Symbolic Protocol Analysis With an Abelian Group Operator or Diffie-Hellman Exponentiation

Jonathan Millen and Vitaly Shmatikov*
Computer Science Laboratory
SRI International
{millen,shmat}@csl.sri.com

Abstract

We demonstrate that for any well-defined cryptographic protocol, the symbolic trace reachability problem in the presence of an Abelian group operator (*e.g.*, multiplication) can be reduced to solvability of a decidable system of quadratic Diophantine equations. This result enables complete, fully automated formal analysis of protocols that employ primitives such as Diffie-Hellman exponentiation, multiplication, and xor, with a bounded number of role instances, but without imposing any bounds on the size of terms created by the attacker.

1 Introduction

Conventional formal analysis of cryptographic protocols relies on the so called "Dolev-Yao" attacker model, which assumes that the attacker can intercept any message and construct or modify messages using a given set of computational and cryptographic primitives. Cryptographic operations are treated abstractly as black boxes, in the sense that they are assumed to have no computational features other than those associated with encryption and decryption. Black-box cryptographic primitives are characterized using simple axioms such as $\mathtt{dec}(\mathtt{enc}(x,k),k^{-1}) = x$, where k^{-1} is the inverse key to k, which might be either k itself for symmetric encryption, or the corresponding private key for public-key encryption. Sometimes even less is assumed: for example, in the free algebra model dec is not used explicitly (the consequences of this restriction are discussed in [Mil03]).

This rudimentary treatment of encryption is not adequate to deal with primitives such as xor (exclusive or), multiplication, and Diffie-Hellman exponentiation, which are widely used in security protocols. The attacker can and will exploit associativity, commutativity, cancellation, and other properties of these operations. For example, Bull's recursive authentication protocol was formally proved correct in a model that treated xor as an abstract encryption, and then found to be vulnerable once self-cancellation properties of xor are taken into account [Pau97, RS98].

We use *symbolic trace reachability* as the standard representation of the protocol analysis problem for trace-based security properties, which include secrecy and most authentication properties. This problem has been shown to be undecidable in several general settings

[*] Partially supported by ONR Grants N00014-01-1-0837 and N00014-03-1-0961, and by DARPA contract N66001-00-C-8014.

(for example, see [DLMS99]). Our approach follows the line of research that makes the reachability problem decidable by bounding the number of sessions, but allowing an unbounded attacker who may create messages of arbitrary depth [AL00, FA01, Bor01, MS01]. Other model checking approaches require *a priori* bounds on message complexity or may fail to terminate. Inductive proof approaches have no finiteness limitations, but require substantially more human effort per protocol, and are still subject to undecidability results.

In symbolic approaches, messages are represented as terms in an algebra generated by abstract computational primitives. Messages may contain variables, representing data fields whose value is not known in advance to the recipient. A variable can be viewed as the attacker's input to the protocol execution since the attacker can instantiate it with any term available to him as long as the instantiation is consistent in every term where the variable appears.

A trace is a sequence of messages sent and received. A trace is *reachable* if there is a substitution that instantiates all variables with ground terms such that all messages sent by the honest parties are consistent with the protocol specification, and all messages received by the honest parties from the network could have been constructed by the attacker from the previously sent messages and attacker's initial knowledge.

A trace is an attack if it violates the security condition – in the case of secrecy, if a value that is supposed to remain secret appears in the trace as an unencrypted received message (*i.e.*, is announced by the attacker). For a bounded number of sessions, the symbolic trace reachability problem has been shown to be NP-complete [RT01], assuming a free term algebra.

Our main contribution is to extend the constraint solving approach, first proposed in [MS01], to handle the algebraic properties of Abelian group operators. For any well-defined cryptographic protocol, we show that symbolic trace reachability is equivalent to solvability in integers of a certain system of quadratic equations. We then prove that this system is decidable. Decidability of the bounded-session protocol insecurity problem for `xor` (first shown in [CLS03, CKRT03b]) and for the free attacker algebra without equational properties (previously proved in [RT01, CCM01, MS01]) follow as special cases.

1.1 Overview

In Section 2, we introduce our formal model and describe how to reduce the protocol analysis problem to a sequence of symbolic constraints. In Section 3, we posit the *origination stability* condition, which is a necessary property of any well-defined protocol. In Section 4, we summarize the theory of ground term derivability in the presence of an Abelian group operator, due to Comon-Lundh and Shmatikov [CLS03].

The main technical result of the paper appears in Section 5. If the constraint sequence has a solution, we prove that it has a *conservative* solution. Intuitively, the conservative solution uses only the structure that is already present in the original sequence. We show that the substitution for any variable is a product of terms (and their inverses) drawn from a finite set: the non-variable subterms of the original constraint sequence. The resulting set of product derivation problems is naturally reduced to a system of quadratic Diophantine equations, as shown in Section 6. One of the steps along the way is Abelian group unification, which is known to be decidable [BS01]. We then proceed to demonstrate that the quadratic system has a solution if and only if a particular linear subsystem has a solution. Since linear Diophantine equations are decidable (*e.g.*, [CD94]), this establishes decidability of the protocol analysis problem in the presence of an Abelian group operator.

In Section 7, we extend our approach to protocols with Diffie-Hellman exponentiation, under the restriction that multiplication may appear only in exponents. We replace exponentials by a combination of products and uninterpreted functions, which reduces the symbolic analysis problem for such protocols to the solvability of a symbolic constraint sequence with an Abelian group operator. Conclusions are in Section 8.

1.2 Related work

Boreale and Buscemi [BB03] and Chevalier *et al.* [CKRT03a] recently developed decision procedures for protocol analysis in the presence of Diffie-Hellman exponentiation. Neither addresses decidability in the presence of an Abelian group operator. The decision procedure of [BB03] requires an *a priori* upper bound on the number of factors in each product, but the paper does not indicate how to compute this bound for a given symbolic trace. In general, establishing upper bounds on the size of variable instantiations needed for a feasible attack on a protocol is a highly non-trivial problem and one of the main challenges in proving decidability. Therefore, the technique of [BB03] cannot be considered a complete decision procedure even for protocols with Diffie-Hellman exponentiation.

Chevalier *et al.* [CKRT03a] proved that the insecurity problem for a restricted class of protocols is NP-complete in the presence of Diffie-Hellman exponentiation. They do not consider Abelian group operators outside exponents, and their result only applies to protocols in which no more than one new variable is introduced in each protocol message. Also, they do not permit variables to be instantiated with products. These restrictions are quite strong, ruling out some well-defined protocols. For example, a protocol in which an honest participant receives $x \cdot y$, then receives y, and then sends x is not permitted by the syntactic restrictions of [CKRT03a] (this protocol may be rewritten so as to satisfy the restrictions, but it is not clear whether there exists a general-purpose syntactic transformation that converts any protocol into one satisfying the restrictions of [CKRT03a]). In contrast, the results of this paper are directly applicable to any protocol which is *well-defined* in the following sense: an honest participant is not required to output the value of an attacker's variable *before* he has received any message containing that variable.

The technique of [CKRT03a] is more general in its treatment of Diffie-Hellman exponentiation since it allows exponentiation from an arbitrary base, while only constant-base exponentiation is considered in this paper. See [Shm04] for an extension of our constraint solving technique to modular exponentiation from an arbitrary base.

Pereira and Quisquater [PQ01] discovered an attack on a group Diffie-Hellman (GDH) protocol that exploits algebraic properties of Diffie-Hellman exponents. Their approach is specific to GDH-based protocols, and the attacker model is restricted correspondingly (*e.g.*, the attacker is not even equipped with the ability to perform standard symmetric encryption). They do not attempt to address the general problem of deciding whether a term is derivable in an attacker algebra with the equational theory of multiplication, or whether a particular symbolic attack trace has a feasible instantiation. Since they only consider the problem in the ground case, the resulting system of equations is linear, whereas the system we obtain in the general case with variables is quadratic (see Section 6). An application of our approach to one of the Pereira-Quisquater examples is summarized in Section 7.

Recent research by Narendran *et al.* focuses on decidability of unification modulo the equational theory of multiplication and exponentiation [MN02, KNW02, KNW03]. While equational unification is an important subproblem in symbolic protocol analysis, unification alone is insufficient to decide whether a particular symbolic attack trace is feasible.

Decidability of symbolic protocol analysis in the presence of xor has been proved in [CKRT03b, CLS03]. Chevalier *et al.* [CKRT03b] showed that the problem is NP-complete in a restricted protocol model which is very similar to the one proposed in this paper. Independently, Comon-Lundh and Shmatikov [CLS03] demonstrated decidability of symbolic protocol analysis with xor in the unrestricted model. This paper lifts the results of [CLS03] by considering the symbolic analysis problem in the presence of an arbitrary Abelian group operator, resulting in a substantially more complicated theory than in the xor case. In contrast, [CLS03] only considers Abelian group operators in the ground case, and obtains symbolic decidability results for xor only.

Bertolotti *et al.* [BDSV03] investigated cryptographic protocol analysis in the presence of associative and commutative operators. The algebraic theory considered in this paper is significantly more complicated. In protocols such as group Diffie-Hellman [STW96], the exponents form an Abelian group. In particular, the attacker can easily compute multiplicative inverses. To discover attacks such as that found by Pereira and Quisquater [PQ01], the algebraic theory must include inverses and cancellative reductions such as $t \cdot t^{-1} \rightarrow \mathbf{1}$. Demonstrating decidability in the presence of an Abelian group operator (rather than mere associativity and commutativity) is the main technical contribution of this paper.

2 Model

We begin with the strand space model of [THG99]. A *strand* is a sequence of *nodes* representing the activity of one party executing the protocol. Strands are finite and do not have branching or loops. Associated with each node is a message term with a sign, + or $-$, indicating that the message is sent or received, respectively. Messages in a strand are ground terms in a suitable algebra.

A protocol specification is a set of roles for legitimate parties in the protocol. A role can be instantiated with different data in different protocol sessions. Hence, roles in a protocol are specified as *strand schemas*, in which message terms may contain variables. A *role strand* is a partially (or fully or un-) instantiated strand schema that is a role; a *role instance* is a fully instantiated (ground) role strand.

In the usual strand space model, there is a standard set of *penetrator* roles representing primitive computations that an attacker can perform. An attack on a protocol may involve many concurrent sessions or role instances woven together with penetrator strands. A *bundle* is a collection of (protocol and penetrator) role instances in which the source of each received message is identified. Thus, nodes in a bundle are partially ordered by their strand sequence and also by the connection of a send node to a receive node for the same message. Bundles are backward complete in the sense that the strand predecessor of each (non-initial) node must be present, and the send node for each receive node must be present. A bundle is essentially a Lamport diagram [Lam78] in which the processes are strands. (Lamport called this a space-time diagram, but others renamed it in the context of distributed systems.)

2.1 Overview of constraint solving

It is shown in [THG99] and elsewhere how security questions can be reduced to questions about the existence of a bundle that exhibits a security violation. In our constraint solving approach begun in [MS01], bundle existence is determined by starting with a *semibundle* consisting of partially instantiated role instances, in which the sources of received messages are not necessarily determined. (The term "semibundle" comes from the Athena paper [Son99].) In a semibundle to be analyzed, the number of instances of each role has been chosen, and variables representing nonces (or session keys) have been instantiated to symbolic constants in the roles that generate them. The remaining variables are, for purposes of analysis, viewed as chosen or constructed by the attacker.

As in Athena, search for a solution begins with a semibundle that has no penetrator strands. Athena adds penetrator strands and role strands as necessary to extend the semibundle until it is a complete bundle. We never add role strands, because we bound the number of roles to begin with to achieve decidability. We never add penetrator strands, because their purpose is modeled implicitly by *derivation constraints*. We solve the constraints to see if the original semibundle can be instantiated to the role strands of a bundle. Unlike Athena, we solve the problem in an *infinite* state space that includes *all* possible combinations of penetrator strands.

A different sequence of derivation constraints is generated for each possible trace. Derivation constraints assert that each received message is derivable, using attacker term-generation rules, from messages that were previously sent in the trace. A solution instantiates variables in the semibundle so that ground terms representing received messages are all derivable. If the constraint sequence is not solvable for any of the possible traces, then an attack bundle does not exist for the given set of role strands (though one might exist for a larger set).

An efficient method for generating traces and solving a set of derivation constraints by applying rules for successive transformations of the constraint sequence is given in [MS01]. One important advance in that paper is the ability to handle non-atomic or constructed symmetric keys, that is, keys that may be the result of a combination of operations such as concatenation, encryption, and hashing. Some work was done subsequently to improve the efficiency of constraint generation and solving. Corin and Etalle devised an incremental approach [CE02] that has been adopted and incorporated into our own software tool. Recently, the AVISPA project made further improvements with a "constraint differentiation" approach [BMV03].

This paper focuses on the decidability of constraint solving in the extended model with Abelian group operations. The constraint solving step is different from that of [MS01], and consists mainly in reducing the constraint solving problem to a choice among a finite selection of substitutions, followed by solving a system of simultaneous linear Diophantine equations.

The solution approach presented here is aimed only at establishing decidability. Analyzing complexity and finding an efficient way to carry out protocol analysis are left to future work. A practical algorithm similar to that of [MS01] may work by gradual discovery of the right set of substitutions by successive unifications, but unification would have to be performed modulo the equational theory of Abelian groups, followed by solving a system of linear Diophantine equations. Practical techniques for solving linear Diophantine equations have already been developed in the context of associative-commutative unification [LC89].

2.2 Term algebra

To focus on decidability in the presence of an Abelian group operator, we use a simplified term algebra that includes only pairing, symmetric encryption (but not decryption), a one-argument function f modeling a one-way hash function, and an Abelian group operator written as multiplication. There are also assumed to be an unlimited number of variables and free constants (zero-argument functions). This is almost a free algebra, that is, one with no valid equations between terms (other than identities). But our term algebra is not free because multiplication forms an Abelian group, with unit **1** and a multiplicative inverse. The notation for these operations is shown in Fig. 1.

As in prior work with free algebras, there is no explicit decryption operator. Decryption is performed implicitly by protocol participants. The attacker's ability to extract components of a pair, or to decrypt an encrypted term when he knows the decryption key is modeled by separate attacker inference rules, which are discussed in Section 2.4.

The overall algebraic structure is described as the disjoint combination of a free theory and the Abelian group theory, following [SS89]. In this context, "disjoint" means that each relation involves only functions (and constants) from one theory at a time, in this case the group theory. However, any term is acceptable as an argument to any function. One way of viewing this is that all terms are untyped (or share a common base type, or sort). In particular, we do not distinguish between keys and other kinds of messages.

Actual cryptographic operators do have requirements on their arguments, at least on their size in bits, and in many cases more subtle restrictions. Protocol implementations depend on observing these restrictions. For example, our algebra would allow a term like $\{t\}_{\langle k,k'\rangle}$, but a protocol would normally have to apply a type coercion operator (a hash or truncation, perhaps) to $\langle k, k'\rangle$ before it could be used as a key. Moreover, when the Abelian group operator is applied to a compound term (*e.g.*, a pair), in the actual implementation the corresponding bitstring must be interpreted as an element of the right group, which may or may not be possible. We trust the protocol specification in this respect. If terms appearing in the abstract protocol specification involve application of the Abelian group operator to compound terms, we assume that such terms can be mapped into the group. Nevertheless, due to our abstract treatment of cryptographic functions, our analysis may generate unimplementable attacks, *e.g.*, those involving application of the Abelian group operator to ciphertexts encrypted with a symmetric key, *etc.* These spurious attacks can be recognized by static inspection and discarded.

It is natural to ask here about the relationship between ground terms in our term algebra and the bitstrings they are mapped to in the protocol implementation. This is not an easy question to answer because of the level of abstraction of our model (and all Dolev-Yao-style [DY83] models in general). For example, regarding encryption as a free operator means that the infinite sequence of terms $t, \{t\}_k, \{\{t\}_k\}_k, \ldots$ are all distinct, whereas in practice the bitstring values would all have the same length and could not all be distinct. Thus, there are additional relations in reality, and this implies that there may be more attacks on the real protocol than on the abstract version. Such concerns are addressed in work on computationally sound formal models, which is beyond the scope of this paper. The Abadi-Rogaway paper [AR02] is a good introduction to this issue.

Additional relations may also come about because of particular ways that encryption is performed. Encryption can be accomplished using exponentiation or multiplication, for example, and some interaction with the multiplicative Abelian group operator could occur, especially if they both use modular arithmetic. Such design choices might lead to attacks that would not be discovered in the present model. However, our decision procedure will determine the existence of any attack strategies that work *uniformly*, that is, for all possible cryptographic implementations that satisfy the abstract axioms.

We describe a specific extension with exponentials in Section 7, for application to pro-

$\langle t_1, t_2 \rangle$	Pairing of terms t_1 and t_2.
$\{t_1\}_{t_2}$	Term t_1 encrypted with term t_2 using a symmetric algorithm.
$t_1 \cdot \ldots \cdot t_n$	Product of terms (associative and commutative).
t^{-1}	Multiplicative inverse of term t.
$f(t)$	Any free function.

Figure 1: Message term constructors

$$\begin{array}{rcl} t \cdot \mathbf{1} & \rightarrow & t \\ t \cdot t^{-1} & \rightarrow & \mathbf{1} \\ & & \\ (t^{-1})^{-1} & \rightarrow & t \\ (t_1 \cdot t_2)^{-1} & \rightarrow & {t_2}^{-1} \cdot {t_1}^{-1} \end{array}$$

Figure 2: Normalization rules for products and inverses

tocols using Diffie-Hellman key agreement, while still allowing ordinary symmetric encryption in the protocol. The idea is to transform constraints with exponential terms into constraints with products of exponents. Other extensions are possible with no conceptual difficulty. Abstract public key encryption can be handled in a way similar to symmetric encryption, using a pair of functions $pk(a)$ and $sk(a)$ to generate a pair of keys for a principal a. Variations and extensions of pairing could also be added without affecting decidability, such as n-tuples for some or all $n > 2$, or associative concatenation.

The associative and commutative properties of our multiplicative group allow us to regard the product as an operator on a set of any number of terms, as suggested by the extended product notation $t_1 \cdot \ldots \cdot t_n$.

Terms can be put in *normal form* by applying the reduction rules given in Fig. 2. A term is in normal form if no further reductions are possible, even after rearranging products using associativity and commutativity. Normalization also includes “flattening” products, so that $(a \cdot b) \cdot c$ will be normalized to $a \cdot b \cdot c$. The normal form is unique up to permutations of the extended product. Thus, for example, $(a \cdot b) \cdot (a^{-1} \cdot c)$ reduces to either $b \cdot c$ or $c \cdot b$, and these two productss are regarded as equal. We assume that terms are always normalized.

A term with a positive integer exponent is defined in the expected way; for example, $a^2 = a \cdot a$. A term with a negative exponent, like t^{-n}, is an abbreviation for an inverse with a positive exponent, like $(t^{-1})^n$.

2.3 Unification

There is a unification algorithm that can be applied to any two terms t_1 and t_2 constructed using the syntax of Fig. 1, by combining conventional structural unification on the free operators and Abelian Group (AG-) unification modulo associativity and commutativity of the $\cdot$ operator and normalization rules of Fig. 2. If both terms to be unified are not products, Unpairing (UL, UR)

$$\frac{T \vdash \langle u, v\rangle}{T \vdash u} \qquad \frac{T \vdash \langle u, v\rangle}{T \vdash v}$$

Decryption (D)

$$\frac{T \vdash \{u\}_v \quad T \vdash v}{T \vdash u}$$

Pairing (P)

$$\frac{T \vdash u \quad T \vdash v}{T \vdash \langle u, v\rangle}$$

Encryption (E)

$$\frac{T \vdash u \quad T \vdash v}{T \vdash \{u\}_v}$$

Function (F)

$$\frac{T \vdash u}{T \vdash f(u)}$$

Inversion (I)

$$\frac{T \vdash u}{T \vdash u^{-1}}$$

Multiplication (M)

$$\frac{T \vdash u_1 \quad \ldots \quad T \vdash u_n}{T \vdash u_1 \cdot \ldots \cdot u_n}$$

Figure 3: Attacker's capabilities

we use conventional unification by structural recursion, treating all constructors of Fig. 1 as free functions. If at least one of the terms is a product, unification is performed modulo AG, which is known to be decidable [BS01] and which produces a finite number of most general unifiers. In the rest of the paper, we use $MGU_{\mathrm{AGF}}(t_1, t_2)$ notation for the finite set of most general unifiers of t_1 and t_2. Note that because a non-product term may contain products as inner subterms, t_1 and t_2 may have more than one most general unifier even if neither is a product.

