Decidability of context-explicit security protocols

R. Ramanujam
The Institute of Mathematical Sciences
C.I.T. Campus, Chennai 600 113, India.
E-mail: jam@imsc.res.in

S. P. Suresh*
Chennai Mathematical Institute
92 G.N.Chetty Road, T.Nagar, Chennai 600 017, India.
E-mail: spsuresh@cmi.ac.in

Abstract

An important problem in the analysis of security protocols is that of checking whether a protocol preserves *secrecy*, i.e., no secret owned by the honest agents is unintentionally revealed to the intruder. This problem has been proved to be undecidable in several settings. In particular, [DLMS99] prove the undecidability of the secrecy problem in the presence of an unbounded set of nonces, even when the message length is bounded. In this paper we prove that even in the presence of an unbounded set of nonces the secrecy problem is decidable for a reasonable subclass of protocols, which we call *context-explicit protocols*.

1 Introduction

Security protocols are specifications of communication patterns which are intended to let agents share secrets over a public network. They are required to perform correctly even in the presence of malicious intruders who listen to the message exchanges that happen over the network and also manipulate the system (by blocking or forging messages, for instance). An obvious correctness requirement is that of secrecy: an intruder cannot read the contents of a message intended for others.

The presence of intruders necessitates the use of encrypted communication. It has been widely acknowledged that even if the cryptographic tools are perfect, desired security goals may not be met, due to logical flaws in the design of protocols. (See [AN95] for an illuminating account.) To deal with this, many approaches have been proposed, one among which is the

*Work done while at The Institute of Mathematical Sciences.

development of *systematic methods for finding flaws* in security protocols. This is complicated by the fact that when we model security protocols precisely, we get infinite state systems. As such, it is to be expected that it is not possible to verify even simple properties like secrecy of such systems. It has been formally proved in ([DLMS99], [HT96], [ALV02]) that in fact, the secrecy problem is undecidable.

This leads us to examine the sources of undecidability and look for interesting subclasses of protocols for which the secrecy problem is decidable. Firstly, security protocols are informally presented as sequences of communications of the form $A \rightarrow B : t$, where A and B are abstract names for principals and t is an abstract term obtained from atomic terms using tupling and encryption. It is easy to see that such a presentation is too general and that some admissibility conditions are needed. We therefore study *well-formed* protocols where we ensure that in every communication the term sent by a principal can be constructed using the messages received by that principal earlier. Most protocols of interest in the literature follow these well-formedness conditions. The notable exceptions are protocols which are used to illustrate the computational power of the model, and which occur in the undecidability proofs. But this is not to say that considering only well-formed protocols leads to decidability of secrecy.

From the specification of a protocol, one can extract a set of roles played by principals participating in the protocol. An event of a protocol Pr is simply an occurrence of an action in an instantiation of a role of Pr. The instantiations of a role of Pr are got by substituting concrete terms for the abstract terms mentioned in the roles. It is clear that there are potentially infinitely many events of a protocol. A run of a protocol is simply a sequence of events of the protocol satisfying certain admissibility conditions. It is important to note that there is no bound on the number of events which can occur in a run.

The secrecy problem is the problem of checking whether a given protocol has a leaky run, where a run is leaky if some information (typically a nonce or a key) was known only to some honest agents at some intermediate point of the run but is known to the intruder at the end of the run. This problem has been proved to be undecidable in [DLMS99]. In [HT96] and [ALV02], it has been shown that this problem is undecidable even if the set of atomic terms is assumed to be fixed and finite, but if terms of arbitrary size can be substituted for nonces.

In the presence of such results, one natural way to proceed is to insist on a fixed finite set of nonces. [DLMS99] shows that in this case the secrecy problem is DEXPTIME-complete. One of the key sources of undecidability is the fact that there is no bound on the number of distinct events that can occur in any run of a protocol. The approach to decidability in [MS01] and [RT03] essentially bounds the number of events that can occur in any run of the protocol, thereby obtaining decidability even in the setting which allows terms of arbitrary size to be substituted for nonces.

The results here are similar to that of [Low99], which introduces a property called *strong secrecy* and proves for a syntactic subclass of protocols (similar to the context-explicit protocols here) that the protocol has a breach of strong secrecy iff a *small system* corresponding to the protocol has a breach of strong secrecy. A small system as defined in [Low99] admits runs in which each role is assigned to an honest agent, who plays at most one session of the role in any run. The intruder is allowed to combine the information in these sessions to send arbitrary messages, though. This result provides a justification for model checking approaches which usually work with such small systems.

[AC02] is another work which is similar in spirit. For every protocol, a "collapsed semantics" is introduced by allowing nonces generated in earlier sessions of roles of the protocol to be confused with each other. This yields a finite system which can be effectively checked for secrecy. A syntactic subclass of protocols is defined for which the collapsed semantics admits a leaky run exactly when the protocol does. A technique called *set based analysis* is also introduced to verify secrecy of the collapsed system.

Independently of our work, [BP03] also shows decidability of secrecy in the presence of unboundedly many nonces and unboundedly long messages. They use *tagging schemes* to obtain termination of the verification algorithm. The protocol and intruder abilities are coded up a set of Horn clauses. The resolution-based algorithm proceeds in two phases: in the first phase resolution is used to complete the given set of clauses, and in the second phase a backward depth-first search is used to determine whether a target formula (representing a breach of secrecy in the protocol) can be derived from the (completed) set of clauses. The tagging scheme is primarily used to prove that the first phase of the algorithm terminates. The scheme used in their paper defines a strictly larger syntactic subclass than that presented in this paper. But there are significant differences between the proof ideas employed here and in [BP03].

An analysis of the undecidability proofs [RS03d] shows that the protocols used in them (coding up two-counter machines) have the following important structural property:

- Different messages in the protocol specification have encrypted subterms which can be unified.

The intruder crucially uses this property to transfer information from one play to another, and 'pump' this process (using either unboundedly many nonces or non-atomic instantiations) to generate unboundedly many plays with *distinct information content*, leading to undecidability.

This suggests that it is desirable to avoid unification of the kind mentioned above. We formalise a way to do this in our definition of context-explicit protocols. These are essentially protocols such that "distinct" encrypted subterms occurring in their specifications are not unifiable. We go on to show that the secrecy problem is decidable for context-explicit protocols. The key to the decidability argument is to show the following property of runs of a context-explicit protocol:

- Whenever the intruder learns information from a message t and uses it to construct a message t' (where t and t' are instances of distinct communications in the protocol specification), then t' can be constructed from just the nonces and keys learnt by the intruder on receiving t.

The restriction on unifiability of encrypted subterms is crucially used in proving this. Interestingly, the crucial steps of the proof of the above property and others which lead to our main decidability result can be stated as properties of analz-proofs and synth-proofs, which formalize how messages are generated and received terms are analyzed.

