol generates the shared key for all 3 members in a single round, as shown in Figure 1: Participant A_1 picks an ephemeral secret key, say x_1, and computes and broadcasts $X_1 = [x_1]P$. Afterwards A_1 receives X_2 and X_3 from the other two participants and computes the joint shared key as $\hat{e}(X_2, X_3)^{x_1} = \hat{e}([x_2]P, [x_3]P)^{x_1} = \hat{e}(P,P)^{x_1x_2x_3}$. The other two participants act analogously.

Group Joux: The Joux protocol can be extended to the *group Joux* protocol, which works for arbitrarily many participants. Group Joux constructs a balanced tree of repeated applications of the basic tripartite Joux protocol. At the leaves it is run by three participants, and at inner nodes it is used for three subgroups, each of which already has a subgroup key. The groups and the flow of keying material are depicted in Figure 1. If the number of participants is not a power of three, then the protocol works with a minor change. Whenever a node in the tree only has two children, those children do the basic Joux exchange with the left child contributing a second, dummy, key share. All numbers of participants can be dealt with using a similar rearrangement of the tree.

As an example, consider group Joux for 8 participants, A_1 through A_8. We order them as shown in the leaves of Figure 1. Thus, A_1, A_2, and A_3 execute the normal Joux exchange as described above and share the subgroup key k_{123}. Similarly A_4, A_5, and A_6 get the key k_{456}. For A_7 and A_8, the left participant A_7 also generates a dummy value x'_7, so they can create a key k_{78}. At the next level, one representative for each group, say the leftmost one, i.e., A_1, A_4, and A_7 repeat the process. This time, instead of picking new randomness, they use some (publicly known) derivation function that maps the subgroup key to a value usable for another round of the basic Joux protocol. Finally they execute the basic Joux protocol again and get the group key k_{1-8}.

GDH: In GDH, each participant receives i messages of the form g^X from its predecessor, picks an ephemeral secret key x_i, exponentiates all received messages with x_i and sends them to its successor. Additionally, it forwards the last received message. The first participant A_1 is treated as having received the generator g. The last participant also does not send the last message (which is the group key) and either broadcasts all messages or sends each to the appropriate recipient. With the message from the last participant, each group member can compute the group key $g^{x_1 x_2 \cdots x_n}$.

We present a small example with 4 participants in Figure 2. Each row represents the messages sent by a participant, and the columns show the evolution of the message needed by the participant listed in the header. We mark the *round key* for each level with † and the final group key is $g^{x_1x_2x_3x_4}$. Only the final group key must stay secret and all intermediate keys are sent out.

The figure also depicts the flow of messages. Each participant performs the following actions. Participant A_1 picks its ephemeral secret key x_1 and 'receives' g. It sends out g as it is the last received message as well as g^{x_1}. Participant A_2 picks ephemeral secret key x_2, receives g, and thus sends g^{x_2}. It also receives g^{x_1} as the last message, so it sends g^{x_1} back out as well as $g^{x_1x_2}$. Participant A_3 then picks x_3, receives g^{x_2}, and sends $g^{x_2x_3}$. It receives g^{x_1} and sends $g^{x_1x_3}$. Finally it receives $g^{x_1x_2}$, so it sends $g^{x_1x_2}$ and $g^{x_1x_2x_3}$. The final participant A_4 picks x_4, receives $g^{x_2x_3}$, and sends $g^{x_2x_3x_4}$. On receiving $g^{x_1x_3}$, it sends $g^{x_1x_3x_4}$. After receiving $g^{x_1x_2}$, it sends $g^{x_1x_2x_4}$. The last message received is $g^{x_1x_2x_3}$, which allows A_4 to compute the group secret $g^{x_1x_2x_3x_4}$. Sending $g^{x_1x_2x_3}$ (done when not the last member) is not needed as that would only be for its own use. Of course, A_4 does *not* send out the group shared key. All the other participants know their ephemeral secret key x_i (with $1 \le i < 4$) and can therefore compute the group shared key after receiving their designated message from A_4. Their designated message only lacks their secret, e.g., A_2's message is $g^{x_1x_3x_4}$.

$$\begin{tikzcd}
 & \text{for } A_1 & \text{for } A_2 & \text{for } A_3 & \text{for } A_4 & \text{group key}\dagger \\
A_1: & g \arrow[d] & g^{x_1}\dagger \arrow[d] \arrow[rd] & & & \\
A_2: & g^{x_2} \arrow[d] & g^{x_1} \arrow[d] & g^{x_1x_2}\dagger \arrow[d] \arrow[rd] & & \\
A_3: & g^{x_2x_3} \arrow[d] & g^{x_1x_3} \arrow[d] & g^{x_1x_2} \arrow[d] & g^{x_1x_2x_3}\dagger \arrow[d] \arrow[rd] & \\
A_4: & g^{x_2x_3x_4} & g^{x_1x_3x_4} & g^{x_1x_2x_4} & g^{x_1x_2x_3} & g^{x_1x_2x_3x_4}\dagger
\end{tikzcd}$$

Fig. 2. GDH

III. Background on the Tamarin prover

In this section, we describe the Tamarin prover [9,10] and its support for induction [11]. We present the implementation of our algorithms as Tamarin extensions in Section IV, so that the protocols from Section II-B can be analyzed with the help of the extended tool. This section contains several simplifications of the theory and algorithm from [9]. Full details of this and other protocols are given in [12], with the implementation and all case studies available at [13].

A. Security protocol model

In Tamarin, the execution of a security protocol in the context of an adversary is modeled 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.

Cryptographic messages: To model cryptographic messages, we use an order-sorted term algebra and an equational theory. The function symbols in the signature model the algorithms that can be applied to messages, such as encryption or decryption, and the equational theory models the properties of the algorithms. We use the partially ordered set of sorts consisting of the top sort *msg* and two incomparable subsorts *fr* and *pub*. We use *fr* for fresh names modeling random values such as keys or nonces and *pub* for public names modeling known constants such as agent identities. To model DH exponentiation, Tamarin uses the signature

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

where $x\,\hat{}\,y$ denotes the exponentiation of the base x to the power y, and the remaining operators model multiplication, multiplicative unit, multiplicative inverse in $\mathbb{Z}_q^\times$ (modeling the corresponding operations on exponents), pairing, and projection. The properties of these operators are modeled by the equational theory generated by

$$\begin{aligned} E_{DH} = \{ & (x\,\hat{}\,y)\,\hat{}\,z = x\,\hat{}\,(y*z),\ x\,\hat{}\,1 = x,\ x*y = y*x, \\ & (x*y)*z = x*(y*z),\ x*1 = x,\ x*x^{-1} = 1, \\ & \mathrm{fst}(\langle x,y\rangle) = x,\ \mathrm{snd}(\langle x,y\rangle) = y\}. \end{aligned}$$

The first two equations capture the properties of exponentiation, the last two equations model projection, and the remaining equations model the exponents as an abelian group. To model additional cryptographic operators, such as signature schemes or (a)symmetric encryption, the user can specify a disjoint subterm-convergent equational theory. See [12] for details.

Note that TAMARIN does not model multiplication of group elements and addition of exponents. Full support for these operations would require modeling the exponents as a field, for which no unification algorithm is known. Furthermore, there are undecidability results for unification in theories that are more expressive than E_{DH} [14].

Execution state: We model the states of our transition system as finite multisets of facts. Facts are built from terms over a 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 $F(t_1,\ldots,t_k)$ such that t_i is a term and F is a k-ary fact symbol. We denote the set of ground facts by $\mathcal{G}$. We say that a fact $F(t_1,\ldots,t_k)$ is linear if F is linear and it is 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. We assume that the linear fact symbol Fr is included in Σ_{Fact}. The semantics of Fr is fixed and the fact $\mathsf{Fr}(n)$ denotes that the name n is freshly generated. The semantics of the remaining fact symbols are defined by the multiset rewriting rules that generate and consume them. In our examples, we prefix all persistent protocol fact symbols with the ! symbol.

Rules: We use labeled multiset rewriting to define the possible transitions of our transition system. A labeled multiset rewriting rule is a triple (l,a,r) with $l,a,r \in \mathcal{F}^*$, denoted $l\text{-}[\,a\,]\!\rightarrow r$. For $ru = l\text{-}[\,a\,]\!\rightarrow r$, we define the premises as $\mathit{prems}(ru) = l$, the actions as $\mathit{acts}(ru) = a$, and the conclusions as $\mathit{concs}(ru) = r$. We call a set of labeled multiset rewriting rules a labeled multiset rewriting system. In the following, we often drop the "labeled" qualifier.

Fresh name generation is handled by the rule FRESH, which is defined as $[]\text{-}[]\!\rightarrow \mathsf{Fr}(x{:}\mathit{fr})$. To define the protocol and adversary rules, we assume that Σ_{Fact} includes the persistent fact symbol K modeling messages known to the adversary, the linear fact symbol Out modeling messages sent by the protocol, and the linear fact symbol In modeling messages sent by the adversary. The adversary's message deduction capabilities are captured by the following set of rules.

$$\begin{aligned}\mathit{MD} = \{\ & \mathsf{Out}(x)\text{-}[]\!\rightarrow \mathsf{K}(x),\ \mathsf{K}(x)\text{-}[\,\mathsf{K}(x)\,]\!\rightarrow \mathsf{In}(x),\\ & \mathsf{Fr}(x{:}\mathit{fr})\text{-}[]\!\rightarrow \mathsf{K}(x{:}\mathit{fr}),\ []\text{-}[]\!\rightarrow \mathsf{K}(x{:}\mathit{pub})\ \}\\ \cup\ \{\ & \mathsf{K}(x_1),\ldots,\mathsf{K}(x_n)\text{-}[]\!\rightarrow \mathsf{K}(f(x_1,\ldots,x_n)) \mid f \in \Sigma^n\ \}\end{aligned}$$

Note that we will define the semantics of the message deduction rules modulo E_{DH}. This allows the adversary to deduce, for example, x from x^{-1} since $(x^{-1})^{-1} =_{E_{DH}} x$.

The protocol rules are user-defined and must satisfy the following conditions.

1) All variables that occur in conclusions are bound in premises (except for public variables).
2) They respect fresh name creation, i.e., fresh names cannot occur in rules and Fr can only be used in the premises.
3) They respect the semantics of K, In, and Out, i.e., K cannot be used, In can only be used in the premises, and Out can only be used in the conclusions.
4) Multiplication in the group of exponents cannot be directly used.

Their usage of protocol-specific facts is unrestricted.

Labeled operational semantics: To define the labeled operational semantics of a multiset rewrite system R, we first define the labeled transition relation

$$\mathit{steps}(R) \subseteq \mathcal{G}^\sharp \times \mathit{ginsts}_{E_{DH}}(R \cup \{\text{FRESH}\}) \times \mathcal{G}^\sharp$$

as

$$\frac{\begin{array}{c} l\text{-}[\,a\,]\!\rightarrow r \in \mathit{ginsts}_{E_{DH}}(R \cup \{\text{FRESH}\}) \\ S' = (S \setminus^\sharp \mathit{lfacts}(l)) \cup^\sharp r \\ \mathit{lfacts}(l) \subseteq^\sharp S \qquad \mathit{pfacts}(l) \subseteq S \end{array}}{(S, l\text{-}[\,a\,]\!\rightarrow r, S') \in \mathit{steps}(R)}\ ,$$

where each transition is labeled with the applied rule instance. We use $\mathcal{G}^\sharp$ to denote the set of all (finite) multisets of ground facts, $\mathit{ginsts}_{E_{DH}}$ to denote the set of ground E_{DH}-instances of the given rules, $\sharp$ to denote the multiset equivalents of the corresponding set operations, and $\mathit{lfacts}(l)$ (respectively $\mathit{pfacts}(l)$) to denote the multisets of linear (respectively persistent) facts in l. A transition rewrites the state S with a ground instance of FRESH or a rule from R. Since we perform multiset rewriting modulo E_{DH}, any applicable ground E_{DH}-instance of a rule in $R \cup \{\text{FRESH}\}$ can be used. An instance $l\text{-}[\,a\,]\!\rightarrow r$ is applicable to S if the multiset of linear facts in l is included in S with respect to multiset inclusion and the set of persistent facts in l is included in S with respect to set inclusion. To obtain the successor state S', the consumed linear facts are removed and the generated facts are added.