2.4 Attacker model

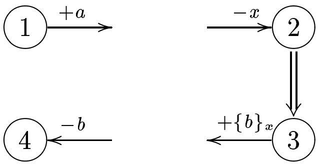
We use the standard attacker model augmented with rules concerning products and inverses (an extension to exponentials can be found in Section 7). The attacker's ability to derive terms is characterized as a term closure under the inference rules of Fig. 3. The sequent $T \vdash u$ means that u is derivable (computable) from the terms in the set T. The term closure of T is the set of all terms derivable from T (including members of T).

2.5 Constraint generation

Suppose we are given a protocol specified as a set of roles (strand schemas). We first choose a finite set of role strands for the semibundle. (There is no algorithm to determine how many of each are needed.) Each role may be instantiated zero or more times. In each role strand, any nonce variable generated by that role is instantiated with a distinct symbolic constant.

As an example, consider the protocol with two roles $+x$ and $-x + \{y\}_x$, where x and y are nonces, and the security policy is to keep y secret. In the semibundle pictured below, x has been instantiated with a in the first role strand, which generates it, and y has been instantiated with b.

The third strand (node 4) is an artificial "test" strand introduced to detect compromise of b. If b can be received (in the clear) by the test party, then b has been compromised.



No variables remaining in a semibundle should occur in more than one strand. Even though role specifications may use the same variable like A or K in different roles because the value is expected to be the same, there is no guarantee that corresponding variables will be instantiated with the same value during execution.

We generate all possible node orderings, or traces, that are consistent with the given strands. This is a finite set that grows exponentially with the number of strands. (Some traces can be discarded safely, but for proving decidability we may as well assume that we have all of them.) Each trace yields a sequence of derivation constraints.

In general, if $u_1, \ldots, u_k$ is the sequence of receive-node messages, and T_i is the set of messages sent in nodes prior to the node in which u_i is received, then the constraints are just the sequence

$$\mathbf{C} = \{T_i \rhd u_i\}.$$

Each individual constraint $T_i \rhd u_i$ can be interpreted as "at step i, the attacker knows messages in T_i and needs to generate message u_i." We will refer to u_i as the *target term* of the constraint, and T_i as the *source set* of the constraint. Both u_i and messages in T_i may contain variables. We assume that T_1 contains terms that are initially known to the attacker, such as constants specific to the protocol and the attacker's own long-term keys. It is usually not necessary to include the constant $\mathbf{1}$, since if T_1 contains any term t, the attacker can derive $\mathbf{1}$ as $t \cdot t^{-1}$.

The properties of protocol-generated sequences are discussed in Section 3.

Our example semibundle has traces corresponding to all node orderings that respect the ordering of nodes 2 and 3. Note also that the only traces of interest are attacks, which end with node 4. Thus, the complete set of orderings to examine is 1234, 124, 14, 2134, 2314, 214, 24, 4. (One can show that it is sufficient to examine 1234 and 14, since any attack possible with a different ordering is possible with one of these.) The ordering 1234 generates the constraint sequence

$$\begin{array}{l} a \rhd x \\ \{a, \{b\}_x\} \rhd b. \end{array}$$

For convenience, we simplify the notation for source sets by regarding a list as a union. Thus, $\{a, \{b\}_x\} \rhd u$ may be written $a, \{b\}_x \rhd u$ and $T \cup \{a\} \rhd u$ may be written $T, a \rhd u$.

We say that σ is a *solution* of $T \rhd u$ (written $\sigma \Vdash T \rhd u$) if σ is a ground substitution such that either $u\sigma \in T\sigma$, or $T\sigma \vdash u\sigma$ is derivable using the inference rules of Fig. 3.

Given a constraint sequence $\mathbf{C} = \{T_i \rhd u_i\}$, σ is a solution of the constraint sequence ($\sigma \Vdash \mathbf{C}$) if σ simultaneously solves every constraint $T_i \rhd u_i$.

The constraint sequence arising from the trace 1234 in the example can be satisfied with the substitution $x \mapsto a$, since the attacker's Decryption rule (D) can be applied to satisfy the second constraint.

2.6 Subterms and product closures

We introduce a few definitions for convenience. If T is a finite set of terms, let $\mathrm{St}(T)$ be the set of *subterms* of T, which is the least set of terms such that:

- If $t \in T$ then $t \in \mathrm{St}(T)$
- If $\langle u, v\rangle \in \mathrm{St}(T)$ then $u, v \in \mathrm{St}(T)$
- If $\{u\}_v \in \mathrm{St}(T)$ then $u, v \in \mathrm{St}(T)$
- If $f(u) \in \mathrm{St}(T)$ then $u \in \mathrm{St}(T)$
- If $u_1 \cdot \ldots \cdot u_n \in \mathrm{St}(T)$ then $u_i \in \mathrm{St}(T)$ for each i

Note that $u_1 \cdot u_2$ is not considered a subterm of $u_1 \cdot u_2 \cdot u_3$. For an individual term t, $\mathrm{St}(t) = \mathrm{St}(\{t\})$. We say that t is a *superterm* of u if $u \in \mathrm{St}(t)$.

Define

$$\begin{array}{rcl}
\mathrm{PC}(T) & \overset{def}{=} & \{t_1 \cdot \ldots \cdot t_n \quad \mid (\forall i)\ t_i \in T \text{ or } {t_i}^{-1} \in T\} \\
\mathrm{St}(\mathbf{C}) & \overset{def}{=} & \bigcup_{T_i \rhd u_i \in \mathbf{C}} \mathrm{St}(T_i \cup \{u_i\}) \\
\mathrm{Var}(\mathbf{C}) & \overset{def}{=} & \mathrm{Var}(\mathrm{St}(\mathbf{C})) \\
\mathrm{Var}(T) & \overset{def}{=} & \{x \in \mathrm{St}(T) \mid x \text{ is a variable}\} \\
\mathcal{S}(\mathbf{C}) & \overset{def}{=} & \mathrm{St}(\mathbf{C}) \setminus \mathrm{Var}(\mathbf{C}) \\
\bar{\mathcal{S}}(\mathbf{C}) & \overset{def}{=} & \mathrm{PC}(\mathcal{S}(\mathbf{C}))
\end{array}$$

Thus, $\mathcal{S}(\mathbf{C})$ is the set of all non-variable subterms of $\mathbf{C}$, and $\bar{\mathcal{S}}(\mathbf{C})$ is the closure of this set under product ($\cdot$) and inverse.

3 Well-Defined Protocols and Constraint Sequences

We start by defining two properties of constraint sequences that are essential for establishing decidability: *monotonicity* and *origination*. Conceptually, these properties are similar to those defined for the constraint solving method of [MS01]. Informally, *monotonicity* means that the attacker's knowledge never decreases as the protocol progresses: all messages intercepted by the attacker are simply added to the set of terms available to him. *Origination* means that each variable appears for the first time in some message generated by the attacker (recall that in the symbolic analysis approach, variables model attacker's input to the protocol execution).

Our proof of decidability requires that monotonicity and origination be preserved by any partial substitution (this is a technical difference from [MS01]). In this section, we argue that this is true for any symbolic constraint sequence associated with a *well-defined* protocol. Our notion of well-definedness is formalized below, but it can be informally understood as follows. For any choice of attacker's inputs to the protocol execution, the protocol should never require an honest participant to generate a message containing an attacker's input *before* the attacker sent any message with that input.

Although a constraint sequence must be solved by a ground substitution, the substitutions we work with below are not always ground substitutions. We do assume, for convenience, that all substitutions are *idempotent*. An idempotent substitution eliminates every variable it instantiates. That is, if we define the domain $\mathcal{D}(\theta) = \{x|x\theta \neq x\}$, then $y \in \mathrm{St}(x\theta)$ implies $y \notin \mathcal{D}(\theta)$. Furthermore, a substitution does not introduce new variables in the context of a constraint sequence: $\mathrm{Var}(\mathbf{C}\theta) \subset \mathrm{Var}(\mathbf{C})$.

3.1 Monotonicity and origination

Let $\mathbf{C} = \{T_1 \rhd u_1;\ \ldots;\ T_n \rhd u_n\}$ be any constraint sequence generated from a protocol.

Definition 3.1 (Monotonicity) $\mathbf{C}$ *satisfies the* monotonicity property *if* $j < i$ *implies that* $T_j \subseteq T_i$.

This property means that the attacker does not "forget" terms. As the protocol progresses, the set of the terms available to the attacker does not decrease. The constraint generation procedure described in Section 2.5 produces monotonic constraint sequences. Furthermore, this property is preserved by substitutions.

To understand the origination property, recall that variables represent the attacker's input to the protocol execution. Therefore, each variable must appear for the first time in some message generated by the attacker, *i.e.*, in some message *received* by an honest party.

Definition 3.2 (First occurrence) *Given a constraint sequence* $\mathbf{C} = \{T_i \rhd u_i\}$*, define* $k_x(\mathbf{C})$ *for any variable* $x \in \mathrm{Var}(\mathbf{C})$ *to be the index of the constraint in which* x *appears for the first time,* i.e., $x \in \mathrm{Var}(T_{k_x} \rhd u_{k_x})$*, but* $\forall i < k_x\ x \notin \mathrm{Var}(T_i \rhd u_i)$*. Where* $\mathbf{C}$ *is clear from the context, we will refer to* $k_x(\mathbf{C})$ *simply as* k_x.

Definition 3.3 (Origination) *A constraint sequence* $\mathbf{C} = \{T_i \rhd u_i\}$ *satisfies the* origination property *if* $\forall y \in \mathrm{Var}(T_{k_x})\ \ k_y < k_x$.

Origination implies that the first occurrence of a variable is in a target term, since, if $x \in \mathrm{Var}(T_{k_x})$, we would have $k_x < k_x$, which is impossible.

In order to ensure this property for the initial constraint sequence generated from a protocol specification, we observe two conventions. First, nonces generated by a role are instantiated with new symbolic constants in that role. This ensures (in a free constant context) that all nonces are different. Second, we require that any other variables chosen by a role, such as the choice of responder principal made by an initiator, and the choice of initiator as well, are either instantiated with constants (for the strand containing a shared secret), or are placed in a prior received message, as if the attacker chose these values. This is artificial, but it makes sense for analysis, since we want to find attacks in which these values are chosen in a worst-case way, and we may as well assume that the attacker has chosen them.

For example, the one-message protocol $A \rightarrow B : A, N$ would have an additional message $E \rightarrow A : A, B$ inserted (E for enemy or attacker). We assume that the attacker initially knows constants a, b, e representing principals' names. We can model this with another, earlier message $A \to E : \langle\langle a, b\rangle, e\rangle$. The two role strand schemas are then

$A : +\langle\langle a, b\rangle, e\rangle - \langle x_a, x_b\rangle + \langle x_a, x_n\rangle$ and
$B : -\langle y_a, y_n\rangle$.

In a semibundle with one A strand and one B strand, the nonce N would be instantiated in A's role, producing the constraint sequence

$a, b, e \rhd \langle x_a, x_b\rangle$
$a, b, e, x_a, n \rhd \langle y_a, y_n\rangle$.

(In this example we assumed that constants a, b, e are known initially to the attacker, and n is a new constant for the nonce generated by A.)

3.2 Well-defined protocols

For subsequent results, we need the property that origination is preserved by substitutions. This property will follow from a condition, defined below, called *well-definedness*, which is intuitively equivalent to requiring that honest participants make *no undetermined choices*. If a protocol is presented in a way that allows an undetermined choice, we want to analyze the protocol as though the attacker made the choice.

To explain our notion of an ill-defined protocol, suppose a protocol specification requires an honest participant to receive $x \cdot y$ at a point in the protocol where he has never received any messages containing x or y before, and then return x. This is an undetermined choice, because an honest recipient needs to split a product of two unknown values. Recall that the attacker has complete control over the values of x and y. Suppose the attacker chooses x and y so that $x = y^{-1}$ in some execution of the protocol. Then, in this execution of the protocol, the honest participant will receive $\mathbf{1}$ (since $y^{-1} \cdot y = \mathbf{1}$), and will then have to return y^{-1}, even though he has not received any prior message containing y and thus has no way of knowing what value was chosen by the attacker for y. In this example, the protocol is not implementable as specified, since there is no way for the honest participant to determine which value of y the attacker had in mind.

We stress that non-implementability of this protocol is unrelated to hardness of factorization. Even if $x \cdot y$ had unique factorization and the honest participant had *unlimited* computational power, he would still have no way of determining which of the two factors was chosen by the attacker as x (because $\cdot$ is commutative, values of x and y are completely symmetric). In this case, the honest participant has to make an undetermined choice between two candidates, which means that the protocol cannot be implemented as specified.

In contrast, suppose a protocol specification requires an honest participant to receive $x \cdot y$, then receive y, and then return x. The correct choice of x is then determined and computable as $x = (x \cdot y) \cdot y^{-1}$. This protocol is well-defined.

The ability of an honest participant to compute the messages he is required to send by the protocol specification is no different in principle from the ability of the attacker to compute the messages received by honest participants (although honest participants may have access to term generation rules other than those of Fig. 3). The honest participants' computations required by the protocol are, roughly, the mirror image of the attacker's computations. Thus, if a constraint sequence $\mathbf{C}$ is generated from a trace of a well-defined protocol, each attacker term set T_i must be computable from terms previously received by honest participants, namely $\{u_j | j < i\}$, plus any initial knowledge of honest participants such as their own secret keys. (At each step, an honest participant needs only to generate the difference $T_i \setminus T_{i-1}$.) This observation will lead us to the formal definition of well-definedness below.

We start by definining the honest participants' *knowledge* at each step of the symbolic protocol trace. Each $\mathcal{K}_i$ is a set of terms. Even though there may be more than one honest participant, we combine all honest participants' knowledge into one set, because this is sufficient for our purpose.

Definition 3.4 *Let* $\mathbf{C} = \{T_i \rhd u_i\}$ *be a constraint sequence generated from the protocol, and let* $\mathcal{K}_0$ *be the set of terms collectively known to honest participants before the start of protocol execution. For all* i *such that* $1 \leq i \leq |\mathbf{C}|$, *define* $\mathcal{K}_i = \mathcal{K}_0 \cup \{u_j | j < i\}$.

Note that $\mathcal{K}_0$ does not contain any terms with variables with them. The reason for this is the origination property, which requires that every variable must appear for the first time in some message sent by the attacker, and no messages had been sent before the protocol started.

Consider the computation that the honest participants must execute according to the protocol specification. In a sense, it is the mirror image of the attacker computation. Recall that each constraint $T_i \rhd u_i$ means that the attacker needs to generate u_i from terms in T_i. For honest participants, the reverse is true: they must generate terms in T_i from their knowledge $\mathcal{K}_i$.

Informally, a protocol is well-defined if it does not require honest participants to output a message containing a variable that they have never observed before. Moreover, this condition must hold for *any* value of variables. Since variables are controlled by the attacker and represent the attacker's input to the protocol execution, the protocol must be well-defined for any choice of these values. Intuitively, this means that a participant's behavior in the protocol must be well-defined regardless of how other participants choose their nonces, what they send in lieu of ciphertexts encrypted by unknown keys, and so on.

Definition 3.5 (Well-defined protocol) *A protocol is* well-defined *if every symbolic constraint sequence* $\mathbf{C} = \{T_i \rhd u_i\}$ *generated from its specification satisfies the following property: for any* i, *for any partial substitution* θ, $\mathrm{Var}((T_i \setminus T_{i-1})\theta) \subseteq \bigcup_{j<i} \mathrm{Var}(u_j\theta)$,

As mentioned above, an example of an ill-defined protocol is a protocol in which an honest participant receives $x \cdot y$ and is then required to send x. This means that honest participants must produce x from their knowledge $\mathcal{K}_0 \cup \{x \cdot y\}$ where $\mathrm{Var}(\mathcal{K}_0) = \emptyset$. Observe that there exist variable values (modeled as partial substitutions), *e.g.*, $\theta = [y \mapsto x^{-1}]$, such that in the resulting concrete protocol execution the honest participant's knowledge is $\mathcal{K}_0 \cup \{\mathbf{1}\}$ and he is required to output x even though he has never seen any terms containing x before. This protocol is not well-defined. On the other hand, a protocol in which an honest participant is required to output x when his knowledge is $\mathcal{K}_0 \cup \{x \cdot y\} \cup \{y\}$ is well-defined. Under any partial substitution θ, $\mathrm{Var}(x\theta) \subseteq \mathrm{Var}(\{x\theta \cdot y\theta, x\theta\})$.

We argue that any protocol that does not force honest participants to make undetermined choices satisfies Definition 3.5. Otherwise, the behavior of some honest participant is not defined for some values of attacker's inputs. For those values, the honest participant is expected to output the value of an attacker's variable before the attacker has sent any messages containing that value.

If a protocol designer wishes to check whether his protocol specification is well-defined, Definition 3.5 may be difficult to verify to practice because, for each constraint sequence generated from the protocol specification, it quantifies over all possible substitutions. In appendix A, we explain how to check whether a constraint sequence satisfies Definition 3.5 by considering only a finite number of substitutions (namely, those that may lead to cancellation of variables in the knowledge of honest participants).

We now prove a simple auxiliary lemma that will help demonstrate stability of origination under any partial substitution.

Lemma 3.6 *If the protocol is well-defined according to Definition 3.5, and* $\mathbf{C} = \{T_i \rhd u_i\}$ *is the corresponding symbolic constraint sequence, then, for any partial substitution* θ*,* $\forall i\ \mathrm{Var}(T_i\theta) \subseteq \bigcup_{j<i} \mathrm{Var}(u_j\theta)$*.*

Proof: Proof is by induction over all constraints in $\mathbf{C}$. By Definition 3.3, $\mathrm{Var}(T_1) = \emptyset$, thus the lemma is satisfied vacuously. Suppose the lemma holds for all $T_j \rhd u_j \in \mathbf{C}$ where $j < i$. Consider $T_i \rhd u_i$. By Definition 3.5, $\mathrm{Var}((T_i \setminus T_{i-1})\theta) \subseteq \bigcup_{j<i} \mathrm{Var}(u_i\theta)$. By the induction hypothesis, $\mathrm{Var}(T_{i-1}\theta) \subseteq \bigcup_{j<i-1} \mathrm{Var}(u_j\theta) \subseteq \bigcup_{j<i} \mathrm{Var}(u_j\theta)$. Therefore $\mathrm{Var}(T_i\theta) = \mathrm{Var}((T_i \setminus T_{i-1})\theta) \cup \mathrm{Var}(T_{i-1}\theta) \subseteq \bigcup_{j<i} \mathrm{Var}(u_j\theta)$. This completes the induction. □

3.3 Stability of monotonicity and origination

This section contains the results which are used in the rest of the paper.

Theorem 3.7 (Stability of monotonicity) *Let* $\mathbf{C} = \{T_1 \rhd u_1; \ldots; T_n \rhd u_n\}$ *be a constraint sequence generated from a protocol. Then* $j < i$ *implies* $T_j\theta \subseteq T_i\theta$*.*

Proof: Observe that for all $t' \in T_j\theta$ there exists $t \in T_j$ such that $t' = t\theta$. Since $\mathbf{C}$ is generated from the protocol, it satisfies Definition 3.1. Therefore, if $j < i$, then for all $t \in T_j), t \in T_i$, and for all $t' \in T_j\theta$, there exists $t'' \in T_i\theta$ such that $t' = t'' = t\theta$. Therefore, $T_j\theta \subseteq T_i\theta$. □

Theorem 3.8 (Origination stability) *Let* $\mathbf{C} = \{T_1 \rhd u_1; \ldots; T_n \rhd u_n\}$ *be a constraint sequence generated from a well-defined protocol. For any partial substitution* θ*,* $\mathbf{C}\theta$ *satisfies the origination property.*

Proof: Stability of origination under any substitution follows directly from Lemma 3.6. As described in Section 3.3, $\mathbf{C}$ satisfies the origination property by construction. Consider any partial substitution θ, let x be some variable occurring in $\mathbf{C}\theta$ and let k_x be the index of the constraint in which it occurs for the first time. By Lemma 3.6, $\mathrm{Var}(T_{k_x}\theta) \subseteq \bigcup_{j<i} \mathrm{Var}(u_j\theta)$. Therefore, for any variable $y \in \mathrm{Var}(T_{k_x}\theta)$, y occurs in some $u_j\theta$ where $j < i$. We conclude that $k_y < k_x$, and the origination property is satisfied. □

4 Ground Derivability

In this section, we outline the theory of ground term derivability in the attacker model with an Abelian group operator due to Comon-Lundh and Shmatikov [CLS03]. We only state the key lemmas. Proofs can be found in [CLS03]. Note that in [CLS03], Abelian groups were considered only in the ground case. In contrast, this paper is devoted to solving the problem in the *symbolic* case.