The technique used here should be contrasted with approaches which impose restrictions on the use of the tupling operator ([ALV02], [DEK82]), or use more stringent admissibility criteria like [CCM01] which uses techniques from tree automata theory to show decidability for the class of protocols in which every agent copies at most one piece of any message it receives into any message it sends.

The work presented here is an expanded version of [RS03a]. In related work, we prove the secrecy problem to be decidable when considering runs based on a fixed finite set of atomic terms but with non-atomic substitutions [RS03b]. In [RS03c] we extend the results of this paper to show that the secrecy problem is decidable for context-explicit protocols in the presence of both unboundedly many nonces and unboundedly long messages. A detailed presentation of all these results as also the undecidability results can be found in the thesis [Sur03].

2 Security protocol modelling

In this section we present the model in which we formulate and prove the main result of the paper. The treatment of cryptography and messages in the model closely follows Dolev and Yao [DY83].

Basic terms

We assume a (potentially infinite) set of *agents* Ag with a special *intruder* $I \in Ag$. The set of *honest agents*, denoted Ho, is defined to be $Ag \setminus \{I\}$. The intruder model which we develop in this section is one which is widely used in the literature. We assume an all-powerful intruder, who can copy every communication in the system, can block any message and can pretend to be any agent. It is assumed that the intruder has unlimited computational resources and can keep a record of every public system event and utilize it at an arbitrarily later time. However, the intruder cannot generate an honest agent's secret autonomously, nor can it break encryption.

Some variations to the above model have been tried but they do not significantly extend the intruder's powers. For example, we might consider a group of colluding intruders rather than a single intruder. But it has been shown in the work [CMS00] that many Dolev-Yao intruders colluding with one another cannot cause more attacks than a single intruder acting alone.

We assume that the set of *keys* K is given by $K_0 \cup K_1$ where K_0 is a countable set and $K_1 \stackrel{\text{def}}{=} \{k_{AB}, pubk_A, privk_A \mid A, B \in Ag, A \neq B\}$. $pubk_A$ is A's *public key* and $privk_A$ is its *private key*. k_{AB} is the *(long-term) shared key* of A and B. For $k \in K$, $\overline{k}$, the *inverse key* of k, is defined as follows: $\overline{pubk_A} = privk_A$ and $\overline{privk_A} = pubk_A$ for all $A \in Ag$, and $\overline{k} = k$ for all the other keys. For every agent A, the set of keys which are assumed to be always known to A, denoted K_A, is defined to be $\{k_{AB}, k_{BA}, pubk_A, privk_A, pubk_B \mid B \in Ag, B \neq A\}$. We also assume a countable set of *nonces* N. ('Nonce' stands for "*n*umber *once* used"). We also assume a perfect nonce generation mechanism which can generate a *nonguessable*, *unique* nonce on each invocation. $\mathcal{T}_0$, the set of *basic terms*, is defined to be $K \cup N \cup Ag$. The set $K_0 \cup N \cup Ag$ will also play a special role in the subsequent development. We use the notation $\mathfrak{T}_0$ to denote it.

Further we fix the nonce n_0 and the key $\mathsf{k}_0 \in K_0$ for the whole discourse. They will essentially play the role of the intruder's initial knowledge, as will be explained later.

Terms

The set of *information terms* is defined to be

$$\mathcal{T} \quad ::= \quad m \mid (t_1, t_2) \mid \{t\}_k$$

where m ranges over $\mathcal{T}_0$ and k ranges over K. These are the terms used in the message exchanges below. The term $\{t\}_k$ is an abstract notation where we make no cryptographic assumptions about the algorithm used to form $\{t\}_k$ from t and k. It could stand for t encrypted with the key k, or it could also stand for t appended with a *signature* using the key k. Following the lead of Dolev and Yao [DY83] we make the perfect encryption assumption. This means that a message encrypted with key k can be decrypted only by an agent who has the corresponding inverse $\overline{k}$. We thus abstract away cryptographic concerns and can treat encryption and decryption as symbolic operators. We also abstract away the real-life phenomenon in which some honest agents lose their long-term keys, unlike [Pau98] which models this explicitly using the notion of an Oops event.

We have also assumed, in keeping with [DY83], that the terms which are communicated in message exchanges come from a free algebra of terms with tupling and encryption operators. This means that we are operating on a space of symbolic terms. This view of terms abstracts away the fact that in the underlying system all messages are bit strings. Thus we sometimes refer to some terms as being of bounded length, even in systems which work with an infinite set of nonces and keys.

Another key point to note is that we work only with *atomic keys*. Working with constructed keys instead would allow us to model more real-life protocols, but some of our results on message generation would fail to hold in a framework which allows constructed keys.

The notion of subterm of a term is the standard one — $ST(m) = \{m\}$ for $m \in \mathcal{T}_0$; $ST((t_1, t_2)) = \{(t_1, t_2)\} \cup ST(t_1) \cup ST(t_2)$; and $ST(\{t\}_k) = \{\{t\}_k, k\} \cup ST(t)$. t' is an *encrypted subterm* of t if $t' \in ST(t)$ and t' is of the form $\{t''\}_k$. $EST(t)$ denotes the set of encrypted subterms of t. The *size* of terms is inductively defined as follows: $|m| = 1$ for $m \in \mathcal{T}_0$; $|(t_1, t_2)| = |t_1| + |t_2| + 1$; and $|\{t\}_k| = |t| + 2$.

Actions

An *action* is either a *send action* of the form $A!B{:}\,(M)t$ or a *receive action* of the form $A?B{:}\,t$ where: $A \in Ho, B \in Ag$ and $A \neq B$; $t \in \mathcal{T}$ is well-formed; and $M \subseteq ST(t) \cap (N \cup K_0)$. For simplicity of notation, we write $A!B{:}\,t$ instead of $A!B{:}\,(\emptyset)\ t$. The set of all actions is denoted by Ac, the set of all send actions is denoted by $Send$, and the set of all receive actions is denoted by Rec.

The agent B is (merely) the intended receiver in $A!B{:}(M)t$ and the purported sender in $A?B{:}t$. As we will see later when the semantics of protocols is elaborated, every send action is an instantaneous receive by the intruder, and similarly, every receive action is an instantaneous send by the intruder.

For $a = A!B{:}(M)t$, $term(a) \stackrel{\text{def}}{=} t$ and $NT(a) \stackrel{\text{def}}{=} M$ and for $a = A?B{:}t$, $term(a) \stackrel{\text{def}}{=} t$ and $NT(a) \stackrel{\text{def}}{=} \emptyset$. $NT(a)$ stands for the *new terms* generated during action a. The notation is appropriately extended so that we can talk of $terms(\eta)$, $NT(\eta)$ and $Actions(\eta)$, for $\eta \in Ac^*$. We also use the notations $ST(a)$ and $EST(a)$ with the obvious meanings.