An execution of R is an alternating sequence

$$e = [S_0, (l_1\text{-}[\,a_1\,]\!\rightarrow r_1), S_1, \ldots, S_{k-1}, (l_k\text{-}[\,a_k\,]\!\rightarrow r_k), S_k]$$

of states and multiset rewriting rule instances such that

1) $S_0 = \varnothing^\sharp$,
2) $(S_{i-1}, (l_i\text{-}[\,a_i\,]\!\rightarrow r_i), S_i) \in \mathit{steps}(R)$, for $1 \le i \le k$,
3) if $r_i = r_j = [\mathsf{Fr}(n)]$, then $i = j$.

We denote the set of executions of R with $\mathit{execs}(R)$. We define the trace of such an execution e as

$$\mathit{trace}(e) = [\mathit{set}(a_1), \ldots, \mathit{set}(a_k)].$$

The trace is the sequence of sets of actions of the multiset rewriting rule instances. We define the observable trace $\overline{tr}$ of a trace tr as the subsequence of all actions in tr that are not equal to $\varnothing$.

Trace formulas: To specify security properties, we use sorted first-order formulas over the atoms $f@i$, $t \doteq t'$, and $i \lessdot j$, where f is a fact, t, t' are terms, and i, j are variables of a new sort *temp* that models timepoints. For an equational theory E, the satisfaction relation $(tr, \sigma) \vDash_E \varphi$ between traces, variable assignments, and formulas is defined as follows. For a trace

tr and variable assignment σ, the atom $f@i$ is satisfied if the $f\sigma \in_E tr_{\sigma(i)}$, the atom $t \doteq t'$ is satisfied if $\sigma(t) =_E \sigma(t')$, and the atom $i \lessdot j$ is satisfied if $\sigma(i) < \sigma(j)$. The satisfaction relation is then extended to the logical operators and quantifiers as usual. A protocol P satisfies a security property φ, written $P \vDash_E \varphi$, if for all traces $tr \in \mathit{trace}(\mathit{execs}(P \cup \mathit{MD}))$ and all variable assignments σ, it holds that $(tr, \sigma) \vDash_E \varphi$.

B. Dependency Graphs

Instead of directly working in the multiset rewriting semantics, TAMARIN uses symbolic reasoning modulo AC using so-called *dependency graphs*. Dependency graphs consist of the sequence of rewriting rule instances corresponding to a protocol execution and their causal dependencies, similar to strand spaces [15]. Schmidt et al. [9] showed that dependency graphs modulo AC can be used instead of dependency graphs modulo E_{DH}. Additionally, TAMARIN exploits specific normal message deductions and the corresponding normal dependency graphs. Normal dependency graphs are weakly trace equivalent to the multiset rewriting semantics in that they have the same observable traces.

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

$$\begin{aligned} P = \{\ &[\mathsf{Fr}(x), \mathsf{Fr}(y)] \\ &\quad -[\ \mathsf{Start}()\ ]\!\!\rightarrow [\mathsf{St}(x), \mathsf{Out}(\langle (\mathsf{g}\,\hat{}\,x)\,\hat{}\,y, y^{-1}\rangle)] \\ ,\ &[\mathsf{St}(x), \mathsf{In}(\mathsf{g}\,\hat{}\,x)] -[\ \mathsf{Fin}()\ ]\!\!\rightarrow [\,]\}, \end{aligned}$$

which models a message deduction problem. Figure 3 shows a dependency graph for an execution of P. Rule instances are represented using inference rule notation with the actions on the right. Nodes 1 and 2 are rule instances that create fresh names. Node 3 is an instance of the first protocol rule. Node 9 is an instance of the second protocol rule. Nodes 4–8 are instances of message deduction rules and denote that the adversary receives a pair of a group element and an exponent, raises the first component to the power of the second component, and sends the result to an instance of the second protocol rule, Node 9. 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}, all nodes are ground E_{DH}-instances of rules from $P \cup \mathit{MD} \cup \{\text{FRESH}\}$.

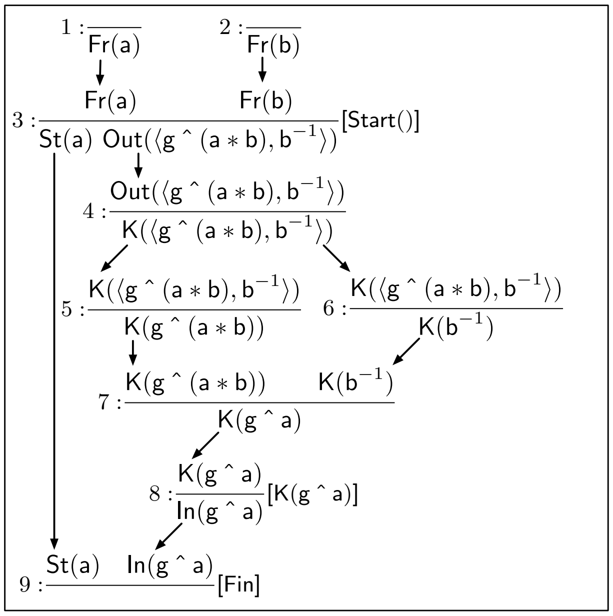
1 : Fr(a)
2 : Fr(b)
Fr(a) Fr(b)
3 : St(a) Out(⟨g ^ (a ∗ b), b⁻¹⟩) [Start()]
4 : Out(⟨g ^ (a ∗ b), b⁻¹⟩) / K(⟨g ^ (a ∗ b), b⁻¹⟩)
5 : K(⟨g ^ (a ∗ b), b⁻¹⟩) / K(g ^ (a ∗ b))
6 : K(⟨g ^ (a ∗ b), b⁻¹⟩) / K(b⁻¹)
7 : K(g ^ (a ∗ b)) K(b⁻¹) / K(g ^ a)
8 : K(g ^ a) / In(g ^ a) [K(g ^ a)]
9 : St(a) In(g ^ a) [Fin]

Fig. 3. Dependency graph modulo E_{DH}.

Formally, let E be an equational theory and R be a set of multiset rewriting rules. $dg = (I, D)$ is a *dependency graph modulo E for R* if $I \in (\mathit{ginsts}_E(R \cup \{\text{FRESH}\}))^*$, $D \subseteq \mathbb{N}^2 \times \mathbb{N}^2$, and dg satisfies the conditions **DG1–4** listed below. These conditions employ the following definitions. We call $\{1, \ldots, |I|\}$ the *nodes* and D the *edges* of dg. $(i, u) \rightarrowtail (j, v)$ denotes 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 $u \in \{1, \ldots, |r_i|\}$. 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 $u \in \{1, \ldots, |l_i|\}$. The corresponding *premise fact* is $(l_i)_u$. A conclusion or premise is *linear* if its fact is linear.

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

The set of all dependency graphs modulo E for R is denoted by $\mathit{dgraphs}_E(R)$. For all multiset rewriting systems R, the multiset rewriting semantics given in the previous section and dependency graphs modulo E_{DH} have the same set of traces, i.e., $\mathit{trace}(\mathit{execs}(R)) = \mathit{trace}(\mathit{dgraphs}_{E_{DH}}(R))$.

C. Normal Dependency Graphs modulo AC

TAMARIN's effectiveness depends on two main elements: the reduction of reasoning modulo E_{DH} to reasoning modulo AC and the use of normal forms during deduction. To perform the reduction to reasoning modulo AC, we use the presentation $DH{,}AC$ of E_{DH} by a convergent rewriting system DH and the equational theory AC. We exploit that this presentation has the finite variant property, which means that for all terms t, there is a finite set $\lceil t \rceil^{DH}$ with the following property: for all E_{DH}-instances s of t, there exists a $t' \in \lceil t \rceil^{DH}$ such that s is an AC-instance of t'. Intuitively, the complete set of variants $\lceil t \rceil^{DH}$ covers all instantiations of t and the finite variant property guarantees that there exists such a *finite* set for all terms t. In the following, we use $t{\downarrow}_{DH}$ to denote t's normal form with respect to $DH{,}AC$. For more details, including the definition of $DH{,}AC$, see [12]. Below we provide some details on the definitions of normal form deductions needed for our extensions in the next section.

We first introduce normal forms for message deduction. In particular, message deduction steps in dependency graphs modulo AC use rules from $\lceil \mathit{MD} \rceil^{DH}$. These rules still allow

Coerce rules: $\textsc{Coerce}\ \dfrac{\mathsf{K}^{\downarrow}_{y}(x)}{\mathsf{K}^{\uparrow}(x)}$ *Communication rules*: $\textsc{Irecv}\ \dfrac{\mathsf{Out}(x)}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x)}$ $\textsc{Isend}\ \dfrac{\mathsf{K}^{\uparrow}(x)}{\mathsf{In}(x)}[\mathsf{K}(x)]$

Construction rules:

$$\dfrac{\mathsf{K}^{\uparrow}(x)\ \ \mathsf{K}^{\uparrow}(y)}{\mathsf{K}^{\uparrow}(x\,\hat{}\,y)} \qquad \dfrac{}{\mathsf{K}^{\uparrow}(x{:}\mathit{pub})} \qquad \dfrac{\mathsf{Fr}(x{:}\mathit{fr})}{\mathsf{K}^{\uparrow}(x{:}\mathit{fr})} \qquad \dfrac{\mathsf{K}^{\uparrow}(x)}{\mathsf{K}^{\uparrow}(x^{-1})} \qquad \dfrac{}{\mathsf{K}^{\uparrow}(1)} \qquad \dfrac{\mathsf{K}^{\uparrow}(x)\ \ \mathsf{K}^{\uparrow}(y)}{\mathsf{K}^{\uparrow}(\langle x,y\rangle)}$$

$$\dfrac{\mathsf{K}^{\uparrow}(x)}{\mathsf{K}^{\uparrow}(\mathrm{fst}(x))} \qquad \dfrac{\mathsf{K}^{\uparrow}(x)}{\mathsf{K}^{\uparrow}(\mathrm{snd}(x))} \qquad \dfrac{\mathsf{K}^{\uparrow}(x_1)\ \ldots\ \mathsf{K}^{\uparrow}(x_n) \qquad \mathsf{K}^{\uparrow}(x_{n+1})\ \ldots\ \mathsf{K}^{\uparrow}(x_l)}{\mathsf{K}^{\uparrow}((x_1 * \ldots * x_n) * (x_{n+1} * \ldots * x_l)^{-1})}$$

Deconstruction rules:

$$\dfrac{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x\,\hat{}\,y)\ \ \mathsf{K}^{\uparrow}(y^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x)} \qquad \dfrac{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x\,\hat{}\,y^{-1})\ \ \mathsf{K}^{\uparrow}(y)}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x)} \qquad \dfrac{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x\,\hat{}\,(y * z^{-1}))\ \ \mathsf{K}^{\uparrow}(y^{-1} * z)}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x)}$$

$$\dfrac{\mathsf{K}^{\downarrow}_{\mathsf{d}}(\langle x,y\rangle)}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x)} \qquad \dfrac{\mathsf{K}^{\downarrow}_{\mathsf{d}}(\langle x,y\rangle)}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(y)} \qquad \dfrac{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x)}$$

Exponentiation rules:

$$\dfrac{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x\,\hat{}\,y)\ \ \mathsf{K}^{\uparrow}(z)}{\mathsf{K}^{\downarrow}_{\mathsf{e}}(x\,\hat{}\,(y * z))} \quad \dfrac{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x\,\hat{}\,y)\ \ \mathsf{K}^{\uparrow}(y^{-1} * z)}{\mathsf{K}^{\downarrow}_{\mathsf{e}}(x\,\hat{}\,z)} \quad \cdots \quad \dfrac{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x\,\hat{}\,(y * z^{-1}))\ \ \mathsf{K}^{\uparrow}(a * b^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{e}}(x\,\hat{}\,(y * a * (z * b)^{-1}))}$$

Fig. 4. Normal message deduction rules *ND*. We use y in COERCE to denote that the premise can be either $\mathsf{K}^{\downarrow}_{\mathsf{d}}(x)$ or $\mathsf{K}^{\downarrow}_{\mathsf{e}}(x)$. 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. We use ND^{destr} to denote the set of deconstruction and exponentiation rules.

redundant steps, some of which are eliminated by tagging the rules to limit their applicability.