The normalization property stated in Lemma 4.5 may appear superficially similar to the analysis-followed-by-synthesis closure previously established for the free attacker algebra [Pau98, CJM00]. Normalization results of [Pau98, CJM00] do not apply, however, to attacker models with non-atomic encryption keys and equational theories with cancellation, requiring development of a new proof normalization theory such as [CLS03]. For example, a cannot be derived from $T = \{\{a\}_{\langle k,k\rangle}, k\}$ by any sequence in which analysis steps are followed only by synthesis steps.

Definition 4.1 (Ground proof) *A* ground proof *of $T \vdash u$ is a tree labeled with sequents $T \vdash v$ such that all terms in T, u and v are ground and:*

- *Every leaf is labeled with $T \vdash v$ such that $v \in T$*
- *Every non-leaf node labeled with $T \vdash v$ has parents $s_1, \ldots s_n$ such that* $\dfrac{s_1 \ \cdots \ s_n}{T \vdash v}$ *is an instance of one of the inference rules of Fig. 3*
- *The root is labeled with $T \vdash u$*

The *size* of a proof is the number of its nodes.

Informally, there exists a proof of $T \vdash u$ if and only if the attacker can construct term u from the term set T using its capabilities as defined by the rules of Fig. 3.

Lemma 4.2 *If there is a minimal size proof $\mathcal{P}$ of one of the following forms:*

$$\dfrac{\dfrac{\vdots}{T \vdash \langle u,v\rangle}}{T \vdash u} \qquad \dfrac{\dfrac{\vdots}{T \vdash \langle v,u\rangle}}{T \vdash u} \qquad \dfrac{\dfrac{\vdots}{T \vdash \{u\}_v} \quad \dfrac{\vdots}{T \vdash v}}{T \vdash u}$$

then $\langle u,v\rangle \in \mathrm{St}(T)$, $\langle v,u\rangle \in \mathrm{St}(T)$, or $\{u\}_v \in \mathrm{St}(T)$, respectively.

Proof: This lemma was proved in [CLS03]. □

Definition 4.3 (Proof composition) *If $\mathcal{P}_i$ is a proof of $T \vdash v_i$ for $i = 1, \ldots, n$ and $\mathcal{C}$ is a proof of $\{v_i | i = 1, \ldots, n\} \vdash u$, then the* composition $\mathcal{C}[\mathcal{P}_1, \ldots, \mathcal{P}_n]$ *is the proof of $T \vdash u$ constructed by putting proofs $\mathcal{P}_i$ together in the obvious way.*

Definition 4.4 (Normal ground proof) *A ground proof $\mathcal{P}$ of $T \vdash u$ is* normal *if*

- *either $u \in \mathrm{St}(T)$ and every node of $\mathcal{P}$ is labeled $T \vdash v$ with $v \in \mathrm{St}(T)$,*

- *or $\mathcal{P} = C[\mathcal{P}_1, \ldots, \mathcal{P}_n]$ where every proof $\mathcal{P}_i$ is a normal proof of some $T \vdash v_i$ with $v_i \in \mathrm{St}(T)$ and proof C is built using the inference rules (P),(E),(F),(M),(I) only.*

Lemma 4.5 (Existence of normal ground proof) *If there is a ground proof of $T \vdash u$, then there is a normal ground proof of $T \vdash u$.*

Proof: This lemma was proved in [CLS03] for an attacker theory which is identical to that of Fig. 3, but without (F) rule. The proof can be trivially extended to account for (F). □

Lemma 4.6 *If there exists a proof of $T \vdash t$ and $c(u, v) \in \mathrm{St}(t)$, where $c(u, v)$ is either $\langle u, v\rangle$, $\langle v, u\rangle$, or $\{u\}_v$, then either*

- *$c(u, v) \in \mathrm{St}(T)$, or*
- *there exist normal proofs of $T \vdash u$ and $T \vdash v$ in which no nodes are labeled with $T \vdash c(u, v)$.*

Proof: We prove the lemma for $\langle u, v\rangle$ (the proofs for $\langle v, u\rangle$ and $\{u\}_v$ are similar).

Suppose $\langle u, v\rangle \in \mathrm{St}(t)$, but $\langle u, v\rangle \notin \mathrm{St}(T)$. Consider the minimal size normal proof of $T \vdash t$ (such a proof must exist by Lemma 4.5). By Definition 4.4, this proof has the form $C[\mathcal{P}_1, \ldots, \mathcal{P}_n]$ where $\mathcal{P}_i$ is the proof of some $u_i \in \mathrm{St}(T)$, and context C is built using the inference rules (P),(E),(F),(M),(I) only.

By assumption, for all i, $\langle u, v\rangle \notin \mathrm{St}(u_i)$ (otherwise, there is a contradiction with $\langle u, v\rangle \notin \mathrm{St}(T)$ since $u_i \in \mathrm{St}(T)$, thus $\mathrm{St}(u_i) \subseteq \mathrm{St}(T)$). Consider the *first* inference in C that results in the appearance of $\langle u, v\rangle$, *i.e.*, $\dfrac{T \vdash t_1 \ \ldots \ T \vdash t_k}{T \vdash t'}$ such that $\langle u, v\rangle \in \mathrm{St}(t')$ but for all j, $\langle u, v\rangle \notin \mathrm{St}(t_j)$. Let $\mathcal{P}_j$ be the subproof of $T \vdash t_j$. By minimality of the proof, $\forall t_j$, no node in the subproof of $T \vdash t_j$ is labeled with $T \vdash \langle u, v\rangle$.

For rules (E),(F),(M),(I), if $\langle u, v\rangle \in \mathrm{St}(t')$, then there exists j such that $\langle u, v\rangle \in \mathrm{St}(t_j)$. The condition $(\forall j)\langle u, v\rangle \notin \mathrm{St}(t_j)$ can hold only if (i) the inference rule in question is (P), and (ii) $t' = \langle u, v\rangle$. Therefore, the inference has the form $\dfrac{T \vdash u \quad T \vdash v}{T \vdash \langle u, v\rangle}$. In this case, $u = t_1$. Therefore, no node of the subproof of $T \vdash u$ is labeled with $T \vdash \langle u, v\rangle$. □

5 Conservative Solutions

We will use the ground derivability results of [CLS03] as summarized in Section 4 to reason about solutions of symbolic constraint sequences. Our main insight is that, assuming the symbolic constraint sequence $\mathbf{C}$ is generated from a well-defined protocol and has a solution, there exists a *conservative* solution that uses only the structure already present in $\mathbf{C}$. Even though variables may need to be instantiated, in the conservative solution all instantiations are products of subterms (and their inverses) that are already present in the original sequence $\mathbf{C}$.

To illustrate by example, consider the following constraint sequence, where x and y are variables:

$$\begin{array}{l}\mathbf{1}, a \rhd x;\\ \mathbf{1}, a, \langle x, b\rangle \rhd \langle y, b\rangle;\\ \mathbf{1}, a, \langle x, b\rangle, \{x \cdot y\}_k \rhd \{a\}_k\end{array}$$

One solution of this sequence is the substitution $\sigma = [x \mapsto a \cdot \langle a, a\rangle, y \mapsto \langle a, a\rangle^{-1}]$. This solution, however, is not conservative, since the $\langle a, a\rangle$ term is not among the subterms of the original (uninstantiated) constraint sequence. We demonstrate that if the constraint sequence is solvable, then it also has a conservative solution — in this case, $\sigma^* = [x \mapsto a, y \mapsto \mathbf{1}]$. We can then reduce the protocol analysis problem to the search for conservative solutions of the corresponding constraint sequence.

Lemma 5.1 $\mathrm{St}(T\sigma) \subseteq \mathrm{St}(T)\sigma \cup \bigcup_{x \in \mathrm{Var}(T)} \mathrm{St}(x\sigma)$.

Proof: It is sufficient to show that, for all $t \in T\sigma, \mathrm{St}(t\sigma) \subseteq \mathrm{St}(t)\sigma \cup \bigcup_{x \in \mathrm{Var}(t)} \mathrm{St}(x\sigma)$. The proof is by induction on the depth of t's structure. For the induction basis, consider the case when the depth of t is 0, *i.e.*, $t = a$ for some constant a, in which case $\mathrm{St}(t\sigma) = \mathrm{St}(a) = a \in \mathrm{St}(t)\sigma$, or $t = x$ for some variable x, in which case $\mathrm{St}(t\sigma) = \mathrm{St}(x\sigma)$. Now suppose $\mathrm{St}(t\sigma) \subseteq \mathrm{St}(t)\sigma \cup \bigcup_{x \in \mathrm{Var}(t)} \mathrm{St}(x\sigma)$ for any t of depth less than or equal to i, and consider t of depth $i + 1$. If $t = \langle u, v\rangle$, then $\mathrm{St}(\langle u, v\rangle\sigma) = \langle u\sigma, v\sigma\rangle \cup \mathrm{St}(u\sigma) \cup \mathrm{St}(v\sigma)$. Observe that $\langle u\sigma, v\sigma\rangle = \langle u, v\rangle\sigma \in \mathrm{St}(t)\sigma$, and, since the induction hypothesis holds for u and v, $\mathrm{St}(u\sigma) \subseteq \mathrm{St}(u)\sigma \bigcup_{x \in \mathrm{Var}(u)} \mathrm{St}(x\sigma), \mathrm{St}(v\sigma) \subseteq \mathrm{St}(v)\sigma \bigcup_{x \in \mathrm{Var}(v)} \mathrm{St}(x\sigma)$. Because $\mathrm{St}(u), \mathrm{St}(v) \subseteq \mathrm{St}(\langle u, v\rangle)$ and $\mathrm{Var}(u) \cup \mathrm{Var}(v) = \mathrm{Var}(\langle u, v\rangle)$, we obtain that $\mathrm{St}(\langle u, v\rangle\sigma) = \mathrm{St}(\langle u, v\rangle)\sigma \cup \bigcup_{x \in \mathrm{Var}(\langle u, v\rangle)} \mathrm{St}(x\sigma)$. The proofs for $t = \{u\}_v, f(u), u^{-1}$, and $u_1 \cdot \ldots \cdot u_n$ are identical. □

Lemma 5.2 (Instantiation doesn't introduce structure) *Let* $\mathbf{C}$ *be a constraint sequence generated from a protocol, and let* T_i *be some term set such that* $T_i \rhd u_i \in \mathbf{C}$*. If, for some term* u*, there is a minimal size normal proof* $\mathcal{P}$ *of one of the following forms:*

$$\dfrac{\dfrac{\vdots}{T_i\sigma \vdash \langle u, v\rangle}}{T_i\sigma \vdash u} \qquad \dfrac{\dfrac{\vdots}{T_i\sigma \vdash \langle v, u\rangle}}{T_i\sigma \vdash u} \qquad \dfrac{\dfrac{\vdots}{T_i\sigma \vdash \{u\}_v} \quad \dfrac{\vdots}{T_i\sigma \vdash v}}{T_i\sigma \vdash u}$$

Then $\langle u, v\rangle \in \mathrm{St}(T_i)\sigma$ *(resp.* $\langle v, u\rangle \in \mathrm{St}(T_i)\sigma$*, resp.* $\{u\}_v \in \mathrm{St}(T_i)\sigma$*).*

Proof: We prove the lemma for $\langle u, v\rangle$. The proofs for $\langle v, u\rangle$ and $\{u\}_v$ are similar.

By Lemma 4.2 and Lemma 5.1, $\langle u, v\rangle \in \mathrm{St}(T_i\sigma) \subseteq \mathrm{St}(T_i)\sigma \cup \bigcup_{x \in \mathrm{Var}(T_i)} \mathrm{St}(x\sigma)$. If $\mathrm{Var}(T_i)$ is empty, the lemma follows trivially. Suppose $\mathrm{Var}(T_i)$ is not empty. Since $\mathbf{C}$ is generated from a well-defined protocol, $\mathbf{C}$ satisfies the origination stability property by Theorem 3.8. Construct a linear ordering $\prec$ on $\mathrm{Var}(\mathbf{C})$ consistent with the order of first occurrence, so that $k_x < k_y$ implies $x \prec y$. Arrange variables $x_1, \ldots, x_n \in \mathrm{Var}(T_i)$ so that $x_1 \prec \ldots \prec x_n$.

We prove the lemma by induction over the size of $\mathrm{Var}(T_i)$. More precisely, we will show, if $x_k \in \mathrm{Var}(T_i)$, then if $\langle u, v\rangle \in \mathrm{St}(x_k\sigma)$, then $\langle u, v\rangle \in \mathrm{St}(T_i)\sigma \cup \mathrm{St}(x_1\sigma) \cup \ldots \cup \mathrm{St}(x_{k-1}\sigma)$.

By the origination property (Definition 3.3), there exists $j < i$ such that $x_k \in \mathrm{Var}(u_j)$ and $y \in \mathrm{Var}(T_j)$ implies $y \prec x_k$. Observe that $\mathrm{St}(T_j\sigma) = \mathrm{St}(T_j)\sigma \cup \bigcup_{y \in \mathrm{Var}(T_j)} \mathrm{St}(y\sigma)$. By the monotonicity property (Definition 3.1), $T_j \subseteq T_i$ (in fact, since $x_k \in \mathrm{Var}(T_i)$ and $x_k \notin \mathrm{Var}(T_j)$, $T_j \subset T_i$). Therefore, $\mathrm{St}(T_j)\sigma \subseteq \mathrm{St}(T_i)\sigma$ and $\mathrm{Var}(T_j) \subset \mathrm{Var}(T_i)$. Since $y \in \mathrm{Var}(T_j)$ implies $y \prec x_k$, it must be that $y = x_l \in \mathrm{Var}(T_i)$ where $l < k$. Therefore, for all $y \in \mathrm{Var}(T_j), \mathrm{St}(y\sigma) \subseteq \mathrm{St}(x_1\sigma) \cup \ldots \cup \mathrm{St}(x_{k-1}\sigma)$. We conclude that $\mathrm{St}(T_j\sigma) \subseteq \mathrm{St}(T_i)\sigma \cup \mathrm{St}(x_1\sigma) \cup \ldots \cup \mathrm{St}(x_{k-1}\sigma)$. To complete the induction step, it remains to show that $\langle u, v\rangle \in \mathrm{St}(T_j)\sigma$.

Since $x_k \in \mathrm{St}(u_j)$ (by the origination property) and $\langle u, v\rangle \in \mathrm{St}(x_k\sigma)$ (by assumption), $\langle u, v\rangle \in \mathrm{St}(u_j\sigma)$. We apply Lemma 4.6 to the proof of $T_j\sigma \vdash u_j\sigma$. If $\langle u, v\rangle \in \mathrm{St}(T_j\sigma)$, we are done.

Now suppose $\langle u, v\rangle \notin \mathrm{St}(T_j\sigma)$. By Lemma 4.6, there must exist a proof of $T_j\sigma \vdash u$ such that no node is labeled with $T \vdash \langle u, v\rangle$. Since $T_j\sigma \subset T_i\sigma$, this proof can also serve as the proof of $T_i\sigma \vdash u$. But this contradicts the assumption that the minimal size normal proof of $T_i\sigma \vdash u$ relies on $T_i\sigma \vdash \langle u, v\rangle$. We conclude that $\langle u, v\rangle \in \mathrm{St}(T_j\sigma)$. □

Lemma 5.3 *If $t = \langle u, v\rangle \in \bar{\mathcal{S}}(\mathbf{C})\sigma$ (resp. $\{u\}_v \in \bar{\mathcal{S}}(\mathbf{C})\sigma$, $f(u) \in \bar{\mathcal{S}}(\mathbf{C})\sigma$), then there exists $t' = \langle u', v'\rangle \in \mathrm{St}(\mathbf{C})$ (resp. $\{u'\}_{v'} \in \mathrm{St}(\mathbf{C})$, $f(u') \in \mathrm{St}(\mathbf{C})$) such that $t'\sigma = t$.*

Proof: Observe that $\mathrm{St}(u_1 \cdot \ldots \cdot u_n) = \{u_1 \cdot \ldots \cdot u_n\} \cup \bigcup_i \mathrm{St}(u_i)$, and $\mathrm{St}(u^{-1}) = \mathrm{St}(u)$. Therefore, it follows immediately from the definition $\bar{\mathcal{S}}(\mathbf{C})\sigma = \mathrm{PC}(\mathrm{St}(\mathbf{C}) \setminus \mathrm{Var}(\mathbf{C}))\sigma$ that $\langle u, v\rangle \in \bar{\mathcal{S}}(\mathbf{C})\sigma$ if and only if $\langle u, v\rangle \in \mathrm{St}(\mathbf{C})\sigma$. This implies that there exists $\langle u', v'\rangle \in \mathrm{St}(\mathbf{C})$ such that $\langle u', v'\rangle\sigma = \langle u, v\rangle$. The proofs for $\{u\}_v$ and $f(u)$ are identical. □

Lemma 5.4 *Suppose $\mathcal{P}$ is a proof by Definition 4.1, and let* $\dfrac{T \vdash v_1 \quad [T \vdash v_2]}{T \vdash v_3}$ *be an inference in $\mathcal{P}$ which is an instance of some rule other than (M) or (I) (the $T \vdash v_2$ premise may be absent). For $i \in \{1, 2, 3\}$, if $v_i \in \bar{\mathcal{S}}(\mathbf{C})\sigma$ for some σ, then $v_i \in \mathrm{St}(\mathbf{C})\sigma$.*

Proof: If the rule is (P), then $v_3 = \langle v_1, v_2\rangle$. By assumption, $v_3 \in \bar{\mathcal{S}}(\mathbf{C})\sigma$. By Lemma 5.3, there exists $\langle v_1', v_2'\rangle \in \mathrm{St}(\mathbf{C})$ such that $v_i = v_i'\sigma$ for $i \in \{1, 2\}$. Therefore, $v_3 \in \mathrm{St}(\mathbf{C})\sigma$ and, since $v_i' \in \mathrm{St}(\mathbf{C})$ by definition for $i \in \{1, 2\}$, $v_i \in \mathrm{St}(\mathbf{C})\sigma$. If the rule is (UL), then $v_1 = \langle v_3, v'\rangle$ for some term v'. By assumption, $v_1 \in \bar{\mathcal{S}}(\mathbf{C})\sigma$. By Lemma 5.3, this implies that $v_1 \in \mathrm{St}(\mathbf{C})\sigma$, and, by the same reasoning as above, $v_3 \in \mathrm{St}(\mathbf{C})\sigma$. The proofs for (UR),(D),(E),(F) are similar. □

Definition 5.5 (Conservative substitution) *Substitution σ is* conservative *if*

$$(\forall x \in \mathrm{Var}(\mathbf{C})) \quad \mathrm{St}(x\sigma) \subseteq \bar{\mathcal{S}}(\mathbf{C})\sigma$$

Intuitively, a conservative substitution does not introduce any structure that was not already present in the original symbolic sequence $\mathbf{C}$. We will prove that if there exists some solution $\sigma \Vdash \mathbf{C}$, then there exists a conservative solution $\sigma^* \Vdash \mathbf{C}$.

Define transformation $\nu_{\mathbf{C}}$ on normalized ground terms as follows:

$$\begin{array}{lcl}
\nu_{\mathbf{C}}(c) & = & \begin{cases} c & \text{if } c \in \bar{\mathcal{S}}(\mathbf{C})\sigma \\ \mathbf{1} & \text{otherwise} \end{cases} \\
\nu_{\mathbf{C}}(\langle u, v\rangle) & = & \begin{cases} \langle \nu_{\mathbf{C}}(u), \nu_{\mathbf{C}}(v)\rangle & \text{if } \langle u, v\rangle \in \bar{\mathcal{S}}(\mathbf{C})\sigma, \\ \mathbf{1} & \text{otherwise} \end{cases} \\
\nu_{\mathbf{C}}(\{u\}_v) & = & \begin{cases} \{\nu_{\mathbf{C}}(u)\}_{\nu_{\mathbf{C}}(v)} & \text{if } \{u\}_v \in \bar{\mathcal{S}}(\mathbf{C})\sigma, \\ \mathbf{1} & \text{otherwise} \end{cases} \\
\nu_{\mathbf{C}}(f(u)) & = & \begin{cases} f(\nu_{\mathbf{C}}(u)) & \text{if } f(u) \in \bar{\mathcal{S}}(\mathbf{C})\sigma, \\ \mathbf{1} & \text{otherwise} \end{cases} \\
\nu_{\mathbf{C}}(u_1 \cdot \ldots \cdot u_k) & = & \nu_{\mathbf{C}}(u_1) \cdot \ldots \cdot \nu_{\mathbf{C}}(u_k) \quad (k > 1) \\
\nu_{\mathbf{C}}(u^{-1}) & = & \nu_{\mathbf{C}}(u)^{-1}
\end{array}$$

Then define substitution σ^* as $x\sigma^* = \nu_{\mathbf{C}}(x\sigma)$. Essentially, $x\sigma^*$ is the same as $x\sigma$, except that all subterms of $x\sigma$ that are not in $\bar{\mathcal{S}}(\mathbf{C})\sigma$ have been eliminated by the transformation $\nu_{\mathbf{C}}$.