Ac_A, the set of A-*actions* is given by $\{C!D{:}(M)t, C?D{:}t \in Ac \mid C = A\}$. For any $\eta \in Ac^*$ and $A \in Ag$, $\eta{\restriction}A$, A's view of η, is defined to be the subsequence obtained by projecting η to actions in Ac_A.

Protocol specifications

Definition 2.1 *A* **protocol** *is a pair* $\mathsf{Pr} = (\mathsf{C}, \mathsf{R})$ *where*

- C*, the set of* constants *of* Pr*, denoted* $\mathsf{CT}(\mathsf{Pr})$*, is a subset of* $\mathcal{T}_0$*,*
- R*, the set of* roles *of* Pr*, denoted* $Roles(\mathsf{Pr})$*, is a finite nonempty subset of* Ac^+ *such that for all* $\eta \in \mathsf{R}$*, there is some* $A \in Ho$ *such that* $\eta \in Ac_A^+$*, and*
- $\mathsf{C} \cap NT(\mathsf{R}) = \emptyset$*.*

Definition 2.2 *An* information state s *is a tuple* $(s_A)_{A \in Ag}$ *where* $s_A \subseteq \mathcal{T}$ *for each agent* A*.* $\mathcal{S}$ *denotes the set of all information states. For a state* s*, we define* $ST(s)$ *to be* $\bigcup\limits_{A \in Ag} ST(s_A)$*.*

Definition 2.3 *Given a protocol* $\mathsf{Pr} = (\mathsf{C}, \mathsf{R})$*,* $\mathsf{init}(\mathsf{Pr})$*, the* initial state of Pr *is defined to be* $(T_A)_{A \in Ag}$ *where for all* $A \in Ho$*,* $T_A = \mathsf{C} \cup K_A$ *and* $T_I = \mathsf{C} \cup K_I \cup \{\mathsf{n}_0, \mathsf{k}_0\}$*.*

We do not explicitly model intruder actions in our model. Therefore we also do not explicitly model the phenomenon of the intruder generating new nonces in the course of a run, as is done in some other models (for instance, [DLMS99]). An alternative would be to provide an arbitrary set of nonces and keys to the intruder in the initial state. We follow the approach of just providing the intruder with the fixed nonce n_0 and the fixed key k_0 in the initial state. They serve as symbolic names for the set of new data the intruder might generate in the course of a run. This suffices for the analysis we perform in our proofs later. We will ensure as we develop the model that n_0 and k_0 are not generated as a fresh term by any honest agent in the course of a run of Pr.

Example 2.4 A version of the Needham-Schroeder protocol ([NS78]) is presented in this example. The protocol $\mathsf{Pr}_{\mathsf{NS}}$ is given by (C, R) where $\mathsf{C} = \emptyset$, and $\mathsf{R} = \{\eta_1, \eta_2\}$ where η_1 is the following sequence of actions:

$$
\begin{array}{lllllll}
1. & A & ! & B & : & (x) & \{A,x\}_{pubk_B} \\
2. & A & ? & B & : & & \{x,y\}_{pubk_A} \\
3. & A & ! & B & : & & \{y\}_{pubk_B}
\end{array}
$$

and η_2 is the following sequence of actions:

$$
\begin{array}{lllllll}
1. & B & ? & A & : & & \{A,x\}_{pubk_B} \\
2. & B & ! & A & : & (y) & \{x,y\}_{pubk_A} \\
3. & B & ? & A & : & & \{y\}_{pubk_B}
\end{array}
$$

The protocol has two roles: we call η_1 the initiator role and η_2 the responder role. A sends the new nonce x to B as a challenge to prove his (B's) identity. She then receives a response to it as also a challenge from B in the form of a nonce y. She finally responds to B's challenge by sending back y. Since only B can decrypt the contents of the first message, A is at least convinced that B is alive. Similarly, B first receives a challenge from A and responds to it while issuing his own challenge. He finally receives the response to his challenge. Since only A could have decrypted the contents of the message sent by B, the latter is at least convinced that A is alive. □

The protocol is abstractly presented as a finite set of roles which uses names from an *abstract name space*. But it is intended to represent the potentially infinite set of its runs which use an infinite set of *concrete names*. Runs are obtained by combining several instantiations of the roles of the protocols, where each instantiation is got by substituting appropriate concrete names for the abstract names.

Example 2.5 An example run of $\mathsf{Pr}_{\mathsf{NS}}$ is ξ_1, given below:

$$
\begin{array}{llllll}
(\eta_1,\sigma_1,1) & C & ! & I & : & (m) \quad \{A,m\}_{pubk_I} \\
(\eta_2,\sigma_2,1) & D & ? & C & : & \{A,m\}_{pubk_D} \\
(\eta_2,\sigma_2,2) & D & ! & C & : & (n) \quad \{m,n\}_{pubk_C} \\
(\eta_1,\sigma_1,2) & C & ? & I & : & \{m,n\}_{pubk_C} \\
(\eta_1,\sigma_1,3) & C & ! & I & : & \{n\}_{pubk_I} \\
(\eta_2,\sigma_2,3) & D & ? & C & : & \{n\}_{pubk_D}
\end{array}
$$

Here m and n are distinct concrete nonces, and C and D are concrete agent names. I is the intruder. σ_1 is a substitution such that $\sigma_1(x) = m$, $\sigma_1(y) = n$, $\sigma_1(A) = C$ and $\sigma_1(B) = I$. σ_2 is a substitution such that $\sigma_2(x) = m$, $\sigma_2(y) = n$, $\sigma_2(A) = C$ and $\sigma_2(B) = D$. (η_i, σ_j, k) is the occurrence of the kth action of the role η_i under the instantiation σ_j.

Incidentally, the above run illustrates Lowe's famous attack on the Needham-Schroeder protocol. The intruder fools D into thinking she is talking to C, and uses a parallel session with C in his (intruder's) own identity to help him respond properly to D. □

The rest of the section is devoted to defining the semantics of protocols, as outlined above. But we work only with abstract names for ease of presentation. We use renamings of abstract names instead of substitutions from abstract names to concrete names. It can be checked that all the results go through even if we change the formalism to introduce concrete names etc.

Substitutions and events of a protocol

A *substitution* σ is a partial map from $\mathcal{T}_0$ to $\mathcal{T}_0$ such that:

- for all $A \in Ag$, if $\sigma(A)$ is defined then it belongs to Ag,
- for all $k \in K_0$, if $\sigma(k)$ is defined then it belongs to K_0, and
- for all $n \in N$, if $\sigma(n)$ is defined then it belongs to N.