$\lceil MD \rceil^{DH}$ is partitioned 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 want to enforce normal message deductions, which satisfy the following normal-form condition. All messages are deduced by extracting subterms from messages sent by the protocol, optionally modifying the exponent of extracted exponentiations, and finally applying function symbols to these messages. This allows us to search for message deductions by applying deconstruction and exponentiation rules to messages sent by the protocol top-down and applying construction rules to messages received by the protocol bottom-up until both meet in the middle. To achieve this, we introduce three new fact symbols. $\mathsf{K}^{\downarrow}_{\mathsf{d}}(m)$ denotes that m has been extracted from a message sent by the protocol. $\mathsf{K}^{\downarrow}_{\mathsf{e}}(m)$ denotes that m has been deduced by modifying the exponent of an extracted message. Finally, $\mathsf{K}^{\uparrow}(m)$ denotes that m has been deduced using a normal message deduction.

Figure 4 shows the *normal message deduction rules ND*. For example, the COERCE rule is used to switch from message deconstruction or exponentiation to message construction. The multiplication rules are replaced by l-ary construction rules for multiplication, which are sufficient to capture all possibilities under TAMARIN's restrictions.

Additionally, TAMARIN enforces further normal-form conditions by reasoning over *normal dependency graphs*, which use the normal message deduction rules. To state the dependency-graph normal-form conditions, we define the *non-inverse factors of a term* t as

$$\mathit{nifactors}(t) = \begin{cases} \mathit{nifactors}(a) \cup \mathit{nifactors}(b) & \text{if } t = a * b \\ \mathit{nifactors}(a) & \text{if } t = a^{-1} \\ \{t\} & \text{otherwise.} \end{cases}$$

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 conditions **N1**–**N6** in Figure 5 are satisfied.

Condition **N1** ensures that all rule instances are $\downarrow_{DH}$-normal. Condition **N2** formalizes that the adversary constructs all products directly by multiplying their components. Condition **N3** ensures that the same message never has multiple $\mathsf{K}^{\downarrow}_{y}$ or $\mathsf{K}^{\uparrow}$ deductions. Condition **N4** ensures that pairs and inverses are never deduced by COERCE. Condition **N5** forbids two types of redundancies. First, if there is already a normal deduction for a message, then there is no need for a later deconstruction of the same message. Second, if there is already a deconstruction of a message, then the COERCE rule should be used to create a normal deduction unless it is forbidden by condition **N4**. Condition **N6** forbids instances of exponentiation rules that can be directly replaced by the construction rule for exponentiation.

We write $ndgraphs(P)$ to denote the set of all normal dependency graphs of P. Critically, normal dependency graphs capture exactly the same observable traces as the multiset rewriting semantics.

N1 The dependency graph dg is $\downarrow_{DH}$-normal.

N2 There is no multiplication rule that has a premise fact of the form $\mathsf{K}^{\uparrow}(t * s)$ and all conclusion facts $\mathsf{K}^{\uparrow}(t * s)$ are conclusions of a multiplication rule.

N3 If there are two conclusions c and c' with conclusion facts $\mathsf{K}^{\downarrow}_{y}(m)$ and $\mathsf{K}^{\downarrow}_{y'}(m')$ or with conclusion facts $\mathsf{K}^{\uparrow}(m)$ and $\mathsf{K}^{\uparrow}(m')$ such that $m =_{AC} m'$, then $c = c'$.

N4 No instance of COERCE deduces a pair or an inverse.

N5 If there is a conclusion $(i, 1)$ with fact $\mathsf{K}^{\downarrow}_{y}(m)$ and a conclusion $(j, 1)$ with fact $\mathsf{K}^{\uparrow}(m')$ such that $m =_{AC} m'$, then $i < j$ and j is an instance of COERCE or the construction rule for pairing or the one for inversion.

N6 There is no node $\mathsf{K}^{\downarrow}_{\mathsf{d}}(a), \mathsf{K}^{\uparrow}(b) -[]\!\!\rightarrow \mathsf{K}^{\downarrow}_{\mathsf{e}}(c \hat{} d)$ such that c does not contain any fresh names and $\mathit{nifactors}(d) \subseteq_{AC} \mathit{nifactors}(b)$.

Fig. 5. TAMARIN's normal-form conditions

N7 There is no construction rule for $\sharp$ that has a premise of the form $\mathsf{K}^{\uparrow}(s \sharp t)$. All conclusion facts of the form $\mathsf{K}^{\uparrow}(s \sharp t)$ are conclusions of a construction rule for $\sharp$.

N8 The conclusion of a deconstruction rule for $\sharp$ is never of the form $\mathsf{K}^{\downarrow}_{\mathsf{d}}(s \sharp t)$.

N9 There is no node $[\mathsf{K}^{\downarrow}_{\mathsf{d}}(a), \mathsf{K}^{\uparrow}(b)] -[]\!\!\rightarrow \mathsf{K}^{\downarrow}_{\mathsf{e}}([d]c)$ such that c does not contain any fresh names and $\mathit{nifactors}(d) \subseteq_{ACC} \mathit{nifactors}(b)$.

N10 There is no node i labeled with $[\mathsf{K}^{\downarrow}_{\mathsf{d}}([t_1]p), \mathsf{K}^{\downarrow}_{\mathsf{d}}([t_2]q)] -[]\!\!\rightarrow \mathsf{K}^{\downarrow}_{\mathsf{d}}(\hat{e}(p,q) \hat{} u)$ such that there is a node j labeled with $[\mathsf{K}^{\downarrow}_{\mathsf{d}}(\hat{e}(p,q) \hat{} u), \mathsf{K}^{\uparrow}(v)] -[]\!\!\rightarrow \mathsf{K}^{\downarrow}_{\mathsf{d}}(\hat{e}(p,q) \hat{} w)$, an edge $(i,1) \rightarrowtail (j,1)$, $\mathit{nifactors}(t_i) \subseteq_{ACC} \mathit{nifactors}(v)$ for $i = 1$ or $i = 2$, and $\hat{e}(p,q)$ does not contain any fresh names.

N11 There is no node $[\mathsf{K}^{\downarrow}_{\mathsf{d}}([a]p), \mathsf{K}^{\downarrow}_{\mathsf{d}}([b]q)] -[]\!\!\rightarrow \mathsf{K}^{\downarrow}_{\mathsf{d}}(\hat{e}(p,q) \hat{} (a * b))$ such that the send-nodes of the first and second premise are labeled with ru_1 and ru_2 and $\mathit{fsyms}(ru_2) <_{fs} \mathit{fsyms}(ru_1)$.

Fig. 6. Our new normal-form conditions for AC operators and bilinear pairing. We explain these in Section IV-C.

Our search algorithm exploits the following fact about normal dependency graphs to reason about the possible origins of $\mathsf{K}^{\downarrow}_{y}$-premises. Let $dg = (I, D) \in \mathit{ndgraphs}(P)$ and define the *deconstruction chain relation* $\dashrightarrow_{dg}$ as the smallest relation such that $i \dashrightarrow_{dg} p$ if $(i,1)$ is a $\mathsf{K}^{\downarrow}_{y}$-conclusion in dg and (a) $(i,1) \rightarrowtail p \in D$ or (b) there is a premise $(j,1)$ in dg such that $(i,1) \rightarrowtail (j,1) \in D$ and $j \dashrightarrow_{dg} p$. Then it holds that for every $\mathsf{K}^{\downarrow}_{y}$-premise p of dg, there is a node i in dg such that $I_i \in \mathit{ginsts}_{acDH}(\text{IRECV})$ and $i \dashrightarrow_{dg} p$. In our search algorithm, this allows a forward search starting from a protocol send followed by IRECV to find the provider of a $\mathsf{K}^{\downarrow}_{y}$-premise.

D. TAMARIN*'s search algorithm*

We briefly review TAMARIN's algorithm, which tries to determine whether $P \vDash_{E_{DH}} \varphi$ for a protocol P and a trace property φ. The algorithm uses constraint solving to perform

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

Fig. 7. Pseudocode of the constraint solving algorithm.

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

1) Syntax and Semantics of Constraints: In the remainder of this section, let ri range over multiset rewriting rule instances, f over facts, i and j over temporal variables, u and v over natural numbers, and φ over guarded trace formulas. Intuitively, a trace formula is guarded if all quantified variables are guarded or bounded by fact atoms. 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 a *provides* $i \triangleright f$, which denotes that f is the first conclusion of the node i. A *constraint* is a graph constraint or a guarded trace formula.

A *structure* is a pair (dg, θ) of a dependency graph $dg = (I, D)$ and a valuation θ. The application of the homomorphic extension of θ to a rule instance ri is denoted by $ri\,\theta$. The structure (dg, θ) *satisfies* a constraint γ, written $(dg, \theta) \Vdash \gamma$ if:

$$\begin{array}{ll}
(dg,\theta) \Vdash i : ri & \text{iff } \theta(i) \in \{1,\ldots,|I|\} \text{ and } ri\,\theta =_{AC} I_{\theta(i)} \\
(dg,\theta) \Vdash (i,u) \rightarrowtail (j,v) & \text{iff } (\theta(i),u) \rightarrowtail (\theta(j),v) \in D \\
(dg,\theta) \Vdash (i,u) \dashrightarrow (j,v) & \text{iff } (\theta(i),u) \dashrightarrow_{dg} (\theta(j),v) \\
(dg,\theta) \Vdash i \triangleright f & \text{iff } \mathit{concs}(I_{\theta(i)})_1 =_{AC} f\theta \\
(dg,\theta) \Vdash \varphi & \text{iff } (\mathit{trace}(dg),\theta) \vDash_{AC} \varphi
\end{array}$$

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. (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 $(dg, \theta) \Vdash \Gamma$.

2) Constraint-Solving Algorithm: Let P be a protocol and φ a guarded trace property. The 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. The algorithm is based on the constraint-reduction relation $\rightsquigarrow_P$ between constraint systems and sets of constraint systems. Sets of constraint systems are used to represent case distinctions.