In our choice of $\nu_{\mathbf{C}}$, we use $\mathbf{1}$ as the replacement for any subterm whose structure was not already present in the original constraint sequence. This choice is arbitrary. In fact, *any* value computable by the attacker would work just as well. The essence of our argument in the rest of this section is that if there exists some attack (*i.e.*, some instantiation of variables that makes the symbolic trace feasible), then there is an equivalent attack which can be constructed without using these terms at all. Therefore, it does not matter what these subterms are replaced with, as long as the replacement value can be feasibly computed by the attacker.

Lemma 5.6 *If $v \in \mathrm{Var}(\mathbf{C})$ or $v \in \bar{\mathcal{S}}(\mathbf{C})$, then $v\sigma^* = \nu_{\mathbf{C}}(v\sigma)$.*

Proof: If $v \in \mathrm{Var}(\mathbf{C})$, then $v\sigma^* = \nu_{\mathbf{C}}(v\sigma)$ by definition of σ^*. Otherwise, by induction on the structure of v. If $v = c \in \bar{\mathcal{S}}(\mathbf{C})$, c is a constant, then $\nu_{\mathbf{C}}(v\sigma) = \nu_{\mathbf{C}}(c\sigma) = \nu_{\mathbf{C}}(c) = c = c\sigma^* = v\sigma^*$. For the induction hypothesis, assume that the lemma is true for $v_1, \ldots, v_k$. For the induction step, we need to show that it holds for $\langle v_1, v_2\rangle$, $\{v_1\}_{v_2}$, $f(u)$, ${v_1}^{-1}$ and $v_1 \cdot \ldots \cdot v_k$.

Consider $v = \langle v_1, v_2\rangle \in \bar{\mathcal{S}}(\mathbf{C})$. This implies that $v\sigma \in \bar{\mathcal{S}}(\mathbf{C})\sigma$. Observe that $v\sigma = \langle v_1\sigma, v_2\sigma\rangle \in \bar{\mathcal{S}}(\mathbf{C})\sigma$, and $v\sigma^* = \langle v_1\sigma^*, v_2\sigma^*\rangle$. By the induction hypothesis, $\nu_{\mathbf{C}}(v_i\sigma) = v_i\sigma^*$. Given that $v\sigma \in \bar{\mathcal{S}}(\mathbf{C})\sigma$, by definition of $\nu_{\mathbf{C}}$, $\nu_{\mathbf{C}}(v\sigma) = \langle \nu_{\mathbf{C}}(v_1\sigma), \nu_{\mathbf{C}}(v_2\sigma)\rangle = \langle v_1\sigma^*, v_2\sigma^*\rangle = v\sigma^*$.

The proofs for $\{v_1\}_{v_2}$, $f(u)$, ${v_1}^{-1}$ and $v_1 \cdot \ldots \cdot v_k$ are similar. □

Lemma 5.7 *σ^* is a conservative substitution.*

Proof: Consider any $x \in \mathrm{Var}(\mathbf{C})$. By definition, $x\sigma^* = \nu_{\mathbf{C}}(x\sigma)$. We prove that $\mathrm{St}(\nu_{\mathbf{C}}(x\sigma)) \subseteq \bar{\mathcal{S}}(\mathbf{C})\sigma$ by induction on the structure of $\nu_{\mathbf{C}}(x\sigma)$.

If $\nu_{\mathbf{C}}(x\sigma) = c$ where c is a constant, then, by definition of $\nu_{\mathbf{C}}$, it must be that $c \in \bar{\mathcal{S}}(\mathbf{C})\sigma$. For the induction hypothesis, suppose $\nu_{\mathbf{C}}(x\sigma) = \langle t_1, t_2\rangle$ or $\{t_1\}_{t_2}$ or $f(t_1)$ or $t_1 \cdot \ldots \cdot t_k$ or ${t_1}^{-1}$, and $t_1, \ldots, t_k \in \bar{\mathcal{S}}(\mathbf{C})\sigma$.

First, consider the case when $\nu_{\mathbf{C}}(x\sigma) = \langle t_1, t_2\rangle$ (the $\{t_1\}_{t_2}$ and $f(t_1)$ cases are similar). If $\langle t_1, t_2\rangle \notin \bar{\mathcal{S}}(\mathbf{C})\sigma$, then, by definition of $\nu_{\mathbf{C}}(x\sigma)$, it must be the case that $\langle t_1, t_2\rangle = \nu_{\mathbf{C}}(x\sigma) = \mathbf{1}$, and we obtain a contradiction. Therefore, $\langle t_1, t_2\rangle \in \bar{\mathcal{S}}(\mathbf{C})\sigma$.

Now consider the case when $\nu_{\mathbf{C}}(x\sigma) = t_1 \cdot \ldots \cdot t_k$ or $\nu_{\mathbf{C}}(x\sigma) = {t_1}^{-1}$ and, by the induction hypothesis, $t_i \in \bar{\mathcal{S}}(\mathbf{C})\sigma$ for all i. Since $\bar{\mathcal{S}}(\mathbf{C})\sigma$ is closed under $\cdot$ and inverse, $\nu_{\mathbf{C}}(x\sigma) \in \bar{\mathcal{S}}(\mathbf{C})\sigma$. □

Lemma 5.8 *Given $T \rhd u \in \mathbf{C}$ and some ground term t, if there exists a proof of $T\sigma \vdash t$, then there exists a proof of $T\sigma^* \vdash \nu_{\mathbf{C}}(t)$.*

Proof: Let $\mathcal{P}$ be the normal proof of $T\sigma \vdash t$ using the inference rules of Fig. 3. Such a proof exists by Lemma 4.5. We prove the lemma by induction over the structure of $\mathcal{P}$.

For the induction basis, suppose that $\mathcal{P}$ consists of a single leaf node $T\sigma \vdash t$ such that $t \in T\sigma$. This implies that there exists $v \in T$ such that $t = v\sigma$. Either $v \in \mathrm{Var}(\mathbf{C})$, or $v \in T \setminus \mathrm{Var}(\mathbf{C}) \subseteq \bar{\mathcal{S}}(\mathbf{C})$. In either case, by Lemma 5.6, $v\sigma^* = \nu_{\mathbf{C}}(t)$. Therefore, $\nu_{\mathbf{C}}(t) \in T\sigma^*$, and the proof of $T\sigma^* \vdash \nu_{\mathbf{C}}(t)$ consists of a single node $T\sigma^* \vdash \nu_{\mathbf{C}}(t)$.

Consider one inference of $\mathcal{P}$ of the form $\dfrac{T\sigma \vdash t_1 \quad [\ldots\ T\sigma \vdash t_k]}{T\sigma \vdash t}$ (the $T\sigma \vdash t_i$ premises for $i > 1$ may be absent) and assume as the induction hypothesis that for all i there exist proofs $\mathcal{P}_i$ of $T\sigma^* \vdash \nu_{\mathbf{C}}(t_i)$. To complete the induction, it is sufficient to show that for any inference rule, there exists a proof of $T\sigma^* \vdash \nu_{\mathbf{C}}(t)$.

If the rule is (D), then $t_1 = \{t\}_{t_k}$, $t_2 = t_k$ for some term t_k. By Lemma 5.2, $\{t\}_{t_k} \in \mathrm{St}(T)\sigma$, *i.e.*, there exists $\{v_1\}_{v_2} \in T$ such that $v_1\sigma = t, v_2\sigma = t_k$. Since $\{v_1\}_{v_2} \notin \mathrm{Var}(T)$, we obtain that $\{v_1\}_{v_2} \in \mathcal{S}(T) \subseteq \bar{\mathcal{S}}(\mathbf{C})$. Therefore, $\{t\}_{t_k} \in \bar{\mathcal{S}}(\mathbf{C})\sigma$, thus $\nu_{\mathbf{C}}(\{t\}_{t_k}) = \{\nu_{\mathbf{C}}(t)\}_{\nu_{\mathbf{C}}(t_k)}$. By the induction hypothesis, $\mathcal{P}_1$ is the proof of $T\sigma^* \vdash \nu_{\mathbf{C}}(\{t\}_{t_k})$ and $\mathcal{P}_2$ is the proof of $T\sigma^* \vdash \nu_{\mathbf{C}}(t_k)$. The proof of $T\sigma^* \vdash \nu_{\mathbf{C}}(t)$ is then constructed as follows:

$$\dfrac{\begin{array}{c}\mathcal{P}_1\\ T\sigma^* \vdash \{\nu_{\mathbf{C}}(t)\}_{\nu_{\mathbf{C}}(t_k)}\end{array} \qquad \begin{array}{c}\mathcal{P}_2\\ T\sigma^* \vdash \nu_{\mathbf{C}}(t_k)\end{array}}{T\sigma^* \vdash \nu_{\mathbf{C}}(t)}$$

A similar argument applies for rules (UL),(UR).

If the rule is (P), then $t = \langle t_1, t_2\rangle$. According to the definition of $\nu_{\mathbf{C}}$, there are two possibilities. If $\nu_{\mathbf{C}}(\langle t_1, t_2\rangle) = \mathbf{1}$, then $\nu_{\mathbf{C}}(\langle t_1, t_2\rangle) \in T\sigma^*$, and the proof of $T\sigma^* \vdash \nu_{\mathbf{C}}(\langle t_1, t_2\rangle)$ consists of one node $T\sigma^* \vdash \mathbf{1}$. If $\nu_{\mathbf{C}}(\langle t_1, t_2\rangle) = \langle \nu_{\mathbf{C}}(t_1), \nu_{\mathbf{C}}(t_2)\rangle$, the proof of $T\sigma^* \vdash \nu_{\mathbf{C}}(\langle t_1, t_2\rangle)$ is constructed as follows:

$$\dfrac{\begin{array}{c}\mathcal{P}_1\\ T\sigma^* \vdash \nu_{\mathbf{C}}(t_1)\end{array} \qquad \begin{array}{c}\mathcal{P}_2\\ T\sigma^* \vdash \nu_{\mathbf{C}}(t_2)\end{array}}{T\sigma^* \vdash \nu_{\mathbf{C}}(\langle t_1, t_2\rangle)}$$

A similar argument applies for rules (E),(F),(M),(I). □

Theorem 5.9 (Existence of conservative solution) *If there exists a solution $\sigma \Vdash \mathbf{C}$, then there exists a conservative solution $\sigma^* \Vdash \mathbf{C}$.*

Proof: Let $\sigma \Vdash \mathbf{C}$ be a solution of $\mathbf{C}$. Define substitution σ^* with domain $\mathrm{Var}(\mathbf{C})$ by $x\sigma^* = \nu_{\mathbf{C}}(x\sigma)$. By Lemma 5.7, σ^* is a conservative substitution. To show that $\sigma^* \Vdash \mathbf{C}$, consider any constraint $T \rhd u \in \mathbf{C}$. Since $\sigma \Vdash \mathbf{C}$, there exists a proof of $T\sigma \vdash u\sigma$. Either $u \in \mathrm{Var}(\mathbf{C})$, or $u \in \mathrm{St}(\mathbf{C}) \setminus \mathrm{Var}(\mathbf{C}) \subseteq \bar{\mathcal{S}}(\mathbf{C})$. In either case, by Lemma 5.6, $u\sigma^* = \nu_{\mathbf{C}}(u\sigma)$. By Lemma 5.8, there exists a proof of $T\sigma^* \vdash \nu_{\mathbf{C}}(u\sigma) = u\sigma^*$. Therefore, $\sigma^* \Vdash \mathbf{C}$. □

Lemma 5.10 *If σ is a conservative substitution, then* $\mathrm{St}(\mathbf{C})\sigma \subseteq \bar{\mathcal{S}}(\mathbf{C})\sigma$.

Proof: First, observe that $\mathrm{St}(\mathbf{C})\sigma \subseteq \mathcal{S}(\mathbf{C})\sigma \cup \bigcup_{x\in \mathrm{Var}(\mathbf{C})} \mathrm{St}(x\sigma)$. By definition of $\mathcal{S}(\mathbf{C})$, $\mathcal{S}(\mathbf{C})\sigma \subseteq \bar{\mathcal{S}}(\mathbf{C})\sigma$. By Definition 5.5, $(\forall x \in \mathrm{Var}(\mathbf{C}))\mathrm{St}(x\sigma) \subseteq \bar{\mathcal{S}}(\mathbf{C})\sigma$. Therefore, $\mathrm{St}(\mathbf{C})\sigma \subseteq \bar{\mathcal{S}}(\mathbf{C})\sigma$. □

Definition 5.11 (Conservative proof) *Given $T \rhd u \in \mathbf{C}$ and a substitution σ, a proof $\mathcal{P}$ of $T\sigma \vdash u\sigma$ is* conservative *if, for every node of $\mathcal{P}$ labeled $T\sigma \vdash v$,*

- *either $v \in \mathrm{St}(\mathbf{C})\sigma$, or*
- *node $T\sigma \vdash v$ is obtained by (M) or (I) inference rule* and *is only used as a premise of an (M) or (I) rule.*

Lemma 5.12 (Existence of conservative proof) *If $\sigma^* \Vdash \mathbf{C} = \{T_i \rhd u_i\}$ is a conservative solution, then there exists a conservative proof of $T_i\sigma^* \vdash u_i\sigma^*$ for each i.*

Proof: Let σ^* be the conservative solution of $\mathbf{C}$. Consider any constraint $T_i \rhd u_i$. Since $\sigma^* \Vdash \mathbf{C}$, by Lemma 4.5 there exists a normal proof $\mathcal{P}_i$ of $T_i\sigma^* \vdash u_i\sigma^*$. Let $\hat{T}_i = T_i \cup \{u_i\}$, and let $V_i = \mathrm{Var}(\hat{T}_i) \subseteq \mathrm{Var}(\mathbf{C})$. ¿From Definition 4.4 of normal proofs, it follows that every node of $\mathcal{P}_i$ is labeled $T_i\sigma^* \vdash v$ where $v \in \mathrm{St}(\hat{T}_i\sigma^*)$. Since $\hat{T}_i \subseteq \mathrm{St}(\mathbf{C})$, by Lemma 5.10 $v \in \bar{\mathcal{S}}(\mathbf{C})\sigma^*$.

Any inference in $\mathcal{P}_i$ other than (M) or (I) must have the form $\dfrac{T \vdash v_1 \quad [T \vdash v_2]}{T \vdash v_3}$.

Since $v_{1,2,3} \in \bar{\mathcal{S}}(\mathbf{C})\sigma^*$, by Lemma 5.4 $v_{1,2,3} \in \mathrm{St}(\mathbf{C})\sigma^*$. □

6 Decision Procedure for Symbolic Constraints

In this section, we present a decision procedure for symbolic constraint sequences associated with well-defined protocols. The essence of our decidability result is the proof that for each symbolic constraint sequence $\mathbf{C}$, there exists a *finite* number of systems of simultaneous Diophantine equations such that (i) each system is decidable, and (ii) $\mathbf{C}$ has a solution if and only if at least one of the systems has a solution in integers. We emphasize that our goal is a theoretical decidability result. Therefore, we are concerned only with showing finiteness of our procedure and decidability of a particular class of Diophantine equations. In future work, we plan to investigate an efficient constraint solving procedure based on [MS01] that can be applied to practical protocol analysis.

Our decision procedure starts with two *finite, nondeterministic* steps $\leadsto_1$ and $\leadsto_2$, followed by two deterministic steps $\leadsto_3$ and $\leadsto_4$. Let $\mathbf{C}_0$ be an initial constraint sequence generated from the protocol specification. For each step $\leadsto_i$, we show that (i) there are finitely many $\mathbf{C}_i$ such that $\mathbf{C}_{i-1} \leadsto_i \mathbf{C}_i$, and (ii) $\mathbf{C}_{i-1}$ has a solution *if and only if* at least one $\mathbf{C}_i$ has a solution. This guarantees soundness and completeness. *Soundness* means that if any member of the set of sequences obtained by a particular step has a solution, then the original sequence has a solution. *Completeness* means that if the original sequence has a (conservative) solution, then at least one of the sequences obtained at each step has the same solution. Performing the steps does not require *a priori* knowledge of the solution. If a solution exists, it will be discovered by exhaustively enumerating all possible sequences produced by our procedure and checking whether each one has a solution.

For each of the constraint sequences $\mathbf{C}_4$ produced by the last step, we show that $\mathbf{C}_4$ has a solution if and only if a special system of quadratic Diophantine equations has a solution. Quadratic Diophantine equations are undecidable in general, but the system obtained in our case is solvable if and only if a particular linear subsystem is solvable. Since linear Diophantine equations are decidable, this establishes that the symbolic protocol analysis problem is decidable in the presence of an Abelian group operator.

Following Theorem 5.9, we limit our attention to conservative solutions. Our decision procedure consists in the following steps:

1. Guess subterm equalities.

2. For each constraint, guess all subterms derivable from the set of terms available to the attacker, and add them to this set.

3. Remove all constraints in which the derivation involves inference rules other than (M) or (I).

4. Substitute all target terms that introduce new variables.

5. For each of the resulting sequences, solve a system of linear Diophantine equations to determine whether the sequence has a solution or not.

Running example. We will use the following symbolic trace as an (artificial) running example to illustrate our decision procedure. An event $A \longrightarrow t$ is a $+t$ node in an A-role strand, *etc.*

1.	A	$\longrightarrow$	$a \cdot b$	4.	B	$\longleftarrow$	$\{Y\}_b$
2.	B	$\longleftarrow$	$a \cdot X \cdot Y$	5.	B	$\longrightarrow$	$b \cdot X$
3.	A	$\longrightarrow$	$\{a\}_b$	6.	A	$\longleftarrow$	a^6

Recall that the goal of symbolic protocol analysis is to determine whether this trace is *feasible*, *i.e.*, whether there exists an instantiation of variables X and Y such that every term sent from the network and received by an honest participant (*i.e.*, every term of the form $P \longleftarrow$) is derivable using the rules of Fig. 3. This is equivalent to deciding whether the corresponding symbolic constraint sequence $\mathbf{C}$ has a solution:

$$\begin{array}{l} a \cdot b \rhd a \cdot X \cdot Y \ ; \\ a \cdot b, \{a\}_b \rhd \{Y\}_b \ ; \\ a \cdot b, \{a\}_b, b \cdot X \rhd a^6 \end{array}$$

6.1 Determine subterm equalities

Suppose $\mathbf{C}$ has some solution σ. In the first step $\leadsto_1$, we guess the equivalence relation on $\mathrm{St}(\mathbf{C})$ induced by substitution σ. As we argue below, there are only finitely many possibilities to consider, and one of them is the right one. Of course, we don't know σ beforehand. Therefore, to discover which equivalence relation is the right one, we will need to enumerate all possible relations and perform the remaining steps of the decision procedure for each one to determine whether it leads to a solvable sequence. If $\mathbf{C}$ does *not* have a solution, it will not matter which equivalence relation we choose, since none of them will lead to a solvable sequence.

More precisely, for all $s_i, s_j \in \mathrm{St}(\mathbf{C})$, we guess whether $s_i\sigma = s_j\sigma$ or not. Since $\mathrm{St}(\mathbf{C})$ is finite, there are only a finite number of possible equivalence relations to consider. Each equivalence relation represents a set of unification problems modulo associativity and commutativity of $\cdot$ and normalization rules of Fig. 2 (see Section 2.3). There are finitely many most general unifiers consistent with any given equivalence relation. Let Θ be the finite set of candidate unifiers. For each $\theta \in \Theta$, let $\mathbf{C}_1 = \mathbf{C}\theta$.