Substitutions are extended to terms, sets of terms, actions and sequences of actions in a straightforward manner. We present the interesting cases:

- $\sigma(pubk_A)$ and $\sigma(privk_A)$ are defined only if $\sigma(A)$ is defined, in which case they are defined to be $pubk_{\sigma(A)}$ and $privk_{\sigma(A)}$, respectively.
- $\sigma(k_{AB})$ is defined only if $\sigma(A)$ and $\sigma(B)$ are defined and different from each other, in which case it is defined to be $k_{\sigma(A)\sigma(B)}$.
- $\sigma(A!B{:}(M)t)$ is defined only if $\sigma(A)$, $\sigma(B)$ and $\sigma(t)$ are defined, $\sigma(A) \in Ho$, and $\sigma(A)$ is different from $\sigma(B)$, in which case it is defined to be $\sigma(A)!\sigma(B){:}(\sigma(M))\sigma(t)$.
- $\sigma(A?B{:}t)$ is defined only if $\sigma(A)$, $\sigma(B)$ and $\sigma(t)$ are defined, $\sigma(A) \in Ho$, and $\sigma(A)$ is different from $\sigma(B)$, in which case it is defined to be $\sigma(A)?\sigma(B){:}\sigma(t)$.

A substitution σ is said to be *suitable for an action a* iff $\sigma(a)$ is defined, and *suitable for a sequence of actions* η iff $\sigma(\eta)$ is defined. σ is said to be *suitable for* a protocol Pr if $\sigma(t)$ is defined and equal to t for all constants $t \in \mathsf{CT}(\mathsf{Pr})$.

Note that substitutions are partial maps rather than total maps. This makes sense because when we define instantiations of roles, we only need to provide instantiations for the set of names mentioned in the role.

An *event* is a triple (η, σ, lp) such that $\eta \in Ac^+$, σ is a substitution suitable for η, and $1 \leq lp \leq |\eta|$. The set of all events is denoted *Events*. An event (η, σ, lp) is said to be *well-typed* iff σ is well-typed. For a set $T \subseteq \mathcal{T}_0$, an event (η, σ, lp) is said to be a T*-event* iff σ is a T-substitution. An event $e = (\eta, \sigma, lp)$ is said to be an event of a protocol Pr if $\eta \in Roles(\mathsf{Pr})$ and σ is suitable for Pr. The set of all events of Pr is denoted $Events(\mathsf{Pr})$.

For an event $e = (\eta, \sigma, lp)$ with $\eta = a_1 \cdots a_\ell$, $act(e) \stackrel{\text{def}}{=} \sigma(a_{lp})$. If $lp < |\eta|$ then $(\eta, \sigma, lp) \rightarrow_\ell (\eta, \sigma, lp+1)$. For any event e, $LP(e)$, the *local past* of e, is defined to be the set of all events e' such that $e' \stackrel{+}{\rightarrow}_\ell e$. For any event e, $term(e)$ will be used to denote $term(act(e))$ and similarly for $NT(e)$, $ST(e)$, $EST(e)$, etc. For any sequence $\xi = e_1 \cdots e_k$ of events, $terms(\xi) \stackrel{\text{def}}{=} \bigcup_{1 \leq i \leq k} term(e_i)$. $NT(\xi)$, $ST(\xi)$, $EST(\xi)$ etc. are similarly defined. For any sequence of events $\xi = e_1 \cdots e_k$, $Events(\xi) \stackrel{\text{def}}{=} \{e_1, \ldots, e_k\}$.

$$\frac{}{T \cup \{t\} \vdash t}\ \mathsf{Ax}_a \qquad \frac{}{T \cup \{t\} \vdash t}\ \mathsf{Ax}_s$$

$$\frac{T \vdash (t_1, t_2)}{T \vdash t_i}\ \mathsf{split}_i (i = 1, 2) \qquad \frac{T \vdash t_1 \qquad T \vdash t_2}{T \vdash (t_1, t_2)}\ \mathsf{pair}$$

$$\frac{T \vdash \{t\}_k \qquad T \vdash \overline{k}}{T \vdash t}\ \mathsf{decrypt} \qquad \frac{T \vdash t \qquad T \vdash k}{T \vdash \{t\}_k}\ \mathsf{encrypt}$$

$$\frac{T \vdash \{\{t\}_k\}_{\overline{k}}}{T \vdash t}\ \mathsf{reduce}$$

analz-rules $\qquad$ synth-rules

Figure 1: analz and synth rules.

Message generation rules

Definition 2.6 *A* sequent *is of the form $T \vdash t$ where $T \subseteq \mathcal{T}$ and $t \in \mathcal{T}$.*

An analz*-proof (*synth*-proof) π of $T \vdash t$ is an inverted tree whose nodes are labelled by sequents and connected by one of the* analz*-rules (*synth*-rules) in Figure 1, whose root is labelled $T \vdash t$, and whose leaves are labelled by instances of the* Ax_a *rule (*Ax_s *rule). For a set of terms T,* $\mathsf{analz}(T)$ *(*$\mathsf{synth}(T)$*) is the set of terms t such that there is an* analz*-proof (*synth*-proof) of $T \vdash t$.*

For ease of notation, $\mathsf{synth}(\mathsf{analz}(T))$ *is denoted by $\overline{T}$.*

The analz-rule $\mathsf{decrypt}$ says that if the abstract term $\{t\}_k$ and $\overline{k}$ can be derived from T, then t can also be derived. This could either mean decrypting the encrypted term $\{t\}_k$ using the inverse key $\overline{k}$. It could also mean the verification of a *signed term* $\{t\}_k$ using the corresponding *sign verifier* $\overline{k}$. Thus this is an abstract rule in which depending on the status of k, the concrete algorithm which leads to the derivation of t differs. Similarly, the synth-rule $\mathsf{encrypt}$ could denote either encryption or signing. The rule reduce really says that $\{\{t\}_k\}_{\overline{k}}$ is a different abstract notation which denotes the same term as t. This is again a consequence of the fact that $\{t\}_k$ denotes different cryptographic algorithms — encryption, decryption, signing, verifying signatures, etc.