Intuitively, $\rightsquigarrow_P$ refines constraint systems and the algorithm refines the initial constraint system $\{\hat{\varphi}\}$ until it either encounters a solved system or all systems contain (trivially)

Example trace formula reduction rules:

$\mathsf{S}_{\approx}: \quad \Gamma \rightsquigarrow_P (\Gamma\sigma_1) \parallel \ldots \parallel (\Gamma\sigma_k) \qquad$ if $(t_1 \approx t_2) \in \Gamma$ and $\mathit{unify}_{AC}(t_1, t_2) = \{\sigma_1, \ldots, \sigma_k\}$

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

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

$\mathsf{S}_{@}: \quad \Gamma \rightsquigarrow_P (i : ri_1, f \approx g_1, \Gamma) \parallel \ldots \parallel (i : ri_k, f \approx g_k, \Gamma)$

if $(f@i) \in \Gamma$ and $\{(ri_1, g_1), \ldots, (ri_k, g_k)\} = \{(ru, g) \mid ru \in \lceil P \rceil^{DH} \cup \{\textsc{ISend}\} \wedge g \in \mathit{acts}(ru)\}$

Example graph constraint reduction rules:

$\mathsf{S}_{\mathbf{ULabel}}: \quad \Gamma \rightsquigarrow_P (ri \approx ri', \Gamma) \qquad$ if $\{i : ri, i : ri'\} \subseteq \Gamma$

$\mathsf{S}_{\mathbf{Acyc}}: \quad \Gamma \rightsquigarrow_P \bot \qquad$ if $i \lessdot_\Gamma i$

$\mathsf{S}_{\mathbf{UFresh}}: \quad \Gamma \rightsquigarrow_P (\Gamma, i \doteq j, u \doteq v\} \qquad$ if $\{i : ri, j : ru\} \subseteq \Gamma$ and $(\mathit{prems}(ri))_u = (\mathit{prems}(ru))_v = \mathsf{Fr}(m)$

$\mathsf{S}_{\downarrow}: \quad \Gamma \rightsquigarrow_P \bot \qquad$ if $(i : ri) \in \Gamma$ and ri not $\downarrow_{DH}$-normal

Example message deduction constraint reduction rules:

$\mathsf{S}_{\mathbf{Prem},\mathsf{K}^\uparrow}: \quad \Gamma \rightsquigarrow_P (\Gamma, j \triangleright \mathsf{K}^\uparrow(m), j \lessdot i)$

if $(i : ri) \in \Gamma$, $\mathit{prems}(ri)_u = \mathsf{K}^\uparrow(m)$ for some u, and j freshly chosen

$\mathsf{S}_{\triangleright,\mathsf{K}^\uparrow}: \quad \Gamma \rightsquigarrow_P (\Gamma, i : [\mathsf{K}^\uparrow(m_1), \ldots, \mathsf{K}^\uparrow(m_k)] \text{-[]}\!\!\rightarrow [\mathsf{K}^\uparrow(m)]) \parallel (\Gamma, i : [\mathsf{K}^\downarrow_y(m)] \text{-[]}\!\!\rightarrow [\mathsf{K}^\uparrow(m)])$

if $(i \triangleright \mathsf{K}^\uparrow(m)) \in \Gamma$, $m = f(m_1, \ldots, m_k)$, $f \notin \{*, \langle\_,\_\rangle, \_^{-1}\}$, and y freshly chosen

$\mathsf{S}_{\mathbf{Prem},\mathsf{K}^\downarrow_y}: \quad \Gamma \rightsquigarrow_P (\Gamma, j \dashrightarrow (i, u), j : \mathsf{Out}(z) \text{-[]}\!\!\rightarrow \mathsf{K}^\downarrow_{\mathsf{d}}(z))$

if $(i : ri) \in \Gamma$, $\mathit{prems}(ri)_u = \mathsf{K}^\downarrow_y(m)$ for some u and y, and j, z freshly chosen

$\mathsf{S}_{\dashrightarrow}: \quad \Gamma \rightsquigarrow_P \quad (\Gamma, i : ri, (i,1) \rightarrowtail (k, w))$

$\parallel (\Gamma, i : ri, (i,1) \rightarrowtail (j,1), j : ru_1, j \dashrightarrow (k, w)) \parallel \ldots \parallel (\Gamma, i : ri, (i,1) \rightarrowtail (j,1), j : ru_l, j \dashrightarrow (k, w))$

if $\{i : ri, i \dashrightarrow (k, w)\} \subseteq_{AC} \Gamma$, $(\mathit{concs}(ri))_1 \notin \{\mathsf{K}^\downarrow_{\mathsf{d}}(x) \mid x \in \mathcal{V}_{pub}\}$, $\{ru_1, \ldots, ru_l\} \in \mathit{ND}^{destr}$, and y freshly chosen

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.

Fig. 8. A subset of the rules defining TAMARIN's constraint-reduction relation $\rightsquigarrow_P$. The full set is given in [12] and is similar to the rules in [9].

Constraint solving rules for message deduction with respect to $\sharp$:

$\mathsf{S}_{\triangleright,\sharp}: \quad \Gamma \rightsquigarrow_P (i \triangleright \mathsf{K}^\uparrow(t), i \lessdot j, \Gamma)$

if $(j \triangleright \mathsf{K}^\uparrow(m)) \in \Gamma$, $\mathit{root}(m) = \sharp$, j freshly chosen, and $t \in \mathit{elems}(m) \smallsetminus \mathcal{V}_{msg}$.

$\mathsf{S}_{\dashrightarrow,\sharp}: \quad \Gamma \rightsquigarrow_P \parallel_{x \in [1,\ldots,l]} (i : ru, (i,1) \rightarrowtail (j,1), j : ru_x, j \twoheadrightarrow (k, w), \Gamma)$

if $(\mathit{concs}(ru))_1 = \mathsf{K}^\downarrow_{\mathsf{d}}(t)$ and $t = a \sharp b$ for some a and b, $\mathit{elems}(t) \cap \mathcal{V}_{msg} \subseteq \mathit{known}^{\lessdot}_\Gamma(i)$,

$\{ru_1, \ldots, ru_l\} = \left\{\mathsf{K}^\downarrow_{\mathsf{d}}(t) \text{-[]}\!\!\rightarrow \mathsf{K}^\downarrow_{\mathsf{d}}(m) \mid m \in \mathit{elems}(t) \smallsetminus \mathcal{V}_{msg}\right\}$, and j freshly chosen

Constraint solving rules for bilinear pairing that ensure **N9–N11**:

$\mathsf{S}_{\mathbf{N9}}: \quad \Gamma \rightsquigarrow_P \bot$

if $\{\, i : [\mathsf{K}^\downarrow_{\mathsf{d}}(a), \mathsf{K}^\uparrow(b)] \text{-[]}\!\!\rightarrow \mathsf{K}^\downarrow_{\mathsf{e}}([d]c) \,\} \subseteq_{AC} \Gamma$,

$\mathit{vars}(c) \subseteq \mathcal{V}_{pub}$, $\mathit{St}(c) \cap \mathit{FN} = \varnothing$, and $\mathit{nifactors}(d) \subseteq \mathit{nifactors}(b)$

$\mathsf{S}_{\mathbf{N10}}: \quad \Gamma \rightsquigarrow_P \bot$

if $\{\, (j,1) \rightarrowtail (i,1),\ i : [\mathsf{K}^\downarrow_{\mathsf{d}}([t_1]p), \mathsf{K}^\downarrow_{\mathsf{d}}([t_2]q)] \text{-[]}\!\!\rightarrow \mathsf{K}^\downarrow_{\mathsf{d}}(\hat{e}(p,q)\,\hat{}\,c)$,

$j : [\mathsf{K}^\downarrow_{\mathsf{d}}([t_1]p), \mathsf{K}^\downarrow_{\mathsf{d}}([t_2]q)] \text{-[]}\!\!\rightarrow \mathsf{K}^\downarrow_{\mathsf{d}}(\hat{e}(p,q)\,\hat{}\,c) \,\} \subseteq_{AC} \Gamma$,

$\mathit{vars}(p,q) \subseteq \mathcal{V}_{pub}$, $\mathit{St}(p,q) \cap \mathit{FN} = \varnothing$, and $\mathit{nifactors}(t_i) \subseteq \mathit{nifactors}(d)$ for $i = 1$ or $i = 2$

$\mathsf{S}_{\mathbf{N11}}: \quad \Gamma \rightsquigarrow_P \bot$

if $\{\, k : [\mathsf{K}^\downarrow_{\mathsf{d}}([a]p), \mathsf{K}^\downarrow_{\mathsf{d}}([b]q)] \text{-[]}\!\!\rightarrow \mathsf{K}^\downarrow_{\mathsf{d}}(\hat{e}(p,q)\,\hat{}\,c),\ k_1 : ri_1,\ k_2 : ri_2$,

$i_1 : ru_1,\ (i_1, u_1) \rightarrowtail (k_1, 1),\ k_1 \twoheadrightarrow j, 1)$,

$i_2 : ru_2,\ (i_2, u_2) \rightarrowtail (k_1, 1),\ k_2 \twoheadrightarrow j, 2) \,\} \subseteq_{AC} \Gamma$,

ri_1 and ri_2 are instances of $\textsc{IRecv}$ and $\mathit{fsyms}(ru_2) >_{fs} \mathit{fsyms}(ru_1)$

Fig. 9. New constraint reduction rules for AC operators and bilinear pairing, explained in Section IV-D.

contradictory constraints. In the following, we first give the definition of $\rightsquigarrow_P$ and then present the algorithm.

A subset of the rules defining the *constraint-reduction relation* $\rightsquigarrow_P$ is given in Figure 8. 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. For, a constraint system Γ, its *actions* $as(\Gamma)$ are defined as

$$as(\Gamma) = \{f@i \mid \exists r\, a.\ (i : l\text{-}[\ a\ ]\!\rightarrow r) \in \Gamma \wedge f \in a\}$$

and the *temporal order of* Γ is

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

TAMARIN's constraint-solving algorithm is shown in Figure 7. 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, $solved(\mathbf{\Omega})$ is used to denote the set of solved constraint systems in $\mathbf{\Omega}$. The automated analysis uses a heuristic 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 correctness of TAMARIN's algorithm follows from two theorems from [9], paraphrased below:

Theorem 1. *([9]) 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 $1 \leq i \leq n$.*

Theorem 2. *([9]) A P-solution can be constructed from every solved system in the state $\mathbf{\Omega}$ of the constraint-solving algorithm.*

E. Lemmas, axioms, and induction

Note that lemmas and axioms can be handled by adding guarded trace formulas to the initial constraint system. TAMARIN's support for induction exploits this capability. To prove a property φ by induction, TAMARIN first checks if the empty trace satisfies φ. Afterwards, TAMARIN performs a (complete) search for counterexamples with an additional axiom φ' that encodes that $init(tr)$ satisfies φ, where $init(tr) = tr_1, \ldots, tr_{|tr|-1}$. To define φ', a new trace atom is added that allows TAMARIN to translate φ into φ' such that $tr \vDash_{AC} \varphi'$ iff $init(tr) \vDash_{AC} \varphi$.

IV. EXTENDING TAMARIN WITH AC OPERATORS AND BILINEAR PAIRINGS

In this section we present our extensions to TAMARIN's theory and algorithm to support AC operators and bilinear pairings, as described in Section II-A. We will present case studies later in Section V.

Conceptually, extending TAMARIN's theory with new operators requires the following steps. First, the signature must be extended and an appropriate equational theory must be chosen (e.g., possessing the finite variant property). In most cases, it suffices to add a message deduction rule for each new operator. Then, the variants of the new message deduction rules must be annotated appropriately with tags to obtain the normal message deduction rules. Once this is done, one can attempt to prove properties using the algorithm. For most operators this will fail because the new equations and rules typically lead to state space explosion or non-termination. For example, bilinear pairing is a commutative operator, and adding its equations and rules leads to many different ways of deriving the same message, causing the analysis to fail. This can be countered by specifying appropriate normal forms that restrict the ways that particular messages can be derived. Finally, constraint solving rules are added to the constraint solving procedure. These rules typically enforce normal form conditions or construct derivation paths for the new operators.

In the following, we go through the above steps for AC operators and bilinear pairings. The main design choices are choosing the normal forms and constraint solving rules. While some of these choices are canonical, the overall design space is large, and is ultimately justified by the effectiveness and efficiency of the algorithm on case studies. In general, our aim is to reduce the kinds of redundancy that arise in derivations, while avoiding too many case distinctions.