Observe that for *any* substitution σ, there exists a partial substitution $\theta \in \Theta$ such that $s_i\theta = s_j\theta$ if and only if $s_i\sigma = s_j\sigma$. Therefore, any $\sigma = \theta \circ \theta'$ for some $\theta \in \Theta$. (*Notation:* Composition of substitutions is left to right, so $t(\theta_1 \circ \theta_2) = (t\theta_1)\theta_2 = t\theta_1\theta_2$.)

Lemma 6.1 *(Soundness:) For any $\mathbf{C}_1$ such that $\mathbf{C} \leadsto_1 \mathbf{C}_1$, if there exists a solution $\sigma_1 \Vdash \mathbf{C}_1$, then there exists a solution $\sigma \Vdash \mathbf{C}$.*

(Completeness:) If there exists a solution $\sigma \Vdash \mathbf{C}$, then $\exists \mathbf{C}_1$ such that $\mathbf{C} \leadsto_1 \mathbf{C}_1$ and $\sigma \Vdash \mathbf{C}_1$.

Proof: To prove soundness, suppose some $\sigma_1 \Vdash \mathbf{C}_1$ where $\mathbf{C}_1 = \mathbf{C}\theta$ for some $\theta \in \Theta$. This means that $\forall T_i \rhd u_i \in \mathbf{C}$, $T_i\theta\sigma_1 \vdash u_i\theta\sigma_1$ is derivable. Choose as σ any substitution of the form $\theta \circ \theta'$ such that $\forall s \in \mathrm{St}(\mathbf{C})\ s\sigma = s\sigma_1$. Observe that $\forall T_i \rhd u_i \in \mathbf{C}\ \ T_i\sigma \vdash u_i\sigma$ is derivable. Therefore, $\sigma \Vdash \mathbf{C}$.

To prove completeness, observe that, by our definition of $\leadsto_1$, for *any* substitution σ there exists a unifier $\theta \in \Theta$ such that $\sigma = \theta \circ \theta'$. Therefore, for any σ such that $\sigma \Vdash \mathbf{C}$, there exists $\mathbf{C}_1 = \mathbf{C}\theta$ such that $\sigma \Vdash \mathbf{C}_1$. □

Lemma 6.2 *Suppose $\sigma \Vdash \mathbf{C}$. Consider $\mathbf{C}_1$ such that $\mathbf{C} \leadsto_1 \mathbf{C}_1$ and $\sigma \Vdash \mathbf{C}_1$, and any $s, s' \in \mathrm{St}(\mathbf{C}_1)$. If $s \neq s'$, then $s\sigma \neq s'\sigma$.*

Proof: By construction of $\mathbf{C}_1$, for all $s, s' \in \mathrm{St}(\mathbf{C}_1)$, there exist $\hat{s}, \hat{s}' \in \mathrm{St}(\mathbf{C})$ such that $\hat{s}\theta = s, \hat{s}'\theta = s'$. Since $\sigma = \theta \circ \theta'$, $\hat{s}\sigma = s\sigma$ and $\hat{s}'\sigma = s'$. By choice of θ, if $\hat{s}\sigma = \hat{s}'\sigma$, then $\hat{s}\theta = \hat{s}'\theta$. Therefore, if $s\sigma = s'\sigma$, then $s = s'$, or, reversing the order of implication, if $s \neq s'$, then $s\sigma \neq s'\sigma$. □

Running example. In our running example, we guess that the only subterm equality is $\{Y\}_b = \{a\}_b$, giving us partial substitution $[Y \mapsto a]$ and producing the following $\mathbf{C}_1$:

$$\begin{array}{l} a \cdot b \rhd a^2 \cdot X \ ; \\ a \cdot b, \{a\}_b \rhd \{a\}_b \ ; \\ a \cdot b, \{a\}_b, b \cdot X \rhd a^6 \end{array}$$

6.2 Determine order of subterm derivation

In the second step $\rightsquigarrow_2$, we nondeterministically choose one of the candidate sequences $\mathbf{C}_1$ produced by $\rightsquigarrow_1$. Assuming $\mathbf{C}_1$ has a solution σ, we (1) guess which subterms of $\mathbf{C}_1\sigma$ can be derived by the attacker using inference rules of Fig. 3, and (2) add each derivable subterm s to every constraint $T_i \rhd u_i \in \mathbf{C}_1$ such that $s\sigma$ is derivable from $T_i\sigma$.

In the resulting constraint sequence, every constraint is solved either by application of a *single* inference rule, or the derivation involves only rules (M) and (I). In the former case, we can discover the right rule by syntactic inspection. In the latter case, only multiplicative operations are used, and we will convert the constraint solving problem into a system of simultaneous Diophantine equations.

Since we don't know σ in advance, we need to exhaustively try all possible combinations of subterms and constraints. If the chosen $\mathbf{C}_1$ has a solution, one of the candidate combinations will be the right one. If the chosen $\mathbf{C}_1$ does *not* have a solution, all candidate sequences will be unsolvable.

Formally, the $\rightsquigarrow_2$ step consists in the following sub-steps:

1. Guess $S_\vdash = \{s \in \mathrm{St}(\mathbf{C}_1) \mid \exists T_i \rhd u_i \in \mathbf{C}_1$ such that there exists a proof of $T_i\sigma \vdash s\sigma\}$, *i.e.*, $S_\vdash$ is the set of subterms that are derivable from *some* term set available to the attacker. This terminates, since there are only finitely many subsets of $\mathrm{St}(\mathbf{C}_1)$ to consider.

2. For all $s \in S_\vdash$ guess $j_s \in \{1, ..., n\}$ such that there exists a proof of $T_{j_s}\sigma \vdash s\sigma$, but there is no proof of $T_{j_s-1}\sigma \vdash s\sigma$. In other words, j_s is the index of the first constraint in $\mathbf{C}_1$ from whose source term set s can be constructed. This terminates, since $S_\vdash$ is finite, and, for each member of $S_\vdash$, there are only finitely many constraints in $\mathbf{C}_1$ to consider.

3. By definition of the normal proof, for any solution $\sigma \Vdash \mathbf{C}_1$, any term set T such that $T \rhd u \in \mathbf{C}_1$, any $s, s' \in S_\vdash$, if the minimal size normal proof of $T\sigma \vdash s\sigma$ contains a node labeled $T\sigma \vdash s'\sigma$, then the minimal size normal proof of $T\sigma \vdash s'\sigma$ does not contain a node labeled $T\sigma \vdash s\sigma$. Therefore, σ is consistent with at least one linear ordering $\prec$ on $S_\vdash$ that satisfies the following property:

   - If $s \prec s'$, then the normal proof of $T\sigma \vdash s\sigma$ does not contain any node labeled with $T\sigma \vdash s'\sigma$.

   Of all possible orderings on $S_\vdash$ that satisfy this property, we pick one that also satisfies

   - If $j_s < j_{s'}$, then $s \prec s'$.

   This is possible since $j_s < j_{s'}$ means that there does not exist a proof of $T_{j_s}\sigma \vdash s'\sigma$. Therefore, the proof of $T_{j_s}\sigma \vdash s\sigma$ cannot have a node labeled $T_{j_s}\sigma \vdash s'\sigma$.

   Intuitively, $\prec$ is the order in which members of $S_\vdash$ are derived. Since there are only finitely many possible linear orderings on $S_\vdash$ to consider, we find $\prec$ by exhaustive enumeration.

4. We arrange $s_1, \ldots, s_k \in S_\vdash$ according to the ordering $\prec$, and insert each s_i in the constraint sequence immediately before the $T_{s_i} \rhd u_{s_i}$ constraint. More precisely, we

replace $\mathbf{C}_1$ with

$$\begin{array}{l} T_1 \rhd u_1; \\ \ldots \\ T_{j_{s_1}-1} \rhd u_{j_{s_1}-1}; \\ T_{j_{s_1}} \rhd s_1; \\ T_{j_{s_1}}, s_1 \rhd u_{j_{s_1}}; \\ \ldots \\ T_n, s_1 \rhd u_n \end{array}$$

Call the resulting sequence $\mathbf{C}_1^{(1)}$, and repeat this step for $s_2, \ldots, s_k \in S_{\vdash}$. Given $\mathbf{C}_1^{(i-1)}$, $\mathbf{C}_1^{(i)}$ is constructed by inserting $T \rhd s_i$ immediately before $T \rhd u_{j_{s_i}} \in \mathbf{C}_1^{(i-1)}$ and adding s_i to the term sets of all subsequent constraints.

Let $\mathbf{C}_2 = \mathbf{C}_1^{(k)}$. Without loss of generality, assume that all duplicated constraints are removed from $\mathbf{C}_2$.

In the following lemma, we use $\mathbf{C} \rightsquigarrow \mathbf{C}_2$ as a shorthand for $\mathbf{C} \rightsquigarrow_1 \mathbf{C}_1 \rightsquigarrow_2 \mathbf{C}_2$.

Lemma 6.3 *(Soundness:) For any $\mathbf{C}_2$ such that $\mathbf{C} \rightsquigarrow \mathbf{C}_2$, if there exists a solution $\sigma_2 \Vdash \mathbf{C}_2$, then there exists a solution $\sigma \Vdash \mathbf{C}$.*

(Completeness:) If there exists a solution $\sigma \Vdash \mathbf{C}$, then $\exists \mathbf{C}_2$ such that $\mathbf{C} \rightsquigarrow \mathbf{C}_2$ and $\sigma \Vdash \mathbf{C}_2$.

Proof: To prove soundness, observe that for every constraint $T \rhd u \in \mathbf{C}_1$ there exists $\hat{T} \rhd u \in \mathbf{C}_2$ such that $T \subseteq \hat{T}$. Consider all subterms $s_1, \ldots, s_k \in \hat{T} \setminus T$. By construction of $\mathbf{C}_2$, $\forall s_j\ T_{s_j} \rhd s_j \in \mathbf{C}_2$, $T_{s_1} = T$, and $T_{s_{j+1}} \setminus T_{s_j} = s_j$. Because $\sigma_2 \Vdash \mathbf{C}_2$, there exists a proof of $T\sigma_2 \vdash s_1\sigma_2$. By induction over $\{s_1, \ldots, s_k\}$, there exists a proof of $T\sigma_2 \vdash s_j\sigma_2\ \forall s_j$. Since $\sigma_2 \Vdash \mathbf{C}_2$, there also exists a proof of $\hat{T}\sigma_2 \vdash u\sigma_2$. Therefore, for all $T \rhd u \in \mathbf{C}_1$, there exists a proof of $(\hat{T} \setminus \{s_1, \ldots, s_k\})\sigma_2 = T\sigma_2 \vdash u\sigma_2$. We conclude that $\sigma_2 \Vdash \mathbf{C}_1$. By Lemma 6.1, this implies that there exists a solution $\sigma \Vdash \mathbf{C}$.

Completeness follows directly from the construction of $\mathbf{C}_2$. For any substitution σ such that $\sigma \Vdash \mathbf{C}_1$, there exists a finite set of derivable subterms $S_{\vdash}$. Also, for each derivable subterm $s \in S_{\vdash}$, there exists some constraint $T_{j_s} \rhd u_{j_s} \in \mathbf{C}_1$ such that $s\sigma$ is derivable from $T_{j_s}\sigma$, but not from the preceding $T_i\sigma$. Since we consider all candidate sequences $\mathbf{C}_2$ associated with all possible values of $S_{\vdash}$ and j_s, there exists $\mathbf{C}_2$ such that $\mathbf{C}_1 \rightsquigarrow_2 \mathbf{C}_2$ and $\sigma \Vdash \mathbf{C}_2$. Completeness then follows from Lemma 6.1. □

Lemma 6.4 *Suppose $\sigma \Vdash \mathbf{C}$. Consider $\mathbf{C}_2$ such that $\mathbf{C} \rightsquigarrow \mathbf{C}_2$ and $\sigma \Vdash \mathbf{C}_2$, and any $s, s' \in \mathrm{St}(\mathbf{C}_2)$. If $s \neq s'$, then $s\sigma \neq s'\sigma$.*

Proof: Follows from Lemma 6.2 since $\mathrm{St}(\mathbf{C}_2) = \mathrm{St}(\mathbf{C}_1)$. □

Lemma 6.5 $\forall s_r \in S_\vdash$

$$\mathbf{C}_1^{(r)} \quad = \quad \begin{array}{l} T_1 \rhd t_1; \\ \ldots \\ T_l \rhd t_l; \\ T_{j_{s_r}} \cup \{s' | s' \prec s_r\} \rhd s_r; \\ T_{j_{s_r}} \cup \{s' | s' \prec s_r\} \rhd u_{j_{s_r}}; \\ \ldots \\ T_n \cup \{s' | s' \prec s_r\} \rhd u_n; \end{array}$$

where $T_1 \rhd t_1, \ldots T_l \rhd t_l \in \mathbf{C}_1^{(r-1)}, T_{j_{s_r}} \rhd u_{j_{s_r}}, \ldots T_n \rhd u_n \in \mathbf{C}_1$.

Proof: Proof is by induction over $i \in \{1, ..., r\}$. If $i = 1$, $\mathbf{C}_1^{(1)}$ satisfies the lemma by construction. Suppose the result holds for $\mathbf{C}_1^{(i-1)}$.

By construction,

$$\mathbf{C}_1^{(i)} \quad = \quad \begin{array}{l} T_1 \rhd t_1; \\ \ldots \\ T_l \rhd t_l; \\ T'_{j_{s_i}} \rhd s_i; \\ T'_{j_{s_i}} \cup \{s_i\} \rhd u_{j_{s_i}}; \ldots \\ T'_n \cup \{s_i\} \rhd u_n \end{array}$$

where $T_1 \rhd t_1, \ldots T_l \rhd t_l \in \mathbf{C}_1^{(i-1)}$.

By the induction hypothesis, $m > j_{s_{i-1}}$ implies $T'_m = T_m \cup \bigcup_{s' \preceq s_{i-1}} \{s'\}$ where $T_m \rhd u_m \in \mathbf{C}_1$. By our choice of ordering, $s_{i-1} \prec s_i$ and $j_{s_i} > j_{s_{i-1}}$. Therefore,

$$T'_{j_{s_i}} = T_{j_{s_i}} \cup \bigcup_{s' \preceq s_{i-1}} \{s'\} = T_{j_{s_i}} \cup \bigcup_{s' \prec s_i} \{s'\}$$

and $\forall m \geq j_{s_i}$

$$T'_m \cup \{s_i\} = T_m \cup \bigcup_{s' \preceq s_{i-1}} \{s'\} \cup \{s_i\} = T_m \cup \bigcup_{s' \preceq s_i} \{s'\}$$

. □

Lemma 6.6 $(\forall s_r \in S_\vdash)(\forall i \geq r)$ $T_{j_{s_r}} \cup \{s' | s' \prec s_r\} \rhd s_r$ *appears in* $\mathbf{C}_1^{(i)}$ *before* $T \rhd u_{j_{s_i}}$.

Proof: The proof is by induction over $i \geq r$. For the induction basis, consider that $T_{j_{s_r}} \cup \{s' | s' \prec s_r\} \rhd s_r \in \mathbf{C}_1^{(r)}$ and appears before $T \rhd u_{j_{s_r}}$ by Lemma 6.5. Suppose the result holds for $\mathbf{C}_1^{(i)}$ and consider $\mathbf{C}_1^{(i+1)}$.

By the induction hypothesis, $T_{j_{s_r}} \cup \{s' | s' \prec s_r\} \rhd s_r$ appears in $\mathbf{C}_1^{(i)}$ before $T \rhd u_{j_{s_i}}$. Since $j_{s_i} < j_{s_{i+1}}$, this means that $T_{j_{s_r}} \cup \{s' | s' \prec s_r\} \rhd s_r$ appears in $\mathbf{C}_1^{(i)}$ before $T \rhd u_{j_{s_{i+1}}}$. By Lemma 6.5, $\forall T_l \rhd t_l \in \mathbf{C}_1^{(i)}$ such that $T_l \rhd t_l$ appears before $T \rhd u_{j_{s_{i+1}}}$, $T_l \rhd t_l$ appears in $\mathbf{C}_1^{(i+1)}$ before $T \rhd u_{j_{s_{i+1}}}$. Therefore, $T_{j_{s_r}} \cup \{s' | s' \prec s_r\} \rhd s_r$ appears in $\mathbf{C}_1^{(i+1)}$ before $T \rhd u_{j_{s_{i+1}}}$. □

Lemma 6.7 *If $T \rhd u \in \mathbf{C}_2$ and $s \in S_\vdash$ such that $s' \prec u$, then $s \in T$.*

Proof: By construction of $\mathbf{C}_2$, $u \in S_\vdash$. By Lemma 6.6, $T_{j_s} \cup \{s' | s' \prec s\} \rhd s \in \mathbf{C}_1^{(k)} = \mathbf{C}_2$. □

The most important property of our construction is that every constraint in $\mathbf{C}_2$ is either solved with *one* rule application, or the proof involves only multiplicative rules (M) and (I).

Lemma 6.8 *Let σ be any solution such that $\sigma \Vdash \mathbf{C}_2$ for some $\mathbf{C}_2$ such that $\mathbf{C} \rightsquigarrow \mathbf{C}_2$. Consider any $T \rhd u \in \mathbf{C}_2$ and the last inference of the proof of $T\sigma \vdash u\sigma$.*

- *If $u\sigma \in T\sigma$, then $u \in T$.*
- *If $u\sigma$ is obtained by (UL), then $\langle u, t' \rangle \in T$ for some term t'.*
- *If $u\sigma$ is obtained by (UR), then $\langle t', u \rangle \in T$ for some term t'.*
- *If $u\sigma$ is obtained by (D), then $\{u\}_{t'} \in T$ for some term $t' \in T$.*
- *If $u\sigma$ is obtained by (P), then $u = \langle u_1, u_2 \rangle$ for some terms $u_1, u_2 \in T$.*
- *If $u\sigma$ is obtained by (E), then $u = \{u_1\}_{u_2}$ for some terms $u_1, u_2 \in T$.*
- *If $u\sigma$ is obtained by (F), then $u = f(t')$ for some term $t' \in T$.*

Proof: Consider any $T \rhd u \in \mathbf{C}_2$. Since $\sigma \Vdash \mathbf{C}_2$ is a conservative solution, by Lemma 5.12, there exists a conservative proof of $T\sigma \vdash u\sigma$.

If $u\sigma \in T\sigma$, then there exists $t \in T \subseteq \mathrm{St}(\mathbf{C}_2)$ such that $t\sigma = u\sigma$. ¿From Lemma 6.4, it follows that $u = t \in T$. We conclude that, whenever $u\sigma \in T\sigma$, $T \rhd u \in \mathbf{C}_2$ is such that $u \in T$.

If $u\sigma \notin T\sigma$, consider the *last* inference of the conservative proof of $T\sigma \vdash u\sigma$. It must have the form

$$\frac{T\sigma \vdash v_1 \quad [\ldots\, T\sigma \vdash v_k]}{T\sigma \vdash u\sigma}$$

(the $T\sigma \vdash v_i$ premises for $i > 1$ may be absent). If this inference is an instance of any rule other than (M) or (I), then $i \leq 2$ and, by Definition 5.11, $v_{1,2} \in \mathrm{St}(\mathbf{C}_2)\sigma$, *i.e.*, there exists $v'_{1,2} \in \mathrm{St}(\mathbf{C}_2)$ such that $v'_1\sigma = v_1$ and $v'_2\sigma = v_2$.

Since there exist proofs of $T\sigma \vdash v'_{1,2}\sigma$, it must be that $v'_{1,2} \in S_\vdash$. Since the proof of $T\sigma \vdash u\sigma$ contains nodes labeled $T\sigma \vdash v'_{1,2}\sigma$, by definition of ordering $\prec$ it must be that $v'_{1,2} \prec u$. By Lemma 6.7, $v'_{1,2} \in T$. We conclude that the proof of $T\sigma \vdash u\sigma$ consists of one inference:

$$\frac{T\sigma \vdash v'_1\sigma \in T\sigma \quad [T\sigma \vdash v'_2\sigma \in T\sigma]}{T\sigma \vdash u\sigma}$$

Consider all possible cases for this inference rule other than (M) or (I).