Example 2.7 Let $T = \{t\}$ where $t = (\{\{(m, n)\}_k\}_{k'}, (\overline{k}, \overline{k'}))$. The analz-proof given in Figure 2 shows that $m \in \mathsf{analz}(T)$. For readability, we denote $\{\{(m, n)\}_k\}_{k'}$ by t_1, $(\overline{k}, \overline{k'})$ by t_2, $\{(m, n)\}_k$ by t_3 and (m, n) by t_4. $\square$

Example 2.8 Let $T = \{m, n, k, k'\}$ and $t = \{\{(m, n)\}_k\}_{k'}$. The synth-proof given in Figure 3 shows that $t \in \mathsf{synth}(T)$. For readability, we denote $\{(m, n)\}_k$ by t_1 and (m, n) by t_2. $\square$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{T \vdash t}\,\mathsf{Ax}_a}{T \vdash t_1}\,\mathsf{split}_1 \qquad \dfrac{\dfrac{\dfrac{}{T \vdash t}\,\mathsf{Ax}_a}{T \vdash t_2}\,\mathsf{split}_2}{T \vdash \overline{k'}}\,\mathsf{split}_2}{T \vdash t_3}\,\mathsf{decrypt} \qquad \dfrac{\dfrac{\dfrac{}{T \vdash t}\,\mathsf{Ax}_a}{T \vdash t_2}\,\mathsf{split}_2}{T \vdash \overline{k}}\,\mathsf{split}_1}{T \vdash t_4}\,\mathsf{decrypt}}{T \vdash m}\,\mathsf{split}_1$$

Figure 2: An example analz-proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{T \vdash m}\,\mathsf{Ax}_s \qquad \dfrac{}{T \vdash n}\,\mathsf{Ax}_s}{T \vdash t_2}\,\mathsf{pair} \qquad \dfrac{}{T \vdash k}\,\mathsf{Ax}_s}{T \vdash t_1}\,\mathsf{encrypt} \qquad \dfrac{}{T \vdash k'}\,\mathsf{Ax}_s}{T \vdash t}\,\mathsf{encrypt}$$

Figure 3: An example synth-proof.

We state some basic properties of the synth and analz operators in the following proposition (see also [Pau98]). The proofs are by a routine induction on proof trees.

Proposition 2.9 *Let $T, T' \subseteq \mathcal{T}$, $t \in \mathcal{T}$ and σ be a substitution. Then the following properties hold:*

$T \subseteq \mathsf{analz}(T)$ *and* $T \subseteq \mathsf{synth}(T)$.
if $T \subseteq T'$ *then* $\mathsf{analz}(T) \subseteq \mathsf{analz}(T')$ *and* $\mathsf{synth}(T) \subseteq \mathsf{synth}(T')$.
$\mathsf{analz}(\mathsf{analz}(T)) = \mathsf{analz}(T)$ *and* $\mathsf{synth}(\mathsf{synth}(T)) = \mathsf{synth}(T)$.
$t \in \mathsf{synth}(T)$ *iff* $t \in \mathsf{synth}(T \cap ST(t))$.
$\sigma(\mathsf{analz}(T)) \subseteq \mathsf{analz}(\sigma(T))$ *and* $\sigma(\mathsf{synth}(T)) \subseteq \mathsf{synth}(\sigma(T))$.

It immediately follows from the above proposition that $\overline{T} = \mathsf{synth}(\mathsf{analz}(T))$ is closed under synth. The following proposition says that it is closed under analz as well, thus immediately implying the important statement that $\overline{\overline{T}} = \overline{T}$ for all sets of terms T. It is worth noting that this result depends on the fact that only atomic keys are allowed in the framework.

Proposition 2.10 *For all $T \subseteq \mathcal{T}$, $\mathsf{analz}(\overline{T}) = \overline{T}$.*

Proof: The inclusion from right to left is trivial. We prove the other inclusion now. Suppose $t \in \mathsf{analz}(\overline{T})$. Suppose π is an analz-proof of $\overline{T} \vdash t$. We prove by structural induction that for every subproof ϖ of π with root labelled $\overline{T} \vdash r$, $r \in \overline{T}$. From this it follows that $t \in \overline{T}$ as well.

Suppose ϖ is a subproof of π with root labelled $\overline{T} \vdash r$ such that for all proper subproofs ϖ_1 of ϖ with root labelled $\overline{T} \vdash r_1$, $r_1 \in \overline{T}$. Then we prove that $r \in \overline{T}$ as well. We only consider the case when the rule applied at the root of ϖ is Ax_a or $\mathsf{decrypt}$. The other cases can be similarly handled.

- Suppose ϖ is the following proof:

$$\frac{}{\overline{T} \vdash r}\,\mathsf{Ax}_a$$

Then $r \in \overline{T}$ by definition and we are through.

- Suppose ϖ is the following proof:

$$\frac{\begin{array}{c}(\varpi_1)\\ \vdots \\ \overline{T} \vdash \{r\}_k\end{array} \qquad \begin{array}{c}(\varpi_2)\\ \vdots \\ \overline{T} \vdash \overline{k}\end{array}}{\overline{T} \vdash r}\,\mathsf{decrypt}$$

By induction hypothesis $\{\{r\}_k, \overline{k}\} \subseteq \overline{T}$. From the definition of synth-proofs it follows that for all atomic terms m, if $m \in \overline{T} = \mathsf{synth}(\mathsf{analz}(T))$, then $m \in \mathsf{analz}(T)$. Since $\overline{k}$ is an atomic term, it follows that $\overline{k} \in \mathsf{analz}(T)$. From the fact that $\{r\}_k \in \mathsf{synth}(\mathsf{analz}(T))$, it easily follows that either $\{r\}_k \in \mathsf{analz}(T)$ or $\{r, k\} \subseteq \mathsf{synth}(\mathsf{analz}(T))$. In the first case, since $\overline{k} \in \mathsf{analz}(T)$, it follows that $r \in \mathsf{analz}(T) \subseteq \overline{T}$. In the second case also $r \in \overline{T}$ and we are through.

$\square$

Information states and updates

Definition 2.11 *The notions of an action enabled at a state and update of a state on an action are defined as follows:*

- $A!B{:}(M)t$ *is* enabled *at* s *iff* $t \in \overline{s_A \cup M}$.
- $A?B{:}t$ *is* enabled *at* s *iff* $t \in \overline{s_I}$.
- *update*$(s, A!B{:}(M)t) \stackrel{\text{def}}{=} s'$ *where* $s'_A = s_A \cup M$, $s'_I = s_I \cup \{t\}$, *and for all agents* C *distinct from* A *and* I, $s'_C = s_C$.
- *update*$(s, A?B{:}t) \stackrel{\text{def}}{=} s'$ *where* $s'_A = s_A \cup \{t\}$ *and for all agents* C *distinct from* A, $s'_C = s_C$.

$update(s, \varepsilon) = s$, $update(s, \eta \cdot a) = update(update(s, \eta), a)$.

An aspect worth noting here is that the intruder is acting as an unbounded buffer which synchronises with each send and receive event of the honest agents. In effect the intruder is playing the role of the network as well, but there are some vital differences. The intruder is assumed not to lose any message (even though it might not be sent to the intended recepient). This simplifies much of our analysis since at any point in time, the intruder has all the messages exchanged thus far. In a real-life situation the network (having finite memory) might lose some information and hence our analysis might get more complicated due to consideration of past information.