A. Modeling the new operators

To support the new operators presented in Section II-A, we extend Σ_{DH} and E_{DH} as follows. We define:

$$\Sigma_{BP} = \Sigma_{DH} \cup \{\ \hat{e}(\_,\_), [\_]\_, \_ \sharp \_\ \}$$

and

$$E_{BP} = E_{DH} \cup \{[z]([y]x) = [z * y]x,\ [1]x = x,\ \hat{e}(x,y) = \hat{e}(y,x),\ \hat{e}([z]x,y) = \hat{e}(x,y)\,\hat{}\,z,\ x \sharp (y \sharp z) = (x \sharp y) \sharp z,\ x \sharp y = y \sharp x\ \}.$$

We use a fixed public name P and terms $[s]\mathsf{P}$ to model elements of the group $\mathbb{G}$. The bilinear map sends two terms $[s]\mathsf{P}$ and $[t]\mathsf{P}$ to $\hat{e}([s]\mathsf{P},[t]\mathsf{P}) =_{E_{BP}} \hat{e}(\mathsf{P},\mathsf{P})\,\hat{}\,(s * t)$. The elements of the group $\mathbb{G}_T$ are therefore modeled as terms $\hat{e}(\mathsf{P},\mathsf{P})\,\hat{}\,u$. We use the $\sharp$ operator to model non-empty multisets. For example, $\mathsf{A} \sharp \mathsf{B} \sharp \mathsf{C}$ models the multiset consisting of A, B, and C.

The extended signature Σ_{BP} yields additional message deduction rules for constructing multisets and performing scalar multiplication and bilinear pairings. To allow the adversary to extract elements from multisets, we extend the message deduction rules MD with the rule $\mathsf{K}(x \sharp y)\text{-}[]\!\rightarrow\mathsf{K}(x)$.

We denote the outermost function symbol of a term t by $root(t)$. We relax the definition of guarded trace property to allow for subterms t with $root(t) = \sharp$ in addition to variables, public names and irreducible function symbols from $\Sigma_{\mathcal{ST}}$. This enables the use of $\sharp$ in security properties. We explain the usefulness and usage of $\sharp$ in the context of group protocols, in Section V-A.

B. Example: Joux protocol

We formalize the Joux protocol, described in Section II-B, in Figure 10. We explain our formalization from the point of view of a participant A_1 creating a group with participants A_2 and A_3. In the *first step* rule, A_1 chooses his ephemeral secret key x_1 as well as two peers A_2 and A_3 and sends $[x_1]\mathsf{P}$ on the authentic channel $!\mathsf{AO}(A_1, \ldots)$. The protocol state fact $\mathsf{St}(A_1, A_2 \sharp A_3, x_1)$ denotes that A_1 now expects responses from A_2 and A_3 and will compute the shared key using x_1. In the *second step* rule, A_1 waits for the messages from the two peers sent on their authentic channels which contain their ephemeral public keys, and upon receiving both messages computes the session key as $\hat{e}(X_2, X_3) \hat{} x_1$. The $\mathsf{SessionKey}$-fact denotes that A_1 accepts the given key with the given partners.

The security property is given in Figure 11. It states that whenever k is a session key then the adversary can not know k.

First step:

$\mathsf{Fr}(x_1) \text{-}[\,]\!\!\rightarrow \mathsf{St}(A_1, A_2 \sharp A_3, x_1), !\mathsf{AO}(A_1, [x_1]\mathsf{P})$

Second step:

$\mathsf{St}(A_1, A_2 \sharp A_3, x_1), !\mathsf{AO}(A_2, X_2), !\mathsf{AO}(A_3, X_2)$
$\text{-}[\ \mathsf{SessionKey}(A_1, A_2 \sharp A_3, \hat{e}(X_2, X_3) \hat{} x_1)\ ]\!\!\rightarrow [\,]$

Overhear message:

$!\mathsf{AO}(A, X) \text{-}[\,]\!\!\rightarrow \mathsf{Out}(X)$

Fig. 10. Multiset rewriting rules formalizing Joux where $A_1, A_2, A_3 \in \mathcal{V}_{pub}$.

$\neg(\exists A_1\, A_2\, A_3\, i_1\, i_2\, k.$
// adversary knows the session key k for A_1, A_2, and A_3
$(\mathsf{SessionKey}(A_1, A_2 \sharp A_3, k)@i_1 \wedge \mathsf{K}(k)@i_2))$

Fig. 11. Joux security property.

C. Verification theory

We extend TAMARIN's verification theory to enable reasoning with respect to the new operators. First, we adapt dependency graphs modulo AC to account for the new equations in E_{BP}. Then, we extend the set of normal message deduction rules to account for the new operators. Finally, to enable verification with respect to this new theory, we introduce new normal-form conditions for the new deduction rules. These will be exploited by the new constraint reduction rules that we present in Section IV-D.

Dependency graphs modulo ACC*:* We define the set of equations ACC as

$$ACC = AC \cup \{x \sharp (y \sharp z) = (x \sharp y) \sharp z,\ x \sharp y = y \sharp x,\\ \hat{e}(x, y) = \hat{e}(y, x)\}$$

and the rewriting system BP as

$$BP = DH \cup \{[z]([y]z) \rightarrow [z * y]x,\ [1]x \rightarrow x,\\ \hat{e}([y]x, z) \rightarrow \hat{e}(x, z) \hat{} y\}.$$

BP,ACC is a finite variant decomposition [12,16] of E_{BP} for the following reasons. First, $(BP \cup ACC)$ is an equational representation of E_{BP}. Second, ACC is regular, sort-preserving, and all variables are of sort *msg*. Third, BP is sort-decreasing and BP,ACC-rewriting is convergent and coherent. We have used the AProVE termination tool [17] and the Maude Church-Rosser and Coherence Checker [18,19] to verify both properties. Fourth, there is a complete and finitary ACC-unification algorithm. Finally, the finite variant property can be established for BP, ACC as follows. Comon-Lundh and Delaune [20] prove that DH,AC has the finite variant property. Since the new rules defined in BP work on operators not defined before, and all new operators are bounded, we can conclude the boundedness of BP. This gives us the finite variant property according to [16].

Normal Message deduction: To define the new message deduction rules, we extend the meaning of $\mathsf{K}^{\downarrow}$-facts as follows. $\mathsf{K}^{\downarrow}_{\mathsf{d}}(m)$ means that m is an extracted subterm or the result of applying $\hat{e}$ to an extracted subterm. We extend the meaning of $\mathsf{K}^{\downarrow}_{\mathsf{e}}(m)$ to include that m can be the result of changing the scalar in an extracted scalar multiplication.

The normal message deduction rules ND_{BP} for bilinear pairing and $\sharp$ are given in Figure 12, and extend the set *ND*. Scalar multiplication is treated similar to exponentiation, i.e., there is a construction rule, there are deconstruction rules, and scalar multiplication rules that use the fact symbol $\mathsf{K}^{\downarrow}_{\mathsf{e}}$ in the conclusion. For bilinear pairing, there is a construction rule and there are bilinear pairing rules corresponding to the non-trivial variants of $\hat{e}(x, y)$. They cover all the different possible ways to normalize the product of the scalars from the two scalar multiplications given as arguments to $\hat{e}$. The message in the conclusion of a bilinear pairing rule is always an exponentiation and can therefore only be used by COERCE, an exponentiation rule, or a deconstruction rule for exponentiation. Note that for the first bilinear pairing rule, the first premise is a scalar multiplication and uses $\mathsf{K}^{\downarrow}_{\mathsf{d}}$ and the second premises uses $\mathsf{K}^{\uparrow}$ and cannot be a scalar multiplication if the instance is in normal form. For all remaining bilinear pairing rules, both premises are scalar multiplications and use $\mathsf{K}^{\downarrow}_{\mathsf{d}}$-facts.

Normal Dependency Graphs: TAMARIN's search systematically explores all dependency graphs that represent a set of traces, see Section III-B. By introducing dependency graph normal forms, we reduce the number of dependency graphs that need to be explored, effectively excluding dependency graphs whose traces are subsumed by other (normal) dependency graphs.

To enable the verification of our extension of TAMARIN, we introduce five new normal-form conditions, depicted in Figure 6 on page 7. To state the conditions, we introduce auxiliary definitions. A node i labeled with an instance of a protocol rule is the *send-node of the premise* (j, u) *in dg* if there is a node k labeled with an instance of IRECV such that there is an edge $(i, v) \rightarrowtail (k, 1)$ for some k, v and a chain

Construction rules:

$$\frac{\mathsf{K}^{\uparrow}(x) \qquad \mathsf{K}^{\uparrow}(y)}{\mathsf{K}^{\uparrow}([y]x)} \quad \frac{\mathsf{K}^{\uparrow}(x) \qquad \mathsf{K}^{\uparrow}(y)}{\mathsf{K}^{\uparrow}(\hat{e}(x,y))} \quad \frac{\mathsf{K}^{\uparrow}(x_1) \qquad \dots \qquad \mathsf{K}^{\uparrow}(x_k)}{\mathsf{K}^{\uparrow}(x_1 \sharp \dots \sharp x_k)}$$

Deconstruction rules:

$$\frac{\mathsf{K}^{\downarrow}_{\mathsf{d}}([y]x) \qquad \mathsf{K}^{\downarrow}_{\mathsf{d}}(y^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x)} \quad \frac{\mathsf{K}^{\downarrow}_{\mathsf{d}}([y^{-1}]x) \qquad \mathsf{K}^{\uparrow}(y)}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x)} \quad \frac{\mathsf{K}^{\downarrow}_{\mathsf{d}}([y * z^{-1}]x) \qquad \mathsf{K}^{\uparrow}(y^{-1} * z)}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x)} \quad \frac{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x \sharp y)}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(x)}$$

Scalar multiplication rules:

$$\frac{\mathsf{K}^{\downarrow}_{\mathsf{d}}([y]x) \qquad \mathsf{K}^{\uparrow}(z)}{\mathsf{K}^{\downarrow}_{\mathsf{e}}([y * z]x)} \quad \frac{\mathsf{K}^{\downarrow}_{\mathsf{e}}([y]x) \qquad \mathsf{K}^{\uparrow}(y^{-1} * z)}{\mathsf{K}^{\downarrow}_{\mathsf{e}}([z]x)} \quad \dots \quad \frac{\mathsf{K}^{\downarrow}_{\mathsf{d}}([y_1 * y_2^{-1}]x) \qquad \mathsf{K}^{\uparrow}(z_1 * z_2^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{e}}([y_1 * z_1 * (y_2 * z_2)^{-1}]x)}$$

Bilinear pairing rules:

$$\frac{\mathsf{K}^{\downarrow}_{\mathsf{d}}([z]x) \qquad \mathsf{K}^{\uparrow}(y)}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(\hat{e}(x,y) \,\hat{}\, z)} \quad \frac{\mathsf{K}^{\downarrow}_{\mathsf{d}}([z_1]x) \qquad \mathsf{K}^{\downarrow}_{\mathsf{d}}([z_2]y)}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(\hat{e}(x,y) \,\hat{}\, z_1 * z_2)} \quad \dots \quad \frac{\mathsf{K}^{\downarrow}_{\mathsf{d}}([y_1 * y_2^{-1}]x_1) \qquad \mathsf{K}^{\downarrow}_{\mathsf{d}}([z_1 * z_2^{-1}]x_2)}{\mathsf{K}^{\downarrow}_{\mathsf{d}}(\hat{e}(x_1,x_2) \,\hat{}\, (y_1 * z_1 * (y_2 * z_2)^{-1}))}$$

Fig. 12. The normal message deduction rules ND_{BP} for bilinear pairing. There are construction rules for $\sharp$ for all $k > 1$. There are 42 scalar multiplication rules and 28 bilinear pairing rules computed from the BP,ACC-variants of the corresponding rules.