If the rule is (UL), then $v'_1\sigma = \langle u\sigma, t \rangle$ for some term t. By Lemma 5.10, $v'_1\sigma \in \bar{\mathcal{S}}(\mathbf{C})\sigma$. By Lemma 5.3, this means that there exists $\langle u', t' \rangle \in \mathrm{St}(\mathbf{C}_2)$ such that $\langle u', t' \rangle\sigma = v'_1\sigma$. By Lemma 6.4, this implies that $v'_1 = \langle u', t' \rangle$. Since $u', u \in \mathrm{St}(\mathbf{C}_2)$, Lemma 6.4 also implies that $u' = u$. Therefore, $v'_1 = \langle u, t' \rangle \in T$. We conclude that, whenever $u\sigma$ is obtained by (UL) rule, $T \rhd u \in \mathbf{C}_2$ is such that $\langle u, t' \rangle \in T$. The proofs for (UR) and (D) is similar.

If the rule is (E), then $u\sigma = \{v_1\sigma\}_{v_2\sigma}$. By Lemma 5.10, $u\sigma \in \bar{\mathcal{S}}(\mathbf{C})\sigma$. By Lemma 5.3, there exists $u' = \{v'_1\}_{v'_2} \in \mathrm{St}(\mathbf{C}_2)$ and $u\sigma = u'\sigma$. Since $u \in \mathrm{St}(\mathbf{C}_2)$ by Definition 5.11, Lemma 6.4 implies $u' = u$. We conclude that, whenever $u\sigma$ is obtained by (E) rule, $u = \{u_1\}_{u_2}$ and $T \rhd \{u_1\}_{u_2} \in \mathbf{C}_2$ where $u_{1,2} \in T$. The proofs for (P) and (F) are similar. □

Running example. In our running example, we guess that no subterms (other than those already appearing as target terms) are derivable. Therefore, after removing duplicated constraints from $\mathbf{C}_2$, $\mathbf{C}_2 = \mathbf{C}_1$.

6.3 Eliminate all inferences other than (M) or (I)

In the third step $\rightsquigarrow_3$, we eliminate all constraints which can be satisfied by a single application of an inference rule other than (M) or (I). This step is deterministic. Lemma 6.8 implies that all such constraints can be found by syntactic inspection. Let $\mathbf{C}_3$ be the resulting constraint sequence.

In the following lemma, we use $\mathbf{C} \rightsquigarrow \mathbf{C}_3$ as a shorthand for $\mathbf{C} \rightsquigarrow_1 \mathbf{C}_1 \rightsquigarrow_2 \mathbf{C}_2 \rightsquigarrow_3 \mathbf{C}_3$.

Lemma 6.9 *(Soundness:) For any $\mathbf{C}_3$ such that $\mathbf{C} \rightsquigarrow \mathbf{C}_3$, if there exists a solution $\sigma_3 \Vdash \mathbf{C}_3$, then there exists a solution $\sigma \Vdash \mathbf{C}$.*

(Completeness:) If there exists a solution $\sigma \Vdash \mathbf{C}$, then $\exists \mathbf{C}_3$ such that $\mathbf{C} \rightsquigarrow \mathbf{C}_3$ and $\sigma \Vdash \mathbf{C}_3$.

Proof: To prove soundness, suppose $\sigma_3 \Vdash \mathbf{C}_3$. By Lemma 6.8, for every constraint $T \rhd u$ eliminated by $\rightsquigarrow_3$, there exists a one-step proof of $T\sigma \vdash u\sigma$ for *any* substitution σ. This means that there exists a proof of $T\sigma_3 \vdash u\sigma_3$ for every constraint $T \rhd u \in \mathbf{C}_2 \setminus \mathbf{C}_3$. By our assumption that $\sigma_3 \Vdash \mathbf{C}_3$, there exists a proof of $T\sigma_3 \vdash u\sigma_3$ for every constraint $T \rhd u \in \mathbf{C}_3$. Therefore, $\sigma_3 \Vdash \mathbf{C}_2$. Using Lemma 6.3, this is sufficient to demonstrate soundness.

To prove completeness, observe that $\mathbf{C}_3 \subseteq \mathbf{C}_2$. Therefore, if $\sigma \Vdash \mathbf{C}_2$, then $\sigma \Vdash \mathbf{C}_3$. Then apply Lemma 6.3. □

Lemma 6.10 *If $\sigma \Vdash \mathbf{C}_3$, then $\forall T \rhd u \in \mathbf{C}_3$, the proof of $T\sigma \vdash u\sigma$ uses (M) and (I) rules only.*

Proof: Consider a minimal size conservative proof of $T\sigma \vdash u\sigma$ and suppose it contains an inference

$$\frac{T\sigma \vdash v_1 \in T\sigma \quad [T\sigma \vdash v_2 \in T\sigma]}{T\sigma \vdash v}$$

that is an instance of a rule other than (M) or (I). By construction of $\mathbf{C}_3$, this cannot be the last inference, therefore, $v \neq u$. By Definition 5.11 of a conservative proof, it must be that $v \in \mathrm{St}(\mathbf{C}_3)\sigma \subseteq \mathrm{St}(\mathbf{C}_2)\sigma$, *i.e.*, there exists $v' \in \mathrm{St}(\mathbf{C}_2)$ such that $v'\sigma = v$. By definition of $\prec$, it must be that $v' \prec u$. By Lemma 6.7, this implies that $v' \in T$. This contradicts minimality of the proof. Therefore, the proof of $T\sigma \vdash u\sigma$ does not contain any inferences other than (M) or (I). □

Lemma 6.11 *If $x \in \mathrm{Var}(\mathbf{C}_3)$ let $T_{k_x} \rhd u_{k_x} \in \mathbf{C}_3$ be the constraint in which x occurs for the first time. Then $u = x^{q_x} \cdot \prod_{j \geq 0} u_{k_x j}$ where q_x is an integer, $u_{k_x j}$ are not headed with $\cdot$, and $x \notin \mathrm{St}(T_{k_x} \cup \{u_{k_x j} | j \geq 0\})$.*

Proof: By the variable origination property (Definition 3.3), $x \notin \mathrm{St}(T_{k_x})$. Suppose $x \in \mathrm{St}(u_{k_x j})$ for some j and $u_{k_x j} \neq x^{q_x}$. Lemma 6.10 implies that there exists $t \in \mathrm{St}(T_{k_x})$ such that $t\sigma = u_{k_x j}\sigma$. Since $\mathrm{St}(\mathbf{C}_3) \subseteq \mathrm{St}(\mathbf{C}_2)$, Lemma 6.4 implies that $t = u_{k_x j}$. This means that $x \in \mathrm{St}(T_{k_x})$, and contradicts the variable origination property. □

Definition 6.12 *Define $\mathcal{Q}_{max} = \prod_{x \in \mathrm{Var}(\mathbf{C}_3)} q_x$ where q_x is the power of x in the constraint in which it occurs for the first time.*

Running example. In our example, we guess the first and third constraints were obtained by application of rules (M) and (I) only. We eliminate the second constraint, obtaining the following $\mathbf{C}_3$:

$$\begin{array}{l} a \cdot b \rhd a^2 \cdot X \ ; \\ a \cdot b, \{a\}_b, b \cdot X \rhd a^6 \end{array}$$

6.4 Substitute target terms that introduce new variables

In the fourth step $\rightsquigarrow_4$, we take each target term in which some variable occurs for the first time and introduce a new variable, substituting the *entire* term in question. This step is deterministic. For example, if x occurs for the first time in constraint $T_i \rhd a \cdot x^2$, let $\theta_i = [x \mapsto \hat{x}^{\frac{1}{2}} \cdot a^{-\frac{1}{2}}]$ where $\hat{x}$ is a new variable, and apply θ_i to the entire constraint sequence. In the resulting constraint sequence, each variable $\hat{x}$ occurs for the first time in some constraint of the form $T_i \rhd \hat{x}$.

In the definition below, a term with a fractional exponent $t^{\frac{m}{n}}$ represents a term u such that $u^n = t^m$. Obviously, taking a root of some element of a finite field requires that the root in question exist. Our definition of $\hat{x}$ *guarantees* the existence of all newly introduced roots since the value of every rational exponent introduced by $\rightsquigarrow_4$ appears explicitly in the protocol specification. For example, if we replace x^2 with $\hat{x}$, then $\hat{x}^{\frac{m}{2}}$ will appear in the constraint sequence if and only if x^m appears in the protocol specification (note that $x^m = (x^2)^{\frac{m}{2}}$).

Recall from Definition 3.3 that k_x is the index of the constraint $T_i \rhd u_i$ in which variable x occurs for the first time, and that, by Lemma 6.11, $u_i = x^{q_x} \cdot \prod_{j \geq 0} u_{ij}$ for some integer q_x.

Definition 6.13 *If $T_i \rhd u_i \in \mathbf{C}_3$, define*

$$\theta_i = \begin{cases} [x \mapsto \hat{x}^{\frac{1}{q_x}} \cdot \prod u_{ij}^{-\frac{1}{q_x}}] & \textit{if } i = k_x \textit{ for some } x;\ \hat{x} \textit{ is a fresh variable} \\ \emptyset & \textit{otherwise} \end{cases}$$

If more than one variable appears for the first time in u_i, any one of them may be chosen.

Let $\mathbf{C}_4 = \mathbf{C}_3\theta_1 \ldots \theta_{N_3}$ where N_3 is the number of constraints in $\mathbf{C}_3$. Although only integer powers appear in $\mathbf{C}_3$, $\mathbf{C}_4$ may contain rational powers.

Assume that term sets T_i appearing in $\mathbf{C}_4$ have been ordered, *i.e.*, $T_i = \{t_{i1}, t_{i2}, \ldots\}$.

Definition 6.14 $\mathbf{C}_4$ *is* well-ordered *if $i < i'$ implies that $t_{ij} = t_{i'j} \in T_{i'}$ when $j \leq |T_i|$.*

Informally, Definition 6.14 means that term sets T_i are consistently ordered so that if the same term appears in multiple sets, it always appears in the same position. Due to the monotonicity property of constraint sequences (Definition 3.1), if $i < i'$, then $T_i \subseteq T_{i'}$. Without loss of generality, we can assume that $\mathbf{C}_4$ is well-ordered.

Lemma 6.15 *For any rational r appearing as a power of some term in $\mathbf{C}_4$, $r \cdot \mathcal{Q}_{max}$ is an integer.*

Proof: Follows directly from Definitions 6.12 and 6.13. □

Lemma 6.16 *If $T \rhd u \in \mathbf{C}_4$, the proof of $T\sigma \vdash u\sigma$ uses (M) and (I) rules only.*

Proof: Follows from Lemma 6.10 and construction of $\mathbf{C}_4$. □

Lemma 6.17 *If $x \in \mathrm{Var}(\mathbf{C}_4)$, x occurs for the first time in some constraint of the form $T \rhd x \in \mathbf{C}_4$ where $x \notin \mathrm{St}(T)$.*

Proof: Follows from Definition 6.13 and construction of $\mathbf{C}_4$. □

In the following lemma, we use $\mathbf{C} \leadsto \mathbf{C}_4$ as a shorthand for $\mathbf{C} \leadsto_1 \mathbf{C}_1 \leadsto_2 \mathbf{C}_2 \leadsto_3 \mathbf{C}_3 \leadsto_4 \mathbf{C}_4$.

Lemma 6.18 *There exists a solution $\sigma \Vdash \mathbf{C}$ if and only if there exists a solution $\sigma_4 \Vdash \mathbf{C}_4$ for some $\mathbf{C}_4$ such that $\mathbf{C} \leadsto \mathbf{C}_4$.*

Proof: Substitution 6.13 is simply a renaming of terms in $\mathbf{C}_3$ and does not introduce or lose any solutions. The result follows directly from Lemma 6.9. □

Running example. In our example, $\theta_1 = [X \mapsto \hat{X} \cdot a^{-2}], \theta_2 = \emptyset$. Therefore, $\mathbf{C}_4 = \mathbf{C}_3\theta_1\theta_2$ is:

$$a \cdot b \rhd \hat{X} \ ;$$
$$a \cdot b, \{a\}_b, b \cdot \hat{X} \cdot a^{-2} \rhd a^6$$

6.5 Derive a system of quadratic Diophantine equations

In the last step of the constraint solving procedure, we convert each constraint sequence $\mathbf{C}_4$ into a system of quadratic Diophantine equations which is solvable if and only if $\sigma \Vdash \mathbf{C}_4$ for some σ.

Diophantine equations are polynomial equations in any number of variables with integer coefficients, where only integer solutions are permitted. There is no general procedure for determining the solvability of a Diophantine equation or finding a general solution; that was Hilbert's Tenth Problem. A system of Diophantine equations must be solved in common by the same substitution. One can reduce a single Diophantine equation of any degree to a system of quadratic equations by introducing variables. For example, in the equation $x^3 = 27$ one can let $y = x^2$, reducing the original equation to $xy = 27$. The system of the latter two equations is equivalent to the first. Hence the solvability of systems of quadratic Diophantine equations is also undecidable.

To solve the protocol analysis problem, we generate a system of quadratic Diophantine equations. In our case, we can demonstrate that the system we get is solvable if and only if a particular *linear* subsystem is solvable. Luckily, solvability of systems of linear Diophantine equations is decidable (see, *e.g.*, [CD94]). This problem is equivalent to solving an integer linear programming problem, which is known to be intractable in general, but there are efficient solution techniques that work well most of the time (just as the simplex method works well for real linear programming).

The key to this result is Lemma 6.19. Intuitively, we prove that, for every constraint $T \rhd u \in \mathbf{C}_4$, the target term $u\sigma$ must be equal to some product of integer powers of *non-variable* terms appearing in set T. We then simply represent each power as an integer variable, and convert the constraint satisfaction problem for each constraint into a system of linear Diophantine equations.

There is a complication along the way. In addition to the linear system corresponding to the solvability of a given $T \rhd u$ constraint, the integer variables in question must also satisfy a special system of *quadratic* equations. We show that this quadratic system *always* has a solution. Therefore, only the linear system needs to be solved to determine whether the constraint is satisfiable. Any solution of the linear system will automatically satisfy the quadratic system.

For any term t, define $\Phi(t)$ to be the set of all top-level factors of t. If $t = t_1^{r_1} \cdot \ldots \cdot t_n^{r_n}$ where none of t_i are headed with $\cdot$ and all t_i are distinct, then $\Phi(t) = \{t_1^{r_1}, \ldots, t_n^{r_n}\}$. For example, $\Phi(a^{-2} \cdot b^{\frac{3}{2}}) = \{a^{-2}, b^{\frac{3}{2}}\}$. Define $\Psi(t) = \{t_i^{r_i} \in \Phi(t) \mid t_i \neq x \in \mathrm{Var}(\mathbf{C}_4)\}$ to be the set of all non-variable factors of t. Let $\psi(t) = \prod_{f \in \Psi(t)} f$, *i.e.*, $\psi(t)$ is t with all factors of the form x^r removed. For example, $\psi(a \cdot (\{x\}_k)^3 \cdot x^{\frac{2}{5}}) = a \cdot (\{x\}_k)^3$. Obviously, if t does not contain variables among top-level factors, then $\psi(t) = t$.

Lemma 6.19 *If $x \in \mathrm{Var}(\mathbf{C}_4)$, let k_x be the index of the constraint in which variable x occurs for the first time. Then if $\sigma \Vdash \mathbf{C}_4$ and $T_i \rhd u_i \in \mathbf{C}_4$*

$$u_i\sigma = \prod_{t_{ij} \in T_i} (\psi(t_{ij})\sigma)^{\hat{z}[i,j]} \tag{1}$$

such that

$$\hat{z}[i,j] = z[i,j] + \sum_{j'>j}^{|T_i|} (\sum_{x^r \in \Phi(t_{ij'})} (\hat{z}[k_x, j] \cdot r \cdot z[i,j'])) \tag{2}$$

for some integers $\hat{z}[i,j], z[i,j]$, where $1 \leq i \leq |\mathbf{C}_4|$, and by convention if $j > |T_{k_x}|$ then $\hat{z}[k_x, j] = 0$.

Before we begin the proof, it is helpful to give a small example that gives some insight into how the exponents are computed.

Consider the constraints $t_{11} \rhd x$ and $t_{i1}, t'_{i2} \cdot x \rhd u_i$, where x is a variable and the other terms do not contain variables. Each target term is a product of powers of the terms on the left. Thus, we may write $x = t_{11}^{z[1,1]}$ and $\hat{z}[1,1] = z[1,1]$. We also have $u_i = t_{i1}^{z[i,1]} \cdot (t'_{i2} \cdot x)^{z[i,2]}$. To find $\hat{z}[i,j]$, we substitute for x and also note that $t_{11} = t_{i1}$ because $\mathbf{C}_4$ is well-ordered. Hence,

$$u_i = t_{i1}^{z[i,1]+\hat{z}[1,1]\cdot z[i,2]} \cdot {t'}_{i2}^{z[i,2]}$$

Therefore $\hat{z}[i,1] = z[i,1] + \hat{z}[1,1] \cdot z[i,2]$. This expression should be recognizable as a special case of equation (2).

Proof:(of Lemma 6.19) The proof is by induction over the length of the constraint sequence. For the induction basis, consider $T_1 \rhd u_1 \in \mathbf{C}_4$. By Lemma 6.16, the proof of $T_1\sigma \vdash u_1\sigma$ contains only rules (M) and (I). Therefore, $u_1\sigma = \prod_{t_{1j} \in T_1} (t_{1j}\sigma)^{z[1,j]}$ for some integers $z[1,j]$, where $1 \leq j \leq |T_1|$. By Lemma 6.17, no variables occur in T_1. Therefore, for all j, $t_{1j} = \psi(t_{1j})$ and for all $x \in \mathrm{Var}(\mathbf{C}_4), j'$, and $r, x^r \notin \Phi(t_{1j'})$. Then for all $j, \hat{z}[1,j] = z[1,j]$, and we obtain $u_1\sigma = \prod_{t_{1j} \in T_1} (\psi(t_{1j})\sigma)^{\hat{z}[1,j]}$ This completes the base case.

Now suppose the lemma is true for all constraints up to and including $T_{i-1} \rhd u_{i-1}$, $i \geq 2$. To complete the induction, we need to prove it for $T_i \rhd u_i$. Applying Lemma 6.10 to $T_i \rhd u_i$, we obtain that

$$u_i\sigma = \prod_{t_{ij'} \in T_i} (t_{ij'}\sigma)^{z[i,j']} \tag{3}$$

Now consider any $t_{ij'}$ from the above product, and fix it. By definition of $\psi(t_{ij'})$, $t_{ij'} = \psi(t_{ij'}) \cdot x_1^{r_1} \cdot \ldots \cdot x_m^{r_m}$ for some variables $x_1, \ldots, x_m$ and rational constants $r_1, \ldots, r_m$ where $m \geq 0$. Consider any variable $x \in \{x_1, \ldots, x_m\}$, and let k_x be the index of the first constraint in which x occurs. By Lemma 6.17, the fact that x occurs in T_i implies that $T_i \rhd u_i$ cannot be the first constraint in which x occurs. There must exist a preceding constraint of the form $T_{k_x} \rhd x \in \mathbf{C}_4$ and $k_x < i$. The induction hypothesis holds for this constraint, thus $x\sigma = \prod_{t_{k_x j} \in T_{k_x}} (\psi(t_{k_x j})\sigma)^{\hat{z}[k_x,j]}$. By monotonicity (Definition 3.1), $T_{k_x} \subseteq T_i$. Since all sets T_i are ordered according to Definition 6.14, this means that $(\forall j \leq |T_{k_x}|)\ t_{k_x j} = t_{ij}$. Moreover, since x occurs in $t_{ij'}$, Lemma 6.17 implies that $|T_{k_x}| < j'$. We set $\hat{z}[k_x, j] = 0$ for all $j > |T_{k_x}|$, and we replace each $t_{k_x j}$ with the corresponding t_{ij}, obtaining

$$x\sigma = \prod_{j<j'} (\psi(t_{ij})\sigma)^{\hat{z}[k_x,j]} \tag{4}$$

Substituting values for $x_1\sigma, \ldots, x_m\sigma$ given by equation (4) into equation (3), we obtain that

$$u_i\sigma = \prod_{t_{ij'} \in T_i} (\ \psi(t_{ij'})\sigma \cdot \prod_{x^r \in \Phi(t_{ij'})} (\prod_{j<j'} (\psi(t_{ij})\sigma)^{\hat{z}[k_x,j]\cdot r})\ )^{z[i,j']}$$

Distributing the exponent $z[i,j']$, obtain

$$\begin{aligned} u_i\sigma &= \textstyle\prod_{t_{ij'} \in T_i} (\ (\psi(t_{ij'})\sigma)^{z[i,j']} \cdot \prod_{x^r \in \Phi(t_{ij'})} (\prod_{j<j'} (\psi(t_{ij})\sigma)^{\hat{z}[k_x,j]\cdot r\cdot z[i,j']})\ ) \\ &= \textstyle\prod_{t_{ij'} \in T_i} (\psi(t_{ij'})\sigma)^{z[i,j']}\ \cdot \\ &\qquad \textstyle\prod_{t_{ij'} \in T_i} (\ \prod_{x^r \in \Phi(t_{ij'})} (\ \prod_{j<j'} (\psi(t_{ij})\sigma)^{\hat{z}[k_x,j]\cdot r\cdot z[i,j']}\ )) \\ &= \textstyle\prod_{t_{ij} \in T_i} (\psi(t_{ij})\sigma)^{z[i,j]}\ \cdot \\ &\qquad \textstyle\prod_{t_{ij'} \in T_i} (\ \prod_{j<j'} (\ (\psi(t_{ij})\sigma)^{\sum_{x^r \in \Phi(t_{ij'})} (\hat{z}[k_x,j]\cdot r\cdot z[i,j'])}\ )) \end{aligned}$$

Observing that $\prod_{t_{ij'} \in T_i} (\prod_{j<j'} f(\ldots t_{ij} \ldots)) = \prod_{t_{ij} \in T_i} (\prod_{j'>j} f(\ldots t_{ij} \ldots))$, we obtain

$$\begin{aligned} u_i\sigma &= \prod_{t_{ij} \in T_i} (\psi(t_{ij})\sigma)^{z[i,j]} \quad \cdot \\ &\qquad \prod_{t_{ij} \in T_i} (\ \prod_{j'>j} (\ (\psi(t_{ij})\sigma)^{\sum_{x^r \in \Phi(t_{ij'})} (\hat{z}[k_x,j] \cdot r \cdot z[i,j'])} \ )) \\ &= \prod_{t_{ij} \in T_i} (\ (\psi(t_{ij})\sigma)^{z[i,j]} \cdot (\psi(t_{ij})\sigma)^{\sum_{j'>j} (\sum_{x^r \in \Phi(t_{ij'})} (\hat{z}[k_x,j] \cdot r \cdot z[i,j']))} \ ) \\ &= \prod_{t_{ij} \in T_i} (\psi(t_{ij})\sigma)^{z[i,j] + \sum_{j'>j} (\sum_{x^r \in \Phi(t_{ij'})} (\hat{z}[k_x,j] \cdot r \cdot z[i,j']))} \end{aligned}$$

This completes the induction. □

6.6 Convert into a system of linear Diophantine equations

We now convert each constraint into an equivalent system of linear Diophantine equations. If this system is unsolvable, the constraint cannot be satisfied and the entire constraint sequence does not have a solution. If, on the other hand, there exist some values of $\hat{z}[i,j]$ that solve the linear Diophantine system, we will demonstrate that quadratic equations (2) are guaranteed to have a solution.