Definition 2.12 *Given an information state s and a sequence of events $\xi = e_1 \cdots e_k$, $infstate(s, e_1 \cdots e_k)$ is defined to be $update(s, act(e_1) \cdots act(e_k))$. An event e is said to be* enabled at (s, ξ) *iff $LP(e) \subseteq \{e_1, \ldots, e_k\}$ and $act(e)$ is enabled at $infstate(s, \xi)$.*

Given a protocol Pr and a sequence $\xi = e_1 \cdots e_k$ of events of Pr, $infstate_{\mathsf{Pr}}(\xi)$ is defined to be $infstate(\mathsf{init}(\mathsf{Pr}), e_1 \cdots e_k)$. We omit the subscript Pr if the context is clear. An event e of Pr is said to be enabled at a sequence ξ of events of Pr iff e is enabled at $(\mathsf{init}(\mathsf{Pr}), \xi)$.

The following two propositions, which state that if an agent A is not "involved" in an action a then a does not affect A's state, are easy consequences of the definition of update.

Proposition 2.13 *Suppose s is an information state, η is a finite sequence of actions, $A \in Ho$ and $a \notin Ac_A$. Then $(update(s, \eta))_A = (update(s, \eta \cdot a))_A$. As a consequence, for all information states s, all finite sequences of actions η and for all honest agents A, $(update(s, \eta))_A = (update(s, \eta \restriction A))_A$.*

Proposition 2.14 *Suppose s is an information state, η is a finite sequence of actions, and a is a receive action. Then $(update(s, \eta))_I = (update(s, \eta \cdot a))_I$.*

Runs of a protocol

We isolate the sequences of events which can possibly occur as runs of protocols in the following definition. In the next definition, we define the set of runs of a given protocol.

Definition 2.15 *A sequence of events $e_1 \cdots e_k$ is said to be a* run *with respect to an information state s iff:*

- *for all $i : 1 \leq i \leq k$, e_i is enabled at $(s, e_1 \cdots e_{i-1})$,*
- *for all $i : 1 \leq i \leq k$, $NT(e_i) \cap ST(s) = \emptyset$, and for all $i < j \leq k$, $NT(e_i) \cap NT(e_j) = \emptyset$. (This is the* unique origination property *of runs.)*

Definition 2.16 *Given a protocol Pr, a sequence ξ of events of Pr is said to be a run of Pr iff it is a run with respect to $\mathsf{init}(\mathsf{Pr})$. We let $\mathcal{R}(\mathsf{Pr})$ denote the set of all runs of Pr.*

The following is an easy consequence of the definition of runs.

Proposition 2.17 *Suppose $\xi = e_1 \cdots e_k$ is a run with respect to a state s. Then for all $i \leq k$, $NT(e_i) \cap ST(\mathit{infstate}(s, e_1 \cdots e_{i-1})) = \emptyset$.*

Essentially, the admissibility conditions on runs of protocols achieve the same effect as a presentation in which each agent maintains information about the set of sessions he/she is participating in. In the latter, more operational approach, on receiving a message (which belongs to a particular session), some variables associated with that session would get updated.

Our approach is more denotational than operational in that we abstract away from the specifics of the update of agents' information. For us, a run is not just a sequence of actions but a sequence of events. An event is a record of an action occurrence in a particular context. Thus the substitutions which form part of the events uniquely identify the session of which a given action is part of. They also identify the actual terms substituted for all the variables mentioned in the role. Thus the effect of the more operational models is achieved by imposing conditions on runs. The agents' information state represents only the messages the agent possesses and can synthesize, and not any other control information. Thus when an agent receives a message t, it just adds t to its state. There is no need to update any control information.

The secrecy problem

Definition 2.18 *A basic term $m \in \mathcal{T}_0$ is said to be* secret at state s *iff there exists $A \in Ho$ such that $m \in \mathsf{analz}(s_A) \setminus \mathsf{analz}(s_I)$. Given a protocol Pr and $\xi \in \mathcal{R}(\mathsf{Pr})$, m is said to be secret at ξ if it is secret at $\mathit{infstate}(\xi)$. ξ is* leaky *iff there exists a basic term m and a prefix ξ' of ξ such that m is secret at ξ' and not secret at ξ.*

The secrecy problem is the problem of determining for a given protocol Pr whether some run of Pr is leaky.

Thus we say that a run is leaky if some atomic term is secret at some intermediate state of the run but is revealed to the intruder at the end of the run. It is possible that there are protocols for which leaks of the above form do not constitute a breach of security. A more general notion would be to *allow the user to specify certain secrets which should not be leaked* and check for such leaks. In later chapters, we prove the decidability of the secrecy problem (defined above) for a subclass of protocols. It is still not known whether there is a "reasonable" syntactic subclass of protocols for which the more general secrecy problem (which checks for leaks of user-specified secrets) is decidable.

Example 2.19 The run ξ_1 of Example 2.5 is leaky. This is because n is secret at the prefix $\xi_1' = (\eta_1, \sigma_1, 1) \cdot (\eta_2, \sigma_2, 1) \cdot (\eta_2, \sigma_2, 2)$ of ξ_1, whereas it is not secret at ξ_1. □

3 Well-formed protocols

In the literature, protocols are informally presented as a sequence of communications of the form $A \rightarrow B : t$. There are also some other "well-formedness" conditions which are implicitly assumed. In this section, we formalise these criteria and explore their consequences. The main property of well-formed protocols is that for each of their roles and plays, every send action in it is enabled by the previous actions. As a result, when we analyse well-formed protocols, checking enabledness of send actions by honest agents is relatively straightforward. If ξ is a run of a well-formed protocol and e is a send event such that $LP(e) \subseteq Events(\xi)$, then as a consequence of the propositions proved in this section, e is enabled at ξ and hence $\xi \cdot e$ is also a run of the protocol. Thus it suffices to consider mainly the changes in the intruder state while analysing such protocols. This has also been the standard practice in the analysis of security protocols. It can be seen that it is the implicit assumption of well-formedness that justifies this practice.

Well-formed protocols

Definition 3.1 *A sequence of actions $\eta = a_1 \cdots a_\ell$ is said to be* send-admissible *with respect to a state s if for all $i \leq \ell$, if $a_i \in Send$ then a_i is enabled at $update(s, a_1 \cdots a_{i-1})$. η is said to be said-admissible with respect to a protocol* Pr *iff it is send-admissible with respect to* $\mathsf{init}(\mathsf{Pr})$.

For any send action $A!B{:}(M)t$ with $B \in Ho$, we say that $B?A{:}t$ is its *matching receive* action.