$k \twoheadrightarrow_{dg} (j, u)$. Intuitively, the send-node of a premise $\mathsf{K}^{\downarrow}_{y}(m)$ is the protocol rule that sends the message from which m is extracted. We denote the sequence of fact symbols occurring in a multiset rewriting rule ru with $fsyms(ru)$. We also assume given a total order $<_{fs}$ on sequences of fact symbols.

The condition **N7** is similar to condition **N2** for multiplication, but deals with $\sharp$ instead. The condition **N8** ensures that the deconstruction rule for $\sharp$ never extracts a multiset. Together with **N7**, this enforces that multisets are completely deconstructed and then constructed from scratch. The condition **N9** directly corresponds to condition **N6** for exponentiation and forbids unnecessary uses of scalar multiplication rules.

The condition **N10** prevents deductions where an exponentiation rule is applied to the result of a bilinear pairing rule such that the deduction can be replaced by a simpler one.

The condition **N11** prevents redundant cases resulting from the commutativity of $\hat{e}$, where two dependency graphs only differ in the order of premises of a bilinear pairing rule. This is especially problematic for the second bilinear pairing rule in Figure 12 which is symmetric and occurs very often. We therefore enforce that the send-node of the second premise cannot be smaller than the send-node of the first premise. Since we want to evaluate this condition on *symbolic* constraint systems, we choose a partial order on rule instances that considers only the fact symbols.

D. Constraint Solving

We now present five new constraint solving rules required for $\sharp$ and bilinear pairing. We first introduce two additional definitions. The *messages known before i in* Γ are defined as

$$known^{\prec}_{\Gamma}(i) = \{m \mid \exists j. j \prec_{\Gamma} i \wedge j \triangleright \mathsf{K}^{\uparrow}(m)\}.$$

The *set of elements of a term t* is defined as

$$elems(t) = \begin{cases} elems(a) \cup elems(b) & \text{if } t = a \sharp b \\ \{t\} & \text{otherwise.} \end{cases}$$

The new constraint solving rules for $\sharp$ are shown in Figure 9 on page 8. The rule $\mathcal{S}_{\triangleright,\sharp}$ directly introduces constraints for the premises of the construction rule for $\sharp$ instead of introducing a node constraint for the rule itself. The rule $\mathcal{S}_{\twoheadrightarrow,\sharp}$ solves chains that start at conclusions $\mathsf{K}^{\downarrow}_{\mathsf{d}}(a \sharp b)$. To ensure that the original rule $\mathcal{S}_{\twoheadrightarrow}$ is never used for such conclusions, we redefine $\mathcal{S}_{\twoheadrightarrow}$ and add the additional side condition that $(concs(ri))_1 \neq \mathsf{K}^{\downarrow}_{\mathsf{d}}(s \sharp t)$ for all terms s and t. The new rule $\mathcal{S}_{\twoheadrightarrow,\sharp}$ handles this case by adding one case for every element of $a \sharp b$ that is not a message variable. The rule is only applicable if all elements of $a \sharp b$ that are message variables are known before. The rule's correctness depends on three normal form conditions. Condition **N5** allows us to ignore all cases where a message that is already $\mathsf{K}^{\uparrow}$-known is extracted. Condition **N7** allows us to ignore the COERCE case since $\mathsf{K}^{\uparrow}(a \sharp b)$ is never the conclusion of COERCE. Finally, condition **N8** allows us to ignore all cases where a term of the form $a \sharp b$ is extracted.

The constraint solving rules that ensure the new normal form conditions are also shown in Figure 9: the conditions of each of these rules specify the negation of the corresponding normal form, in which case the constraint system has no solution ($\perp$).

The resulting constraint solving relation is sound and complete, and we can still obtain P-solutions from solved constraint systems. The full proof is given in [12].

E. Example: Joux protocol verification

We now explain the verification of the unsigned Joux protocol from Section IV-B. For a full description of the verification of a signed version of Joux, with perfect forward secrecy, we refer to [12].

The key computation in the *second step* rule applies $\hat{e}$ to two message variables and raises the result to the power of the ephemeral secret key. The main challenge in the automated proof is to compute an exhaustive case distinction that lists all possible ways the adversary can deduce a term of the form $u \,\hat{}\, v$.

Using the extended constraint solving algorithm, TAMARIN computes the following five cases:

(i) Construct an exponentiation from the known base u, which is not an exponentiation, and the non-inverse factors of the exponent v.
(ii) Apply the bilinear map $\hat{e}$ to a scalar multiplication extracted from a message of the *first step* rule, and an arbitrary other message that is not a scalar multiplication. Note that if the other message is a scalar multiplication then the result is not normal.
(iii) Extract the scalar multiplications from two protocol sends and apply the bilinear map to both.
(iv) Perform the same steps as in (ii) and use an exponentiation rule to multiply the exponent of this deduction with a factor.
(v) Perform the same steps as in (iii) and use an exponentiation rule to multiply the exponent of this deduction with a factor.

The automated analysis proceeds along the following lines: Due to the form of the messages received, the key must be of the form $\hat{e}(\mathsf{P},\mathsf{P})\,\hat{}\,(x_1 * x_2 * x_3)$, where x_1, x_2 and x_3 are fresh names that are used as ephemeral secret keys by the participants. Then the computed case distinction is used to derive all possible sources of this exponentiation. All these cases are shown to be contradictions because they require the adversary to know one of the ephemeral secret keys, but the adversary does not know (and cannot deduce) any of the ephemeral secret keys.

V. CASE STUDIES

In this section we report on case studies to validate the effectiveness and efficiency of our TAMARIN extension. The extension has been integrated into the latest TAMARIN version [13]. We give an overview of our case studies in Table I on page 15. In sections V-A through V-D we present details of the analysis of three different group key agreement protocols, STR, group Joux, and GDH. Afterwards, in Section V-E, we report on further case studies that include tripartite and identity-based protocols, and summarize our experimental results.

A. Modeling group protocols

The three group protocols use different structures. For example, STR and group Joux have subgroup keys, but GDH does not. We explain the different challenges arising for these protocols and how we successively tackled them.

In our experience, the protocols that have subgroup keys are easier to analyze, as fewer lemmas are needed since the intermediate keys are directly available. But the verification does not depend on having subgroup keys and we can verify GDH with TAMARIN after establishing some lemmas stating properties of the partial keys. Note that in all three protocols, the number of participants is indeed unbounded and we do not introduce any such limitations in our models.

We now show how the new operators from Section IV-A enable us to model group protocols. We use a unary representation for non-zero natural numbers which uses the constant 1 and the AC-operator +. Based on this representation, we can axiomatize "x less than y" as $\exists z.x + z = y$. In our models, we abbreviate this formula with $x <_{nat} y$ and use standard notation for numbers.

We use the constant *empty* to represent the empty finite map and the pair $\langle k, v\rangle$ to represent the mapping from the key k to the value v. The finite map that maps k_i to v_i for $i = 1$ to n is then represented as $\mathit{empty} \sharp \langle k_1, v_1\rangle \sharp \ldots \sharp \langle k_n, v_n\rangle$. We use matching (modulo AC) to perform map lookups. That is, to look up the value x for the key k in the map m, we match m with $\langle k, x\rangle \sharp m'$ binding the remainder of the map to m'.

Additionally, all of our protocol formalizations contain several axioms (not shown explicitly) defining the meaning of certain facts: $\mathsf{Eq}(x,y)$ formalizes $x \doteq y$, $\mathsf{InEq}(x,y)$ formalizes $x \not\doteq y$, $\mathsf{Less}(x,y)$ formalizes $x <_{nat} y$, and $\mathsf{Uniq}(x)$ formalizes that for each x, there is at most one $\mathsf{Uniq}(x)$ in the trace. We use an authentic channel !AO for communication, which the adversary can read, using the last rule shown in Figure 10.

We model the adversary's interaction with the protocol by queries. For each query **Q** with arguments $a_1, \ldots, a_k$, we define a rule with rule name $\mathbf{Q}(a_1, \ldots, a_k)$. In the corresponding rewrite rule, we add the premise $\mathsf{In}(\langle \mathbf{Q}, a_1, \ldots, a_k\rangle)$ and the action $\mathsf{Uniq}(\langle \mathbf{Q}, a_1, \ldots, a_k\rangle)$.

B. STR

Recall STR as explained in Section II-B, in particular Figure 1. We describe our formalization of the group creation and the group leader A_1 in detail, given in three rules in Figure 13. We only briefly describe the three rules used for participants A_2–A_k. We introduce a function *te* to model the conversion of group elements to exponents, which was denoted by underlining in Section II-B. Because it might be possible to invert the conversion function *te*, we also introduce its inverse function *ite* and the equation $\mathit{ite}(\mathit{te}(x)) = x$.

The **Create-Group** rule stores the three parameters given in the query in a persistent fact !Group. The Uniq action ensures that each group identifier can only be used once. The size of the group is given by the number k and $pMap$ is the mapping from numbers $1, \ldots, k$ to public names.

The **Start-Leader** rule takes an existing !Group fact and reads the group identifier gid. In the map from indices to names, $pMap$, it finds the public name A of the leader with index 1 and finds the public name B for the second participant with index 2. It generates a new ephemeral secret key *esk* for A. On B's authentic channel, it receives a message stating that the content Y is an ephemeral public key (shown by EpkOf) for group gid from participant 2. With this, it can log the accepting fact Accept for that gid, name A, index 1, noting the subgroup key is for the first two participants, the total group size is k, and the key is $Y\,\hat{}\,\mathit{esk}$. Then it sends on A's authentic channel a message that contains the subgroup's ephemeral public key (shown by GpkFor) for group gid for consumption by participant 2 and the key is $\mathsf{g}\,\hat{}\,\mathit{esk}$. It then stores in a Leader fact the group identifier gid, the name of the leader A, the index of the next participant to communicate with, 3, and the current subgroup key $Y\,\hat{}\,\mathit{esk}$. Note that, with

Create-Group $(gid, k, pMap)$:

$[] \dashv[\ \mathsf{Uniq}(gid)\ ]\!\!\rightarrow\ !\mathsf{Group}(gid, k, pMap)$

Start-Leader (gid):

$!\mathsf{Group}(gid, k, \langle 1, A\rangle \sharp \langle 2, B\rangle \sharp pM'), \mathsf{Fr}(esk),$
$!\mathsf{AO}(B, \langle \mathsf{EpkOf}, gid, 2, Y\rangle)$
$-[\ \mathsf{Accept}(gid, A, 1, 2, k, Y\,\hat{}\,esk)\ ]\!\!\rightarrow$
$!\mathsf{AO}(A, \langle \mathsf{GpkFor}, gid, 2, \mathsf{g}\,\hat{}\,esk\rangle),$
$\mathsf{Leader}(gid, A, 3, Y\,\hat{}\,esk)$

Step-Leader (gid, j):

$\mathsf{Leader}(gid, A, j, key), !\mathsf{Group}(gid, k, \langle j, B\rangle \sharp pMap),$
$!\mathsf{AO}(B, \langle \mathsf{EpkOf}, gid, j, Y\rangle$
$-[\ \mathsf{Accept}(gid, A, 1, j, k, Y\,\hat{}\,te(key)), \mathsf{Less}(j, k+1)\ ]\!\!\rightarrow$
$!\mathsf{AO}(A, \langle \mathsf{GpkFor}, gid, j, \mathsf{g}\,\hat{}\,te(key)\rangle),$
$\mathsf{Leader}(gid, A, j+1, Y\,\hat{}\,te(key))$

Fig. 13. Multiset rewriting rules for leader in STR. EpkOf, GpkFor $\in$ *pub*.

$$\neg(\exists i_1\ i_2\ gid\ A\ i\ j\ k\ key.\ (\mathsf{Accept}(gid, A, i, j, k, key)@i_1 \wedge \mathsf{K}(key)@i_2))$$

Fig. 14. STR secrecy lemma: adversary does not know any accepted keys.

the message sent here, participant B can compute the same subgroup key, as it also has its own ephemeral secret key used to generate the ephemeral public key that was received by this rule.