Consider any $T_i \triangleright u_i \in \mathbf{C}_4$. By Lemma 6.19, $u_i\sigma = \prod_{t_{ij} \in T_i} (\psi(t_{ij})\sigma)^{\hat{z}[i,j]}$. By definition, $\psi(t_{ij})$ does not contain any variables as top-level factors. It is possible that $x_k^{p_k} \in \Phi(u_i)$ for some variable x_k and rational p_k. Applying Lemma 6.17 and Lemma 6.19, for all $x_k \in \mathrm{Var}(\mathbf{C}_4)$ we obtain that $x_k\sigma = \prod_{t_{k_x j} \in T_{k_x}} (\psi(t_{k_x j})\sigma)^{\hat{z}[k_x,j]}$. Therefore, equation (1) can be rewritten as

$$\begin{gathered} \prod_{x_k^{p_k} \in \Phi(u_i)} (\prod_{t_{k_x j} \in T_{k_x}} (\psi(t_{k_x j})\sigma)^{\hat{z}[k,j]})^{p_k} \cdot \prod_{u_{il} \in \Psi(u_i)} u_{il}\sigma = \\ \prod_{t_{ij} \in T_i} (\psi(t_{ij})\sigma)^{\hat{z}[i,j]} \end{gathered} \tag{5}$$

For any variable x_k occurring in u_i, it must be that $k_x \leq i$ since k_x is the index of the first constraint in which x_k occurs. By Definitions 3.1 and 6.14, $t_{k_x j} = t_{ij}$. Dividing the right-hand side of equation (5) by

$$\prod_{x_k^{p_k} \in \Phi(u_i)} (\prod_{t_{k_x j} \in T_{k_x}} (\psi(t_{k_x j})\sigma)^{\hat{z}[k_x,j]})^{p_k}$$

we obtain

$$\prod_{u_{il} \in \Psi(u_i)} u_{il}\sigma = \prod_{t_{ij} \in T_i} (\psi(t_{ij})\sigma)^{y[i,j]} \tag{6}$$

where

$$y[i,j] = \hat{z}[i,j] - \sum_{x_k^{p_k} \in \Phi(u_i)} p_k \cdot \hat{z}[k_x,j] \tag{7}$$

Recall that $\hat{z}[k_x,j] = 0$ if $j > |T_{k_x}|$.

Let $\mathrm{Fac}(\mathbf{C}_4) = \bigcup_{T_i \triangleright u_i \in \mathbf{C}_4} (\Psi(u_i) \cup \bigcup_{t_{ij} \in T_i} \Psi(t_{ij}))$ be the set of all factors appearing in equations (6). Since $\mathrm{Fac}(\mathbf{C}_4) \subseteq \mathrm{St}(\mathbf{C}_4) \subseteq \mathrm{St}(\mathbf{C}_2)$, by Lemma 6.4, if $t, t' \in \mathrm{Fac}(\mathbf{C}_4)$

and $t \neq t'$, then $t\sigma \neq t'\sigma$. Therefore, if $w \in \mathrm{Fac}(\mathbf{C}_4)$ and $T_i \rhd u_i \in \mathbf{C}_4$, the following system of linear equations must hold:

$$\underbrace{q}_{\substack{\text{if } w^q \in \Psi(u_i),\\ 0 \text{ otherwise}}} = \sum\nolimits_{t_{ij}\in T_i} \underbrace{q_j \cdot y[i,j]}_{\substack{\text{if } w^{q_j} \in \Psi(t_{ij}),\\ 0 \text{ otherwise}}} \tag{8}$$

where $y[i,j]$ are integer variables (i ranges over the length of the constraint sequence, and, for each i, j ranges from 1 to $|T_i|$), and $q, q_1, \ldots, q_{|T_i|}$ are rational constants. We multiply both sides of each equation (8) by the lowest common multiplier of the denominators of $q, q_1, \ldots, q_{|T_i|}$, and obtain a system of linear Diophantine equations over $y[i,j]$.

Lemma 6.20 $\mathbf{C}_4$ *has a solution if and only if the system of equations (8) has a solution in integers.*

Proof: It follows immediately from the reduction in this section that if system (8) does *not* have a solution in integers, then $\mathbf{C}_4$ does not have a solution, either. To complete the proof, it is the necessary to show that if system (8) has a solution in integers, then system (7) and, especially, the quadratic system (2) also have a solution.

Let $\{y[i,j]\}$ be any solution of system (8). First, set $y[k_x, j] = 0$ for all $x \in \mathbf{C}_4$ and all j. Since $\Psi(u_{k_x}) = \Psi(x) = \emptyset$, equation (8) degenerates into $0 = \sum_{t_{k_x j}\in T_{k_x}} q_j \cdot y[k_x, j]$ and, clearly, is still satisfied. By Lemma 6.17, $u_{k_x} = x_k$. Therefore,

$$\sum_{x_k^{p_k}\in\Phi(u_{k_x})} p_k \cdot \hat{z}[k_x, j] = \hat{z}[k_x, j]$$

and $y[k_x, j] = \hat{z}[k_x, j] - \hat{z}[k_x, j] = 0$. System (7) is thus satisfied by $y[k_x, j] = 0$ as well.

Now, set $\hat{z}[k_x, j] = \mathcal{Q}_{max}$ for all $x \in \mathbf{C}_4$ and all j. Recall from Definition 6.12 and Lemma 6.15 that $\mathcal{Q}_{max}$ is an integer constant such $r \cdot \mathcal{Q}_{max}$ is an integer for any rational power r appearing in $\mathbf{C}_4$. We need to demonstrate that, provided $\hat{z}[k_x, j] = \mathcal{Q}_{max}$, systems (7) and (2) are solvable in integers.

First, consider system (7). If $i = k_x$ for some k_x, it degenerates into $0 = \mathcal{Q}_{max} - \mathcal{Q}_{max}$ (see above). If for all $k_x, i \neq k_x$, it becomes $y[i,j] = \hat{z}[i,j] - \sum_{x_k^{p_k}\in\Phi(u_i)} p_k \cdot \mathcal{Q}_{max}$, and is solved simply by setting $\hat{z}[i,j] = y[i,j] + \sum_{x_k^{p_k}\in\Phi(u_i)} p_k \cdot \mathcal{Q}_{max}$. This works since by Lemma 6.15 $p_k \cdot \mathcal{Q}_{max}$ is an integer.

It remains to show that the quadratic system (2) has a solution in integers. Pick any $T_i \rhd u_i \in \mathbf{C}_4$ and fix it. Proof is by induction over all values of j from $|T_i|$ to 1. For the base case, consider $j = |T_i|$. In this case, there are no $j' > j$, and we simply set $z[i,j] = \hat{z}[i,j]$.

Now suppose the lemma is true for $z[i, j+1], \ldots, z[i, |T_i|]$. To complete the proof, it is sufficient to show that there exists an integer value for $z[i,j]$ that satisfies equation (2). Observe that $z[i,j']$ is an integer for all $j' > j$ (by the induction hypothesis), and $\hat{z}[k_x, j] \cdot r = \mathcal{Q}_{max} \cdot r$ is an integer for all x such that $x^r \in \Phi(t_{ij'})$ (by Lemma 6.15). Therefore, $z[i,j] = \hat{z}[i,j] - \sum_{j'>j}(\sum_{x^r\in\Phi(t_{ij'})}(\hat{z}[k_x, j] \cdot r \cdot z[i,j']))$ is an integer solution for equation (2). This completes the induction and proves that, assuming linear system (8) has an integer solution for $y[i,j]$, systems (7) and (2) are solvable in integers, too. □

Running example. In our running example, we are solving the following $\mathbf{C}_4$:

$$\begin{array}{l} a \cdot b \rhd \hat{X} \ ; \\ a \cdot b, \{a\}_b, b \cdot \hat{X} \cdot a^{-2} \rhd a^6 \end{array}$$

Applying equation (6), $\mathbf{C}_4$ has a solution *iff* the following system is solvable in integers:

$$\begin{array}{rcl} 1 & = & (a \cdot b)^{y[1,1]} \\ a^6 & = & (a \cdot b)^{y[2,1]} \cdot (\{a\}_b)^{y[2,2]} \cdot (b \cdot a^{-2})^{y[2,3]} \end{array}$$

Note that $\Psi(u_1) = \emptyset$, therefore, $\prod_{u_{1l} \in \Psi(u_1)} u_{1l}\sigma = 1$, and $\psi(b \cdot \hat{X} \cdot a^{-2}) = b \cdot a^{-2}$.

We set $y[1,1] = 0$ since $k_x = 1$, and convert the second equation into an equivalent system of linear Diophantine equations (8), treating non-atomic terms such as $\{a\}_b$ as constants:

$$\begin{array}{rcl} 6 & = & y[2,1] - 2 \cdot y[2,3] \\ 0 & = & y[2,2] \\ 0 & = & y[2,1] + y[2,3] \end{array}$$

This system has the following integer solution: $y[2,1] = 2, y[2,2] = 0, y[2,3] = -2$. Therefore, the constraint sequence has a solution. In this example, $\mathcal{Q}_{max} = 1$, therefore, $\hat{z}[1,1] = 1$, and $\hat{X} = (a \cdot b)^{\hat{z}[1,1]} = a \cdot b$. Reconstructing the values of original variables, we obtain $X = \hat{X} \cdot a^{-2} = a^{-1} \cdot b$.

Theorem 6.21 (Soundness and completeness) *Symbolic constraint sequence* $\mathbf{C}$ *has a solution if and only if the system of linear equations (8) has a solution in integers for some* $\mathbf{C}_4$ *such that* $\mathbf{C} \rightsquigarrow \mathbf{C}_4$.

Proof: Follows immediately from lemmas 6.18 and 6.20. □

Corollary 6.22 (Decidability with xor) *If* $\cdot$ *is interpreted as* xor *(*i.e., $t^{-1} = t$*), then symbolic trace reachability is decidable.*

Corollary 6.23 (Decidability with free term algebra) *If there are no operators with algebraic properties, symbolic trace reachability is decidable.*

7 Extension to Group Diffie-Hellman

In this section, we extend the constraint solving approach developed in Section 6 to protocols such as group Diffie-Hellman (GDH) [STW96]. Our extension, however, applies only in a restricted setting. We assume that the Abelian group (multiplication) operator appears *only* in the exponents. In particular, exponentials are not multiplied with each other, *i.e.*, terms such as $g_1^x \cdot g_2^y$ do not appear in the protocol specification, nor is the attacker permitted to multiply exponentials. This restriction is necessary to preserve decidability, since it has been shown that unification (and, therefore, the symbolic analysis problem) is undecidable in the presence of equational theories for *both* Abelian groups and exponentiation [KNW02, KNW03]. This does not affect our ability to analyze protocols such as GDH since they satisfy the restriction (a similar restriction is adopted by [PQ01]).

$$\begin{array}{rcl} t^{\mathbf{1}} & \rightarrow & t \\ (t^u)^v & \rightarrow & t^{u \cdot v} \end{array}$$

Figure 4: Normalization rules for exponentials

We extend the message term constructors of Figure 1 with terms t^u representing exponentials. We also extend the rules of Figure 2 with the rules for exponentials, as shown in Figure 4. The rules of Figure 4 were shown in [MN02] to lead to unique normal forms up to associativity and commutativity of the $\cdot$ operator.

In this paper, we consider only protocols in which all exponentiation is ultimately from a constant base, that is, the normal form of every exponential ground term is g^t where g is a public constant. For purposes of specification and analysis, exponentials are regarded as a separate type. If a variable x is of type exponential, we replace it with a term $g^{x'}$ where x' is a new variable that cannot be an exponential or have an exponential subterm. Products of exponents arise from reductions of the form $(g^x)^y = g^{x \cdot y}$.

In GDH, exponential terms are integers mod p for some prime p. The multiplicative subgroup of $\mathbf{Z}_p$ has order $p - 1$. The base g is chosen to generate the cyclic subgroup of some prime order q that divides $p - 1$. Since $g^q \equiv 1 \equiv g^0 \bmod p$, exponents of g are effectively reducible mod q. Thus, non-zero exponents lie in the multiplicative subgroup of $\mathbf{Z}_q$.

For protocols with exponentials, we apply the basic constraint solving procedure of Section 6 with a modification. We begin as usual with $\mathbf{C}$, but when we arrive at $\mathbf{C}_3$ we find some constraints of the form

$$t_1, \ldots, t_m \rhd g^u \tag{9}$$

Under the Computational Diffie-Hellman Assumption (it is not feasible to compute g^{xy} from g^x and g^y), the *only* way the attacker can compute the needed exponential g^u is to take one (and no more than one!) of the exponentials at his disposal, *i.e.*, some $t_i = g^v$ (at least one such term is available, since $g = g^1$ is the publicly known base) and raise it to a power w such that $(g^v)^w = g^u$, provided that w is derivable from $t_1, \ldots, t_m$. In particular, we may write $w = v^{-1} \cdot u$.

Hence, every time we encounter a constraint of form (9), we nondeterministically choose one of the exponential terms $t_i = g^v$ from the term set available to the attacker, and replace the constraint (9) with

$$t_1, \ldots, t_m \rhd v^{-1} \cdot u. \tag{10}$$

This gives us a constraint sequence $\mathbf{C}'_3$ with no exponential target terms, and we continue with $\mathbf{C}_4$ and solving linear Diophantine equations as in Section 6.

7.1 Pereira-Quisquater example

In [PQ01], Pereira and Quisquater find an attack against key authentication in an authenticated group Diffie-Hellman (A-GDH.2) protocol of [STW96]. In this section, we sketch how to discover this attack using our approach.

The attack involves two key distribution sessions. The first session has four parties, one of which is compromised. The second session occurs among three of those parties after the compromised party has been removed from the group. The attacker, who knows the secret shared keys of the compromised party, causes one of the legitimate parties to accept a non-authentic, compromised key in the second session.

In a decidable finite-session context, the analyst chooses how many strands of each role to put in the semibundle. In this example, involving a four-party group of M_1, M_2, M_3 and M_4, it is sufficient to choose two instances of role M_2 and one of M_4. The behavior of M_1 is ignored (it does not matter for the purposes of discovering the attack), M_2 is the party being attacked, and M_3 is the compromised party, whose role is taken over by the attacker.

The object of the protocol is to construct a group key of the form $g^{r_1 r_2 r_3 r_4}$ where g is a constant and the r_i are secret random contributions from the group members. In the second session, the group key should be $g^{r'_1 r'_2 r'_4}$, where each r'_i is a new random contribution of the ith group member. In each session, partial exponentials are passed up the chain to the last party, who multicasts a message whose ith component is used by the ith party to construct the common key.

The role of the intermediate party M_2 is this:

$$-\langle g, g^{x_{21}}\rangle \; + \langle g^{r_2}, g^{x_{21}}, g^{x_{21} r_2}\rangle \; - \langle x_{22}, g^{x_{23} k_{24}}, x_{24}\rangle$$

where k_{ij} is a long-term secret key shared by M_i and M_j, and x_{ij} are variables used in the specification of the M_i role (*i.e.*, values that are not known to M_i in advance). M_2 assumes that $g^{x_{23}} = g^{r_1 r_3 r_4}$ and computes the group key as $(g^{x_{23} k_{24}})^{k_{24}^{-1} r_2}$.

The role of the last party M_4 in the first (four-party) session is this:

$$-\langle g^{x_{41}}, g^{x_{42}}, g^{x_{43}}, g^{x_{44}}\rangle \; + \langle g^{x_{41} r_4 k_{14}}, g^{x_{42} r_4 k_{24}}, g^{x_{43} r_4 k_{34}}\rangle$$

Our analysis procedure considers all possible event sequences consistent with the semibundle (there are a finite number of possibilities), including the one identified by [PQ01], which yields the following constraint sequence $\mathbf{C}$. Primed random numbers and variables are those associated with the second session.

$$\begin{array}{rcl}
g, k_{34} & \rhd & \langle g, g^{x_{21}}\rangle \\
'', \langle g^{r_2}, g^{x_{21}}, g^{x_{21} r_2}\rangle & \rhd & \langle g^{x_{41}}, g^{x_{42}}, g^{x_{43}}, g^{x_{44}}\rangle \\
'', \langle g^{x_{41} r_4 k_{14}}, g^{x_{42} r_4 k_{24}}, g^{x_{43} r_4 k_{34}}\rangle & \rhd & \langle g, g^{x'_{21}}\rangle \\
'', \langle g^{r'_2}, g^{x'_{21}}, g^{x'_{21} r'_2}\rangle & \rhd & \langle x'_{22}, g^{x'_{23} k_{24}}, x'_{24}\rangle \\
'' & \rhd & g^{x'_{23} r'_2}
\end{array}$$

The ditto mark $''$ stands for repetition of the source terms from the constraint immediately above. The source terms in the first constraint are constants known to the attacker. The first session would normally generate a constraint for the message received by M_2 from M_4, but this message reception is omitted because it is not necessary for the attack. The last constraint is the security objective: it says that the term $g^{x'_{23} r'_2}$, which is computed and accepted as the secret group key in the second session by M_2, is derivable by the attacker.

$\mathbf{C}_1$ is derived from $\mathbf{C}$ by guessing subterms that will be unified. The only subterms appearing in $\mathbf{C}$ are exponentials. We can identify unifiable exponentials at this stage, but we do not need to because those unifications will occur as a byproduct of solving the multiplication-only constraints generated later.