Definition 3.2 *A* **well-formed protocol** *is a pair $\mathsf{Pr} = (\mathsf{C}, \eta)$ where:*

- C*, the set of* constants *of* Pr*, denoted* $\mathsf{CT}(\mathsf{Pr})$*, is a subset of $\mathcal{T}_0$, and*
- $\eta = a_1 b_1 \cdots a_\ell b_\ell \in Ac^+$ *such that:*
  - *for all $1 \leq i \leq \ell$, a_i is a send action and b_i is its matching receive,*
  - *η is send-admissible with respect to $(K_A \cup \mathsf{C})_{A \in Ag}$, and*
  - *$NT(\eta) \cap \mathsf{C} = \emptyset$.*

For a well-formed protocol $\mathsf{Pr} = (\mathsf{C}, \eta)$, $Roles(\mathsf{Pr})$, the set of roles *of* Pr*, is defined to be the set $\{\eta{\upharpoonright}A \mid A \in Ag \text{ and } \eta{\upharpoonright}A \neq \varepsilon\}$.*

Despite different styles of presentation, in fact well-formed protocols are closely related to protocols as defined in Definition 2.1.

Proposition 3.3 *For every well-formed protocol $\mathsf{Pr} = (\mathsf{C}, \eta)$, $(\mathsf{C}, Roles(\mathsf{Pr}))$ is a protocol.*

Proposition 3.4 *All roles of a well-formed protocol $\mathsf{Pr} = (\mathsf{C}, \eta)$ are send-admissible with respect to* Pr.

Proof: For simplicity of notation, let s_0 denote $\mathsf{init}(\mathsf{Pr})$. Suppose $\eta = a_1 \cdots a_\ell$ and suppose $\zeta = a_{i_1} \cdots a_{i_r}$ is a role of Pr, i.e., $\zeta = \eta \restriction A$ for some $A \in Ho$. By Proposition 2.13, it is clear that for all $j : 1 \leq j \leq r$, $(update(s_0, a_1 \cdots a_{i_j}))_A = (update(s_0, a_{i_1} \cdots a_{i_j}))_A$. Since Pr is a well-formed protocol, η is send-admissible with respect to Pr. The send-admissibility of ζ now follows from the above equality. □

Proposition 3.5 *Suppose* $\mathsf{Pr} = (\mathsf{C}, \eta)$ *is a well-formed protocol,* ζ *is a role of* Pr *and* σ *is a substitution suitable for* Pr *and* ζ*. Then* $\sigma(\zeta)$ *is send-admissible with respect to* Pr*.*

Proof: For simplicity of notation, let s_0 denote $\mathsf{init}(\mathsf{Pr})$. Note that $\zeta = \eta \restriction A$ for some $A \in Ho$. Since σ is suitable for Pr and ζ, σ is defined on all actions occurring in ζ, and $\sigma(m) = m$ for all $m \in \mathsf{CT}(\mathsf{Pr})$. We first prove for all prefixes ζ' of ζ that $\sigma(s'_A) \subseteq (s'_1)_{\sigma(A)}$ by induction on $|\zeta'|$ (where we denote $update(s_0, \zeta')$ by s' and $update(s_0, \sigma(\zeta'))$ by s'_1):

$\zeta' = \varepsilon$*:* In this case $s' = s'_1 = s_0$. Now it is clear that $\sigma(\mathsf{C}) = \mathsf{C}$ and $\sigma(K_A) = K_{\sigma(A)}$. Since $A \in Ho$, $\sigma((s_0)_A) = \mathsf{C} \cup \sigma(K_A)$. Further $(s_0)_{\sigma(A)} \supseteq \mathsf{C} \cup K_{\sigma(A)}$ (with inequality when $\sigma(A) = I$). It immediately follows that $\sigma(s'_A) \subseteq (s'_1)_{\sigma(A)}$ in this case.

$\zeta' = \zeta'' \cdot a$*:* Note that $\sigma(\zeta') = \sigma(\zeta'') \cdot \sigma(a)$. For simplicity let us denote $update(s_0, \zeta'')$ by s'' and $update(s_0, \sigma(\zeta''))$ by s''_1. We need to prove that $\sigma(s'_A) \subseteq (s'_1)_{\sigma(A)}$ assuming that $\sigma(s''_A) \subseteq (s''_1)_{\sigma(A)}$.

Now if $a = A!B{:}(M)t$ then $s'_A = s''_A \cup M$. Since $\sigma(s''_A) \subseteq (s''_1)_{\sigma_A}$, and since $\sigma(s'_A) = \sigma(s''_A) \cup \sigma(M)$ and $(s'_1)_{\sigma(A)} = (s''_1)_{\sigma_A} \cup \sigma(M)$ (because $\sigma(a) = \sigma(A)!\sigma(B){:}(\sigma(M)\sigma(t))$,it follows that $\sigma(s'_A) \subseteq (s'_1)_{\sigma(A)}$.

The case when $a = A?B{:}t$ is identically handled. This proves the induction case.

From Proposition 3.4 it follows that ζ is send-admissible. Now consider any prefix $\zeta' \cdot a$ of ζ with $a \in Send$. For simplicity let us denote $update(s_0, \zeta')$ by s' and $update(s_0, \sigma(\zeta'))$ by s'_1. We know that $term(a) \in \overline{s'_A \cup NT(a)}$. Therefore $term(\sigma(a))$ is the same as $\sigma(term(a))$, which belongs to $\sigma(\overline{s'_A \cup NT(a)})$. But it follows from Proposition 2.9 that $\sigma(\overline{T}) \subseteq \overline{\sigma(T)}$ for any σ and T. Further $\sigma(s'_A \cup NT(a)) = \sigma(s'_A) \cup NT(\sigma(a))$ and by what has been proved above $\sigma(s'_A) \subseteq (s'_1)_{\sigma(A)}$. Putting all this together we see that $term(\sigma(a)) \in \overline{(s'_1)_{\sigma(A)} \cup NT(\sigma(a))}$. This shows that $\sigma(\zeta)$ is also send-admissible. □

Context-explicit protocols

Definition 3.6 *A well-formed protocol* $\mathsf{Pr} = (\mathsf{C}, \eta)$ *with* $\eta = a_1 b_1 \cdots a_\ell b_\ell$ *is called a* context-explicit protocol *iff:*

- *for all substitutions* σ, σ' *suitable for* Pr*, for all* $i < j \leq \ell$*, and for all* $t \in EST(a_i)$ *and* $t' \in EST(a_j)$*, if* $\sigma(t) = \sigma'(t')$ *then* $t = t'$ *and* $i = j$*.*
- *for all* $i \leq \ell$*, there exists some* $n \in NT(a_i)$ *such that for all* $t \in EST(a_i)$ *there exists* t' *and* k' *such that* $t = \{(n, t')\}_{k'}$*.*

The syntactic restrictions imposed in the above definition are in keeping with the prudent engineering practices for cryptographic protocols advocated in [AN96]. In particular, Principle 1 of [AN96] says the following:

> Every message should say what it means: the interpretation of the message should depend only on its content. It should be possible to write down a straightforward English sentence describing the content — though if there is a suitable formalism available that is good too.