The **Step-Leader** rule is queried for a given group identifier gid and index j. Note that the adversary can only query these in sequential order as the Leader fact is not persistent and the index is incremented each time it is called. From the Leader fact with matching gid and j, it retrieves the leader name A and the subgroup key key. From the !Group fact it retrieves the public name B associated with index j. It receives on B's authentic channel the message that Y is the ephemeral public key of B for this group. It logs the action Accept with the group identifier, name A, index 1, index j to show the key is for the subgroup A_1–A_j, the group size k and the actual new subgroup key $Y\,\hat{}\,te(key)$. It also logs a Less fact to ensure that j is in bounds, i.e., smaller than $k+1$. On A's authentic channel it sends the subgroup's ephemeral public key for this group gid, designated for participant B with index j, which is $\mathsf{g}\,\hat{}\,te(key)$. The Leader fact is stored again, with incremented index $j+1$ and the new subgroup key $Y\,\hat{}\,te(key)$.

The other participants A_2–A_k use three rules. In their first rule they pick their ephemeral secret key and send on their authentic channel the associated ephemeral public key for the group. In their second rule they receive their subgroup's ephemeral public key from the leader and compute the subgroup key. Their last rule is just like the **Step-Leader** rule, but without sending anything. That is, it receives the ephemeral public key of the participants with higher index in order and computes successively the next subgroup key until done.

The secrecy claim is given in the form of the lemma shown in Figure 14. TAMARIN automatically proves this lemma using induction as described in Section III-E. Without induction, TAMARIN's backwards search would not terminate, because it keeps repeatedly unfolding 'one more step' of the loops when trying to prove the secrecy of STR for an arbitrary number of participants.

When using induction, TAMARIN first checks the empty trace which trivially satisfies the secrecy lemma as no Accept fact was logged yet. For the induction step, TAMARIN's backward search first shows that that the adversary cannot compute the new key without knowing either the previous subgroup key key or some ephemeral secret key esk. It then uses the induction hypothesis to show that the secrecy lemma holds for the subgroup key key and the first case is therefore impossible. Finally, it shows that the second case is also impossible because the adversary cannot deduce esk.

For STR, we analyze two different versions: one using explicitly authentic channels, which we presented, and another version using insecure channels and signatures. See Table I in Section V-E for the different verification run times.

C. From Joux to group Joux

For the analysis of group Joux, we refer to Section IV-B for the Joux protocol and to Section V-B for the use of induction. We have shown in Figure 1 how group Joux extends the Joux protocol. Note that the extension described in Section IV-A is needed for both the bilinear pairing, and the representation of the tree by maps. The TAMARIN tool verifies the secrecy of the group key automatically, when induction is enabled. See the table in Section V-E for more details.

D. GDH

Figure 15 depicts the multiset rewriting rules for GDH, as presented in Section II-B. The **Create-Group** rule stores the two parameters, the group identifier gid and the number of participants k, in the !Group fact. It also logs an action fact to ensure that each group identifier is only used once and checks that the group size is not one.

The **Start-Participant** rule with a given gid, participant index i, and participant name A, reads the group size k from the !Group fact and generates a new ephemeral secret key esk. It logs a number of actions: two uniqueness constraints, so that each index i and name A are only used once per gid, as well as a less-than fact Less ensuring that $i \in \{1, \ldots, k\}$. It also logs in the Esk fact that esk is the ephemeral secret key of i. It stores the participant state fact !Pstate with the group identifier, participant index and name, and the participant's secret. Separately, a mapping !Pmap, parametric on the group identifier, from participant index to name is stored.

In the rule **Send-First** for a group identifier gid, the state must contain the participant state fact for that gid with index 1, name A, and ephemeral secret key esk. It sends two messages on A's authentic channel for this group's participant 2, one with g as the key for 1, and the other with $\mathsf{g}\,\hat{}\,esk$ as the round key for 2. In WaitAccept, it stores its information so it can accept using the last rule.

Create-Group (gid, k):

$[\,]\text{-}[\ \mathsf{Uniq}(gid), \mathsf{InEq}(k,1)\ ]\!\!\rightarrow\ !\mathsf{Group}(gid,k)$

Start-Participant (gid, i, A):

$!\mathsf{Group}(gid,k), \mathsf{Fr}(esk)$
$\text{-}[\ \mathsf{Uniq}(\langle gid, i\rangle), \mathsf{Uniq}(\langle gid, A\rangle), \mathsf{Less}(i, k+1), \mathsf{Esk}(i, esk)\ ]\!\!\rightarrow$
$!\mathsf{Pstate}(gid, i, A, esk), !\mathsf{Pmap}(gid, i, A)$

Send-First (gid):

$!\mathsf{Pstate}(gid, 1, A, esk)\text{-}[\,]\!\!\rightarrow$
$!\mathsf{AO}(A, \langle gid, 2, \mathsf{KeyFor}, 1, \mathsf{g}\rangle),$
$!\mathsf{AO}(A, \langle gid, 2, \mathsf{RoundKey}, \mathsf{g}\,\hat{}\,esk\rangle),$
$\mathsf{WaitAccept}(gid, 1, A, esk)$

Recv-Others (gid, i, j):

$!\mathsf{Pstate}(gid, i, A, esk), !\mathsf{Pmap}(gid, l, B),$
$!\mathsf{AO}(B, \langle gid, i, \mathsf{KeyFor}, j, \mathsf{g}\,\hat{}\,y\rangle)$
$\text{-}[\ \mathsf{Exp}(i, y), \mathsf{Eq}(l+1, i), \mathsf{Less}(j, i)\ ]\!\!\rightarrow$
$!\mathsf{AO}(A, \langle gid, i+1, \mathsf{KeyFor}, j, (\mathsf{g}\,\hat{}\,y)\,\hat{}\,esk\rangle)$

Recv-Roundkey (gid, i):

$!\mathsf{Pstate}(gid, i, A, esk), !\mathsf{Pmap}(gid, l, B), !\mathsf{Group}(gid, k),$
$!\mathsf{AO}(B, \langle gid, i, \mathsf{RoundKey}, \mathsf{g}\,\hat{}\,y\rangle)$
$\text{-}[\ \mathsf{Exp}(i, y), \mathsf{Eq}(l+1, i), \mathsf{InEq}(i, k)\ ]\!\!\rightarrow$
$!\mathsf{AO}(A, \langle gid, i+1, \mathsf{KeyFor}, i, \mathsf{g}\,\hat{}\,y\rangle),$
$!\mathsf{AO}(A, \langle gid, i+1, \mathsf{RoundKey}, (\mathsf{g}\,\hat{}\,y)\,\hat{}\,esk\rangle),$
$\mathsf{WaitAccept}(gid, i, A, esk)$

Recv-Roundkey-Last (gid):

$!\mathsf{Pstate}(gid, k, A, esk), !\mathsf{Pmap}(gid, l, B), !\mathsf{Group}(gid, k),$
$!\mathsf{AO}(B, \langle gid, k, \mathsf{RoundKey}, \mathsf{g}\,\hat{}\,y\rangle)$
$\text{-}[\ \mathsf{Exp}(k, y), \mathsf{Eq}(l+1, k), \mathsf{Accept}(gid, A, k, (\mathsf{g}\,\hat{}\,y)\,\hat{}\,esk)\ ]\!\!\rightarrow\ [\,]$

Accept (gid, i):

$\mathsf{WaitAccept}(gid, i, A, esk), !\mathsf{Group}(gid, k),$
$!\mathsf{Pmap}(gid, k, B), !\mathsf{AO}(B, \langle gid, k+1, \mathsf{KeyFor}, i, \mathsf{g}\,\hat{}\,y\rangle)$
$\text{-}[\ \mathsf{Accept}(gid, A, i, (\mathsf{g}\,\hat{}\,y)\,\hat{}\,esk)\ ]\!\!\rightarrow\ [\,]$

Fig. 15. Multiset rewriting rules formalizing GDH. $\mathsf{KeyFor}, \mathsf{RoundKey} \in pub$.

$$\neg(\exists j_1\ j_2\ gid\ i\ A\ key.(\mathsf{Accept}(gid, A, i, key)@j_1 \wedge \mathsf{K}(key)@j_2))$$

Fig. 16. GDH security property: adversary does not know accepted keys.

$\forall i_1\ i_2\ r_1\ r_2\ esk\ x.$
// if esk is an ephemeral secret key that appears in an exponent,
$(\mathsf{Esk}(r_1, esk)@i_1 \wedge \mathsf{Exp}(r_2, x * esk)@i_2$
// then it was created in an earlier round
$\Rightarrow r_1 <_{nat} r_2)$

Fig. 17. GDH Lemma 1.

The rule **Recv-Others** gets the acting participant A's index i and the index j whose assigned partial key is modified. The state contains facts ensuring that the received message is from the predecessor B, with index l, of i, and the key is for j. On its authentic channel A sends the received key, exponentiated with its ephemeral secret key *esk*, for further handling by the participant with index $i+1$. It also logs that participant i has responded to this message in the Exp fact. Note that receiving $\mathsf{g}\,\hat{}\,y$ (as opposed to a message variable Y) models that the recipient performs a group element check. Note too that the exponent is not directly used by the protocol, and only appears in the action.

The **Recv-Roundkey** rule for participant i receives from A's predecessor B, with index l, the partial key $\mathsf{g}\,\hat{}\,y$ marked as $\mathsf{RoundKey}$. We use $\mathsf{InEq}(i, k)$ to ensure that A is not the last participant. The rule also logs that this message has been consumed by way of Exp. It sends two messages on its authentic channel, both to the successor participant. One is its own partial key, $\mathsf{g}\,\hat{}\,y$, and the other is the successor's round key, $(\mathsf{g}\,\hat{}\,y)\,\hat{}\,esk$. In $\mathsf{WaitAccept}$, it stores its information to be used when accepting the group key in the last rule.

The last participant can execute the **Recv-Roundkey-Last** rule. The index k of A is the group size, and it receives a message from its predecessor B containing the round key $\mathsf{g}\,\hat{}\,y$ for k. This allows A to accept by logging the Accept fact with the group identifier, its name, its index, and the group key $(\mathsf{g}\,\hat{}\,y)\,\hat{}\,esk$. Note that it does not need to send any message as that has already happened in the **Recv-Others** rule steps for k.

All other participants accept using the last rule **Accept** and require their $\mathsf{WaitAccept}$ fact in the state. They receive the partial key assigned to them from the last participant B, which they raise to the power of their ephemeral secret key *esk* and log in the Accept their name, their index, and the group key $(\mathsf{g}\,\hat{}\,y)\,\hat{}\,esk$. Note that the index in the received message is $k+1$ for group size k

This concludes our description of the GDH rules and we now describe the verification. We prove the secrecy for the key of any group member, which is formalized in Figure 16. To prove this, we needed to specify some intermediate lemmas, which we outline next. TAMARIN automatically proves the lemmas and reuses them to prove the secrecy property.

Lemma presentation: We present three key lemmas here. Note that there are also nine more auxiliary lemmas. For those, and additional details, we refer the reader to [13]. Each lemma can be reused to prove subsequent lemmas and the actual proof goal. The key lemmas are:

1) All factors of an exponent that are ephemeral secret keys have been created by a participant *in an earlier round*.
2) For no round is the round key sent the same as one of the keys sent for another participant.
3) All factors of all received exponents are secret.

We explain the first lemma, depicted in Figure 17, in more detail. It states that for all ephemeral secret keys *esk* from round r_1 that also appear as a factor of an exponent in round r_2, it holds that $r_1 < r_2$. TAMARIN proves this using induction. It first proves an auxiliary lemma that all factors are ephemeral secret keys and afterwards that each of them has been created in an earlier round.