To derive $\mathbf{C}_2$, we note that all exponential subterms can be derived simply by extracting them from the concatenated messages in which they appear. $\mathbf{C}_3$ will then discard the concatenations and retain only the exponentials. The variables ${x'}_{22}$ and x'_{24} are not used and we have dropped the constraints on them.

$$\begin{array}{rcl}
g, k_{34} & \rhd & g^{x_{21}} \\
'', g^{r_2}, g^{x_{21}}, g^{x_{21} r_2} & \rhd & g^{x_{41}} \\
'' & \rhd & g^{x_{42}} \\
'' & \rhd & g^{x_{43}} \\
'' & \rhd & g^{x_{44}} \\
'', g^{x_{41} r_4 k_{14}}, g^{x_{42} r_4 k_{24}}, g^{x_{43} r_4 k_{34}} & \rhd & g^{x'_{21}} \\
'', g^{r'_2}, g^{x'_{21}}, g^{x'_{21} r'_2} & \rhd & g^{x'_{22} k_{24}} \\
'' & \rhd & g^{x'_{22} r'_2}
\end{array}$$

To obtain $\mathbf{C}'_3$, for each of the target exponentials we find, by exhaustive search, the exponential in the source set from which it is computed. For example, the second constraint is solved by computing $g^{x_{41}}$ from g^{r_2}. With the correct choices, the exponential removal step leads to the following product-only constraints. Exponential terms have been removed from the source sets because exponential terms cannot occur in exponents.

$$\begin{array}{lcl}
k_{34} & \rhd & x_{21} \\
k_{34} & \rhd & r_2^{-1} x_{41} \\
k_{34} & \rhd & r_2^{-1} x_{42} \\
k_{34} & \rhd & r_2^{-1} x_{43} \\
k_{34} & \rhd & r_2^{-1} x_{44} \\
k_{34} & \rhd & x_{43}^{-1} r_4^{-1} k_{34}^{-1} {x'}_{21} \\
k_{34} & \rhd & x_{42}^{-1} r_4^{-1} {x'}_{23} \\
k_{34} & \rhd & {x'}_{21}^{-1} {x'}_{23}
\end{array}$$

The last step is to introduce variables $\hat{x}_{41}$, *etc.* so that each target term contains at most one variable. This leads to substitutions

$$\begin{array}{rcll}
x_{4i} & \mapsto & r_2 \hat{x}_{4i} & \qquad i = 1, 2, 3, 4 \\
x'_{21} & \mapsto & r_2 \hat{x}_{42} r_4 k_{34} \hat{x}'_{21} & \\
x'_{23} & \mapsto & r_2 \hat{x}_{42} r_4 \hat{x}'_{23} &
\end{array}$$

and constraints

$$\begin{array}{lcl}
k_{34} & \rhd & x_{21}, \hat{x}_{4i}, \hat{x}'_{21}, \hat{x}'_{23} \\
k_{34} & \rhd & k_{34}^{-1} \hat{x'}_{21}^{-1} \hat{x}'_{23}
\end{array}$$

In general, we would have to convert these to Diophantine equations by expressing each variable as a product of powers of non-variable subterms. This particular system can be solved by inspection; we can set every variable in the constraints to 1, so that after substitution we have the solution $x_{21} = 1$, $x_{4i} = r_2$ for $i = 1, 2, 3, 4$, $x'_{21} = r_2 r_4 k_{34}$, and $x'_{23} = r_2 r_4$. This solution is slightly simpler than the solution in [PQ01].

8 Conclusions

We have presented a constraint solving technique that reduces the problem of symbolic protocol analysis in the presence of an Abelian group operator to a finite set of systems of quadratic Diophantine equations. Each system has a solution if and only if a certain linear subsystem has a solution. Since linear Diophantine equations are decidable, the problem of symbolic protocol analysis with Abelian groups is thus decidable. The significance of this result is that it enables *fully automated* formal analysis of a wide class of protocols that cannot be analyzed in a free-algebra model.

Results presented in this paper are but the first step towards reducing the gap between formal methods and mathematical proofs typically employed in cryptographic analysis of security protocols. Even though we take into account some mathematical properties of the underlying cryptographic primitives, we are still analyzing an abstract model, and thus possibly missing attacks due to our idealized treatment of cryptography. It would be interesting to know whether the results of this paper, especially the existence of conservative solutions, can be extended to algebraic theories other than Abelian groups, or to richer equational theories that more accurately represent properties of the relevant cryptographic functions. At the same time, recent undecidability results for equational unification [KNW02, KNW03] suggest that the symbolic constraint solving problem is undecidable in the presence of rich equational theories. Therefore, it is very likely that symbolic analysis can be fully automated only for abstract protocol models, or for protocols that employ cryptographic primitives without visible mathematical properties.

References

[AL00] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. 11th International Conference on Concurrency Theory (CONCUR '00)*, volume 1877 of *LNCS*, pages 380–394, 2000.

[AR02] M. Abadi and P. Rogaway. Reconciling two views of cryptography. *J. Cryptology*, 15(2):103–127, 2002.

[BB03] M. Boreale and M. Buscemi. On the symbolic analysis of low-level cryptographic primitives: modular exponentiation and the Diffie-Hellman protocol. In *Proc. Workshop on the Foundations of Computer Security (FCS)*, 2003.

[BDSV03] I.C. Bertolotti, L. Durante, R. Sisto, and A. Valenzano. Introducing commutative and associative operators in cryptographic protocol analysis. In *Proc. 23rd International Conference on Formal Techniques for Networked and Distributed Systems (FORTE '03)*, volume 2767 of *LNCS*, pages 224–239, 2003.

[BMV03] D. Basin, S. Mödersheim, and L. Vigano. Constraint differentiation: a new reduction technique for constraint-based analysis of security protocols. In *Proc. 10th ACM Conference on Computer and Communications Security (CCS '03)*, pages 335–344, 2003.

[Bor01] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proc. 28th International Colloquium on Automata, Languages and Programming (ICALP '01)*, volume 2076 of *LNCS*, pages 667–681, 2001.

[BS01] F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 8, pages 445–532. Elsevier Science, 2001.

[CCM01] H. Comon, V. Cortier, and J.C. Mitchell. Tree automata with one memory, set constraints, and ping-pong protocols. In *Proc. 28th International Colloquium on Automata, Languages and Programming (ICALP '01)*, volume 2076 of *LNCS*, pages 682–693, 2001.

[CD94] E. Contejean and H. Devie. An efficient algorithm for solving systems of Diophantine equations. *Information and Computation*, 113(1):143–172, 1994.

[CE02] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In *Proc. 9th International Static Analysis Symposium (SAS '02)*, volume 2477 of *LNCS*, pages 326–341, 2002.

[CJM00] E.M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with Brutus. *ACM Transactions in Software Engineering Methodology (TOSEM)*, 9(4):443–487, 2000.

[CKRT03a] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. Technical Report IFI-Report 0305, CAU Kiel, 2003.

[CKRT03b] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science (LICS '03)*, pages 261–270, 2003.

[CLS03] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science (LICS '03)*, pages 271–280, 2003.

[DLMS99] N. Durgin, P. Lincoln, J.C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. FLOC Workshop on Formal Methods in Security Protocols*, 1999.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[FA01] M. Fiore and M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 160–173, 2001.

[KNW02] D. Kapur, P. Narendran, and L. Wang. A unification algorithm for analysis of protocols with blinded signatures. Technical Report 02-5, SUNY Albany, 2002.

[KNW03] D. Kapur, P. Narendran, and L. Wang. An e-unification algorithm for analyzing protocols that use modular exponentiation. In *Proc. 14th International Conference on Rewriting Techniques and Applications (RTA '03)*, volume 2706 of *LNCS*, pages 165–179, 2003.

[Lam78] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Comms. ACM*, 21(7):558–565, 1978.

[LC89] P. Lincoln and J. Christian. Adventures in associative-communtative unification. *J. Symbolic Computation*, 8(1-2):217–240, 1989.

[Mil03] J. Millen. On the freedom of decryption. *Information Processing Letters*, 86(6):329–333, 2003.

[MN02] C. Meadows and P. Narendran. A unification algorithm for the group Diffie-Hellman protocol. In *Proc. Workshop of Issues in Theory of Security (WITS)*, 2002.

[MS01] J. Millen and V. Shmatikov. Constraint solving for bounded process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS '01)*, pages 166–175, 2001.

[Pau97] L. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proc. 10th IEEE Computer Security Foundations Workshop*, pages 84–95, 1997.

[Pau98] L. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6(1/2):85–128, 1998.

[PQ01] O. Pereira and J.-J. Quisquater. A security analysis of the Cliques protocols suites. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 73–81, 2001.

[RS98] P. Ryan and S. Schneider. An attack on a recursive authentication protocol: A cautionary tale. *Information Processing Letters*, 65(1):7–10, 1998.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. 14th IEEE Computer Security Foundations Workshop*, pages 174–190, 2001.

[Shm04] V. Shmatikov. Decidable analysis of cryptographic protocols with products and modular exponentiation. In *Proc. 13th European Symposium on Programming (ESOP '04)*, volume 2986 of *LNCS*, pages 355–369, 2004.

[Son99] D. Song. Athena: a new efficient automatic checker for security protocol analysis. In *Proc. 12th IEEE Computer Security Foundations Workshop*, pages 192–202, 1999.

[SS89] M. Schmidt-Schauss. Unification in a combination of arbitrary disjoint equational theories. *J. Symbolic Computation*, 8:51–99, 1989.

[STW96] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *Proc. 3rd ACM Conference on Computer and Communications Security (CCS '96)*, pages 31–37, 1996.

[THG99] F. Thayer, J. Herzog, and J. Guttman. Strand spaces: proving security protocols correct? *J. Computer Security*, 7(1), 1999.

A Checking whether a constraint sequence is well-defined

Our “well-definedness” condition on constraint sequences formalized in Definition 3.5 quantifies over all possible substitutions, and may be difficult to check directly. In this section, we give a finite procedure for checking whether a constraint sequence is well-defined. We start by considering substitutions which may cause a variable to disappear from a term without explicitly substituting it. For example, substitution $\theta = [y \mapsto x^{-1}]$ causes x to disappear from $(x \cdot y)\theta$, even though θ does not substitute x. Recall that $\mathcal{D}(\theta)$ is the domain of substitution θ.

Lemma A.1 *Suppose $x \in \mathrm{Var}(t)$ in some term t, and let θ be some substitution such that $x \notin \mathcal{D}(\theta)$. If $x \notin \mathrm{Var}(t\theta)$, then, for every occurrence of x in t, there exists a subterm v such that $\ldots t_x \cdot \ldots \cdot v \ldots \in \mathrm{St}(t)$, the occurence of x is contained in t_x, and $t_x\theta = (v\theta)^{-1}$.*

Proof: The only rule in Fig. 2 that can cause a variable to disappear is the cancellation rule for the Abelian group operator: $t \cdot t^{-1} \rightarrow \mathbf{1}$. Therefore, for every occurrence of x in t, there must be a superterm t_x such that $\ldots t_x \cdot \ldots \cdot v \ldots \in \mathrm{St}(t)$ for some v, and $t_x\theta \cdot v\theta \rightarrow \mathbf{1}$. Therefore, $t_x\theta = (v\theta)^{-1}$. □

Lemma A.1 says that a variable x can disappear from a term when *another* variable is substituted only if every occurrence of x is contained in a subterm t_x which is multiplied with another subterm v which the substitution turns into the inverse of t_x. We will call such substitutions *eliminators* of x.

In the rest of this section, we will use $x^{(i)}$ notation to distinguish different occurrences of x. Recall from Section 2.3 that terms t_1, t_2 have a set of most general unifiers $MGU_{\mathrm{AGF}}(t_1, t_2)$ (if the terms cannot be unified, the set is empty).

Definition A.2 (Eliminators) *Let t be a term such that $x \in \mathrm{Var}(t)$. For each occurrence $x^{(i)}$ in t, define the set of* eliminators $\mathcal{E}(x^{(i)}, t) = \{\theta \mid x \notin \mathcal{D}(\theta)$ *and for some* t_x, v, $\theta \in MGU_{\mathrm{AGF}}(t_x, v^{-1}))\}$, *where* $\ldots t_x \cdot \ldots \cdot v \ldots \in \mathrm{St}(t)$, *and the occurrence $x^{(i)}$ is contained in t_x.*

Observe that for any term t and any occurrence $x^{(i)}$, the set $\mathcal{E}(x^{(i)}, t)$ is finite. There are only a finite number of possibilities for subterms t_x containing this occurrence of x, each t_x is multiplied with at most a finite number of terms v, and unification produces a finite number of most general unifiers (see Section 2.3), of which the subset that does not substitute x is selected.

For example, consider the following term: $t = \langle\langle x^{(1)}, a\rangle, \{x^{(2)} \cdot z\}_k \cdot \{y\}_z{}^{-1}\rangle$. For the first occurrence of x, there is no superterm t_x which is multiplied with another term, therefore, $\mathcal{E}(x^{(1)}, t) = \emptyset$. For the second occurrence x, there are two superterms which are multiplied with another term. The first superterm is x itself, which is multiplied with z. The unifier which does not substitute x but will cause x to disappear is $[z \mapsto x^{-1}]$. The second superterm is $\{x^{(2)} \cdot z\}_k$, which is multiplied with $\{y\}_z{}^{-1}$. The unifier which will cause x to disappear is $[y \mapsto x \cdot k, z \mapsto k]$. Therefore, $\mathcal{E}(x^{(2)}, t) = \{[z \mapsto x^{-1}], [y \mapsto x \cdot k, z \mapsto k]\}$.

We now describe how to merge two substitutions π_1 and π_2, producing a finite set of their most general common instances.

Definition A.3 *Let $\mathcal{D}(\pi_1) \cup \mathcal{D}(\pi_2) = \{x_1, \ldots, x_k\}$. Define*

$$\begin{aligned} merge(\pi_1, \pi_2) = MGU_{\mathrm{AGF}}(\quad & \langle \pi_1(x_1), \langle \pi_1(x_2), \langle \ldots, \pi_1(x_k) \rangle \ldots \rangle \rangle, \\ & \langle \pi_2(x_1), \langle \pi_2(x_2), \langle \ldots, \pi_2(x_k) \rangle \ldots \rangle \rangle \ ) \end{aligned}$$

Of course, substitutions π_1 and π_2 may disagree on some variables, in which case $merge(\pi_1, \pi_2) = \emptyset$. We now extend Definition A.3 to any finite number of substitutions in the obvious way.

Definition A.4 *Define $merge{*}(\Pi)$ inductively as follows:*
- For $\Pi = \{\pi_1, \pi_2\}$, $merge{*}(\Pi) = merge(\pi_1, \pi_2)$;
- For $|\Pi| > 1$ and $\pi \notin \Pi$, $merge{*}(\{\pi\} \cup \Pi) = \bigcup_i \{merge(\pi, \pi_i) \mid \pi_i \in merge{*}(\Pi)\}$.

This definition enables us to extend the notion of eliminators to entire term sets rather than single occurrences.

Definition A.5 *Let T be a set of terms such that $x \in \mathrm{Var}(T)$, let $x^{(1)}, \ldots, x^{(k)}$ be all occurrences of x in T, and let $t_1, \ldots, t_k \in T$ be the respective terms in which x occurs (since x may occur in a term more than once, it is possible that $t_i = t_j$ for some $i \neq j$). Define $\mathcal{E}(x, T) = \bigcup_{\forall \Pi} \{merge{*}(\Pi) \mid (\forall i) |\Pi \cap \mathcal{E}(x^{(i)}, t_i)| = 1\}$.*

By construction, $\mathcal{E}(x, T)$ is a finite set of substitutions that eliminate *all* occurrences of x from term set T, but don't substitute x explicitly. Each element of $\mathcal{E}(x, T)$ is the result of merging a set of substitutions with one representative from each $\mathcal{E}(x^{(i)}, t_i)$.

Of course, for some term sets $\mathcal{E}(x, T)$ may be empty. In particular, if there is even one occurrence $x^{(i)}$ such that $\mathcal{E}(x^{(i)}, t) = \emptyset$ for some $t \in T$ (*e.g.*, if x is not contained within any products), then $\mathcal{E}(x, T) = \emptyset$ because there is no most general common substitution produced by $merge{*}$.

The set $\mathcal{E}(x, T)$ is complete in the following sense.

Lemma A.6 *Suppose $x \in \mathrm{Var}(T)$ for some set of terms T. If $x \notin \mathcal{D}(\theta)$, and $x \notin \mathrm{Var}(T\theta)$, then there exists a partial substitution $\pi \in \mathcal{E}(x, T)$ such that $\theta = \pi \circ \pi'$ for some π'.*

Proof: According to Lemma A.1, *every* occurrence $x^{(i)}$ of variable x must be contained within some subterm t_x such that $\ldots t_x \cdot \ldots \cdot v \ldots \in \mathrm{St}(t)$ where $t \in T$, and $t_x\theta = (v\theta)^{-1}$. Therefore, θ is an instance of one of the most general unifiers of t_x and v^{-1}, or, more precisely, for every occurrence $x^{(i)}$, $\theta = \pi_i \circ \pi'_i$ for some $\pi_i \in MGU_{\mathrm{AGF}}(t_x, v^{-1})$. By Definition A.2, this implies that $\pi_i \in \mathcal{E}(x^{(i)}, t)$ for all i. Therefore, θ must be compatible with at least one π_i from each set $\mathcal{E}(x^{(i)}, t)$. By Definition A.5, this means that $\theta = \pi \circ \pi'$ for some $\pi \in \mathcal{E}(x, T)$. □

We are now ready to prove that it is possible to check whether a constraint sequence satisfies Definition 3.5 by considering only a finite number of substitutions.

Lemma A.7 *Let $\mathbf{C}$ be the constraint sequence generated from a protocol, and suppose there exists a substitution θ such that $\mathbf{C}\theta$ does not satisfy the property in Definition 3.5, or, more precisely, there exists a variable x and constraint $T_i \rhd u_i \in \mathbf{C}$ such that $x \in \mathrm{Var}((T_i \backslash T_{i-1})\theta)$, but $x \notin \bigcup_{j<i} \mathrm{Var}(u_j\theta)$. Then $\theta = \pi \circ \pi'$ for some $\pi \in \mathcal{E}(x, \{u_j | j < i\})$.*

Proof: First, observe that $x \notin \mathcal{D}(\theta)$ because $x \in \mathrm{Var}((T_i \setminus T_{i-1})\theta)$. Then note that $x \in \bigcup_{j<i} \mathrm{Var}(u_j)$ because $\mathbf{C}$ satisfies the origination property, but $x \notin \bigcup_{j<i} \mathrm{Var}(u_j\theta)$. By Lemma A.6, this means that $\theta = \pi \circ \pi'$ for some $\pi \in \mathcal{E}(x, \{u_j | j < i\})$. □

Theorem A.8 *Suppose* $\mathbf{C}$ *is a constraint sequence generated from a protocol. Checking whether* $\mathbf{C}$ *is well-defined according to Definition 3.5 is decidable.*

Proof: We construct a decision procedure as follows. For every variable $x \in \mathrm{Var}(\mathbf{C})$, every constraint $T_i \rhd u_i \in \mathbf{C}$ such that $x \in \mathrm{Var}(T_i \setminus T_{i-1})$, compute $\mathcal{E}(x, \{u_j | j < i\})$. Then, for every $\pi \in \mathcal{E}(x, \{u_j | j < i\})$, check whether $\mathbf{C}\pi$ satisfies the property in Definition 3.5, that is, check whether $\forall i \;\; \mathrm{Var}((T_i \setminus T_{i-1})\pi) \subseteq \bigcup_{j<i} \mathrm{Var}(u_j\pi)$.

If there exists a variable x, index i, and substitution π such that the property is violated, then $\mathbf{C}$ is *not* well-defined. Set $\theta = \pi$.

If the property is satisfied for all x, i and π, then $\mathbf{C}$ is well-defined. We prove this by contradiction. Suppose $\mathbf{C}$ is not well-defined. Then there exist x, i and θ such that $x \in \mathrm{Var}((T_i \setminus T_{i-1})\theta)$, but $x \notin \bigcup_{j<i} \mathrm{Var}(u_j\theta)$. By Lemma A.7, $\theta = \pi \circ \pi'$ for some $\pi \in \mathcal{E}(x, \{u_j | j < i\})$. By definition of $\mathcal{E}(x, \{u_j | j < i\})$, $x \notin \bigcup_{j<i} \mathrm{Var}(u_j\pi)$. Because θ is a refinement of π and $x \in \mathrm{Var}((T_i \setminus T_{i-1})\theta)$, $x \in \mathrm{Var}((T_i \setminus T_{i-1})\pi)$. This contradicts our assumption that $\forall x, i, \pi \in \mathcal{E}(x, \{u_j | j < i\}) \; \mathrm{Var}((T_i \setminus T_{i-1})\pi) \subseteq \bigcup_{j<i} \mathrm{Var}(u_j\pi)$. This concludes the proof. □