Protocol	Adversary model	Result	Time [s]
Group protocols:			
STR	authentic channels	proof	4.8
STR with signatures	PFS	proof	15.3
Group-Joux	authentic channels	proof	102.1
GDH	authentic channels	proof	152.7
Tripartite protocols:			
SIGJOUX	PFS	proof	90.7
SIGJOUX	PFS, eph-reveal	attack	99.8
TAK1	weakened eCK-like	proof	56.8
TAK1	eCK-like	attack	77.2
Identity-based protocols:			
RYY	wPFS	proof	8.3
RYY	wPFS, eph-reveal	attack	7.9
Scott	wPFS	proof	19.3
Scott	wPFS, eph-reveal	attack	26.2
Chen-Kudla	eCK-like	proof	61.0
Chen-Kudla	eCK	attack	45.3

TABLE I
OVERVIEW OF CASE STUDIES

E. Experimental Results

Table I provides an overview of our case studies (available at [13]). We discussed the models for the group protocols in detail in the previous section. Note that for STR we verified a model that uses authentic channels and a model using insecure channels with signatures where the adversary can reveal signing keys. For group Joux and for GDH we considered the case of authentic channels. For the two STR versions and Group Joux, TAMARIN performs the proof fully automatically and does not require any intermediate lemmas. For GDH, we give intermediate lemmas that TAMARIN can prove automatically.

Additionally, we analyzed two tripartite protocols: the signed Joux protocol and the TAK1 protocol from [21]. For the signed Joux protocol TAMARIN verifies that it satisfies perfect forward secrecy (PFS). If we additionally allow the adversary to learn ephemeral secret keys, TAMARIN finds an attack. For the TAK1 protocol, TAMARIN finds an attack against the eCK-security property that uses ephemeral key reveals combined with long-term key reveals. TAMARIN proves a weakened security property, which disallows the adversary from revealing both an ephemeral key and a long-term key.

Finally, we analyze three identity-based protocols that use bilinear pairings. TAMARIN shows that the RYY protocol [22] provides weak perfect forward secrecy (wPFS), but is vulnerable against ephemeral key reveal attacks. For the Scott protocol [23] we obtain a similar result. Our last protocol, Chen-Kudla [24], uses point addition. We do not support this operation and the required equalities, like $[c]([a]P + [b]Q) = ([c\,a]P + [c\,b]Q)$, in our model, and approximate the point addition with the associative and commutative operator $\sharp$. TAMARIN verifies that the protocol is secure in an eCK-like model, in which it is allowed to reveal the ephemeral keys of some, but not all agents involved in a session. If we remove this restriction, TAMARIN finds an attack.

VI. Related Work

There have been other applications of symbolic methods to find attacks on group key agreement protocols. In particular, the CORAL tool [6] was used to automatically find six attacks on three different protocols. CORAL can only perform falsification but not verification, i.e., the absence of an attack is not equivalent to a verification of security. Additionally, CORAL cannot handle the protocols covered in our work since it does not support Diffie-Hellman or bilinear pairings.

Meadows et al. used the NPA tool for the formal analysis of an abstract version of the GDOI protocol [25], which is a key transport protocol with a trusted central key server. Maude-NPA [26] is a more recent incarnation of the NPA tool based on rewriting logic. It is an automated tool capable of analyzing secrecy properties of DH protocols by using backwards narrowing with a DH-theory with the finite variant property. Maude-NPA does not support bilinear pairing. However, our bilinear pairing theory with the finite variant property from Section IV could be used as a basis to extend Maude-NPA with bilinear pairing.

Pankova et al. [27] give a transformation from a Horn theory modulo bilinear pairing to a Horn theory that can be analyzed with ProVerif, similar to the transformation given by Küsters and Truderung [28] for DH. In comparison to these two works, we can handle AC operators, do not require the so-called *exponent-ground* property of protocols, and provide support for specifying advanced security properties.

ProVerifList [7] uses a modified version of the basic ProVerif algorithm to enable the verification of protocols with unbounded lists. ProVerifList supports only a strict subset of ProVerif's features. In particular, it does not support equational theories.

There are also early results on the symbolic analysis of bilinear pairing protocols to establish computational guarantees. Kremer and Mazaré propose in [29] a manual symbolic analysis method for secrecy with respect to a passive adversary that is computationally sound. They extend Bellare-Rogaway's soundness results to bilinear pairing. In contrast, while we do not prove computational soundness, our approach deals with active adversaries and a large class of security properties.

VII. Conclusions

We have presented the first algorithm for the symbolic falsification and verification of group key agreement protocols using Diffie-Hellman exponentiation, bilinear pairing and AC-operators, by extending the algorithm underlying TAMARIN with support for AC-operators and bilinear pairing. Case studies show both the effectiveness and efficiency of our approach.

Our extensions are of independent interest. For example, the AC operators can be used to model lists, maps, and multisets. Bilinear pairings are used in many types of protocol, including identity-based protocols.

As future work, it would be interesting to analyze other variants of protocols that have subgroup keys, such as tree group Diffie-Hellman (TGDH) [30], analyze further advanced security properties, or include dynamic group operations such as join and leave. Furthermore, TAMARIN's scope may also facilitate the analysis of systems that use group protocols as components. For example, the SafeSlinger [31] protocol incorporates the STR protocol to generate shared keying material for secure communication among a group of participants at the same physical location.

Our current equational theory does not support point addition and group element multiplication, which is a restriction shared with the other automated approaches. We therefore cannot yet accurately model protocols such as HMQV or Burmester-Desmedt [32]. As future work, we would like to investigate support for such protocols.

REFERENCES

[1] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener, "A secure audio teleconference system," in *Proceedings on Advances in Cryptology, CRYPTO*. Springer-Verlag New York, Inc., 1990, pp. 520–528.

[2] R. Barua, R. Dutta, and P. Sarkar, "Extending Joux's protocol to multi party key agreement," *INDOCRYPT 2003*, pp. 33–60, 2003.

[3] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-Hellman key distribution extended to group communication," in *ACM Conference on Computer and Communications Security (CCS)*. ACM, 1996, pp. 31–37.

[4] O. Pereira and J. Quisquater, "Security analysis of the cliques protocols suites: first results," in *Trusted Information: The New Decade Challenge, 16th Annual Working Conference on Information Security, IFIP/Sec, IFIP Conference Proceedings*, 2001, pp. 151–166.

[5] M. Steiner, G. Tsudik, and M. Waidner, "Cliques: A new approach to group key agreement," in *Proceedings of the 18th International Conference on Distributed Computing Systems*. IEEE, 1998, pp. 380–387.

[6] G. Steel and A. Bundy, "Attacking group protocols by refuting incorrect inductive conjectures," *J. Autom. Reasoning*, vol. 36, no. 1-2, pp. 149–176, 2006.

[7] B. Blanchet and M. Paiola, "Automatic verification of protocols with lists of unbounded length," in *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2013, pp. 573–584.

[8] Y. Kim, A. Perrig, G. Tsudik *et al.*, "Communication-efficient group key agreement," in *Proceedings of the IFIP TC11 Sixteenth Annual Working Conference on Information Security: Trusted Information: The New Decade Challenge*. Kluwer, BV, 2001, pp. 229–244.

[9] B. Schmidt, S. Meier, C. Cremers, and D. Basin, "Automated analysis of Diffie-Hellman protocols and advanced security properties," in *Computer Security Foundations Symposium (CSF)*. IEEE, 2012, pp. 78–94.

[10] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN Prover for the Symbolic Analysis of Security Protocols," in *Computer Aided Verification, 25th International Conference, CAV 2013, Princeton, USA, Proc.*, ser. LNCS, vol. 8044. Springer, 2013, pp. 696–701.

[11] S. Meier, "Advancing automated security protocol verification," PhD dissertation, ETH Zurich, 2012.

[12] B. Schmidt, "Formal Analysis of Key Exchange Protocols and Physical Protocols," PhD dissertation, 2012.

[13] S. Meier, B. Schmidt, and C. Cremers, "The TAMARIN prover: source code and case studies ," March 2014, available http://tamarin-prover.github.io.

[14] D. Kapur, P. Narendran, and L. Wang, "Undecidability of unification over two theories of modular exponentiation," in *Seventeenth International Workshop on Unification (UNIF-2003), Valencia, Spain*, 2003.

[15] F. J. Thayer, J. C. Herzog, and J. D. Guttman, "Strand spaces: Proving security protocols correct," *Journal of Computer Security*, vol. 7, no. 1, pp. 191–230, 1999.

[16] S. Escobar, R. Sasse, and J. Meseguer, "Folding variant narrowing and optimal variant termination," *Journal of Logic and Algebraic Programming*, vol. 81, no. 7-8, pp. 898–928, 2012.

[17] J. Giesl, P. Schneider-Kamp, and R. Thiemann, "Automatic termination proofs in the dependency pair framework," in *IJCAR*, ser. LNCS, vol. 4130. Springer, 2006, pp. 281–286.

[18] F. Durán and J. Meseguer, "A Church-Rosser checker tool for conditional order-sorted equational Maude specifications," in *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Revised Selected Papers*, ser. LNCS, vol. 6381. Springer, 2010, pp. 69–85.

[19] ——, "A Maude coherence checker tool for conditional order-sorted rewrite theories," in *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Revised Selected Papers*, ser. LNCS, vol. 6381. Springer, 2010, pp. 86–103.

[20] H. Comon-Lundh and S. Delaune, "The finite variant property: How to get rid of some algebraic properties," in *RTA*, ser. LNCS, J. Giesl, Ed., vol. 3467. Springer, 2005, pp. 294–307.

[21] S. Al-Riyami and K. Paterson, "Tripartite authenticated key agreement protocols from pairings," *Cryptography and Coding*, pp. 332–359, 2003.

[22] E. Ryu, E. Yoon, and K. Yoo, "An efficient ID-based authenticated key agreement protocol from pairings," *NETWORKING 2004. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*, pp. 1458–1463, 2004.

[23] M. Scott, "Authenticated ID-based key exchange and remote log-in with simple token and pin number," *IACR eprint*, vol. 164, 2002.

[24] L. Chen and C. Kudla, "Identity based authenticated key agreement protocols from pairings," in *Computer Security Foundations Workshop (CSFW)*. IEEE, 2003, pp. 219–233.

[25] C. Meadows, P. F. Syverson, and I. Cervesato, "Formal specification and analysis of the group domain of interpretation protocol using NPATRL and the NRL protocol analyzer," *Journal of Computer Security*, vol. 12, no. 6, pp. 893–931, 2004.

[26] S. Escobar, C. Meadows, and J. Meseguer, "Maude-NPA: Cryptographic protocol analysis modulo equational properties," in *FOSAD*, ser. LNCS, vol. 5705. Springer, 2007, pp. 1–50.

[27] A. Pankova and P. Laud, "Symbolic analysis of cryptographic protocols containing bilinear pairings," in *Computer Security Foundations Symposium (CSF)*. IEEE, 2012, pp. 63–77.

[28] R. Küsters and T. Truderung, "Using ProVerif to analyze protocols with Diffie-Hellman exponentiation," in *Computer Security Foundations Symposium (CSF)*. IEEE, 2009, pp. 157–171.

[29] S. Kremer and L. Mazaré, "Computationally sound analysis of protocols using bilinear pairings," *Journal of Computer Security*, vol. 18, no. 6, pp. 999–1033, Sep. 2010.

[30] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 1, pp. 60–96, 2004.

[31] M. Farb, Y.-H. Lin, M. Burman, G. S. Chandok, J. McCune, and A. Perrig, "SafeSlinger: Easy-to-use and secure public key exchange," CyLab, Carnegie Mellon University, Tech. Rep., 2012, http://sparrow.ece.cmu.edu/group/pub/SafeSlinger.pdf.

[32] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in *Advances in Cryptology - EUROCRYPT'94*. Springer, 1995, pp. 275–286.