Quoting [AN96] further:

> All the elements of [the] meaning [of a message] should be explicitly represented in the message, so that a recipient can recover the meaning without any context.

We would like to view the restrictions imposed in Definition 3.6 as achieving the above effect by making the context explicit in the message itself. This earns these protocols the name *context-explicit* protocols.

A particular instance of the above principle is that for every encrypted term t' an agent receives in the course of a run, it knows precisely the term t occurring in the protocol specification of which t' is an instance. This is ensured by the first clause in the definition of context-explicit protocols. We might also want that instantiations of t belonging to different events are somehow distinguishable. This is ensured by the second clause in the definition, which attaches a new nonce to every instantiation of a term. A discussion of the reasonableness of these restrictions is provided at the end of the paper.

The following is an important technical consequence of the fact that distinct nonces are appended to the encrypted terms occurring in each send message. It is used in proving bounds on the length of runs which we need to consider when analysing a context-explicit protocol for secrecy.

Proposition 3.7 *Let $\mathsf{Pr} = (\mathsf{C}, a_1 b_1 \cdots a_\ell b_\ell)$ be a context-explicit protocol and let $e_1 \cdots e_r$ be a run of Pr. Then for all receive events $e_k (k \leq r)$, there is at most one send event e_i such that $EST(e_i) \cap EST(e_k) \neq \emptyset$.*

Proof: Suppose $e_1 \cdots e_r$ is a run of Pr and suppose there is a receive event e_k and two send events e_i and e_j (with $i \neq j$) such that neither $EST(e_i)$ nor $EST(e_j)$ is disjoint from $EST(e_k)$. Suppose $t_i \in EST(e_i) \cap EST(e_k)$ and $t_j \in EST(e_j) \cap EST(e_k)$. From the definition of context-explicit protocols it is clear that for all events e of ξ, there exists a nonce n such that for all $t \in EST(e)$, $t = \{(n, u)\}_k$ for some u and k. Further if e is a send event, $n \in NT(e)$. Thus there exist $n_i \in NT(e_i)$ and $n_j \in NT(e_j)$ such that $t_i = \{(n_i, u_i)\}_{k_i}$ and $t_j = \{(n_j, u_j)\}_{k_j}$ for some u_i, u_j, k_i and k_j. Now both t_i and t_j belong to $EST(e_k)$, therefore it follows that $n_i = n_j$. But then $n_i \in NT(e_i) \cap NT(e_j)$, which violates the property of unique origination of nonces. This contradicts the fact that ξ is a run. This contradiction leads us to conclude that there is at most one i such that $EST(e_i) \cap EST(e_k) \neq \emptyset$. □

The main result of this paper is the following theorem, whose proof will constitute the following sections.

Theorem 3.8 *The problem of determining for a given context-explicit protocol* Pr *whether it has a leaky run is decidable.*

4 Decidability for bounded length runs

In this section, we prove the decidability of a restricted secrecy problem — that of checking for a given protocol Pr and a number r whether there is some leaky run of Pr *of length bounded by* r. The trouble is that the set of such runs is still infinite. We show that we can always suitably rename nonces and keys occurring in runs with nonces and keys from a fixed finite set. Since there can only be finitely many runs which can be thus formed, we get the desired decidability result.

Fix a context-explicit protocol $\mathsf{Pr} = (\mathsf{C}, \eta)$ with $\eta = a_1 b_1 \cdots a_\ell b_\ell$ for the rest of the section. For any number r, $\mathcal{R}_r(\mathsf{Pr}) \stackrel{\text{def}}{=} \{\xi \text{ is a run of } \mathsf{Pr} \mid |\xi| \leq r\}$. For any $T \subseteq \mathcal{T}_0$ and any number r, we define $\mathcal{R}_r^T(\mathsf{Pr})$ to be $\{\xi \mid \xi \text{ is a } T\text{-run of } \mathsf{Pr} \text{ of length at most } r\}$.

Suppose we fix a *finite* $T \subseteq \mathcal{T}_0$ and a number r. It is clear that there are only finitely many T-substitutions suitable for Pr. Let there be b_1 such T-substitutions. It now follows that there are at most $b_2 = 2 \cdot \ell \cdot b_1$ T-events. This bound easily follows from the fact that the set of distinct (η, i) pairs where η is a role of Pr and $1 \leq |i| \leq |\eta|$ is $2 \cdot \ell$. This coupled with the number of T-substitutions gives us b_2. From this it easily follows that there are at most $(b_2 + 1)^r$ runs in $\mathcal{R}_r^T(\mathsf{Pr})$. Thus we see that $\mathcal{R}_r^T(\mathsf{Pr})$ is a finite, effectively constructible set, and therefore the problem of checking whether there is a leaky run in $\mathcal{R}_r^T(\mathsf{Pr})$ is decidable.

Below we explain how to define a finite set $T(r)$ for any given number r such that $\mathcal{R}_r(\mathsf{Pr})$ has a leaky run iff $\mathcal{R}_r^{T(r)}(\mathsf{Pr})$ has a leaky run. Suppose w is the maximum size of any term occurring in the specification of Pr, and suppose p is the maximum length of any role of Pr. Given r, fix three sets $NT(r) \subseteq N \setminus \mathsf{C}$, $K_0(r) \subseteq K_0 \setminus \mathsf{C}$ and $Ag(r) \subseteq Ag \setminus \mathsf{C}$ such that $|N(r)| = |K_0(r)| = |Ag(r)| = r \cdot p \cdot (w + 2)$. (The reason for choosing this specific number will become clear as we develop the proof of the following lemma.) $T(r)$ is defined to be $N(r) \cup K_0(r) \cup Ag(r) \cup \mathsf{CT}(\mathsf{Pr})$.

Lemma 4.1 *For any* $r \in \mathbb{N}$*, if* $\mathcal{R}_r(\mathsf{Pr})$ *has a leaky run then so does* $\mathcal{R}_r^{T(r)}(\mathsf{Pr})$*.*

Proof: We first set up some notation which we use locally in this proof: for any action a of the form $A!B{:}(M)t$ or $A?B{:}t$, *parties*(a) (the set of *apparent (not actual) participants* in the action a), is defined to be $\{A, B\}$. For any sequence of actions $\eta = a_1 \cdots a_\ell$, $\mathit{parties}(\eta) = \bigcup_{1 \leq i \leq \ell} \mathit{parties}(a_i)$. Let us define the *domain of* η for any $\eta \in \mathsf{Pr}$ to be $(ST(\eta) \cup \mathit{parties}(\eta)) \cap \mathcal{T}_0$. Note that for all $\eta \in \mathsf{R}$, the domain of η contains at most $p \cdot (w + 2)$ terms. It clearly suffices to consider events of Pr of the form (η, σ, lp) where the domain of σ is restricted to the domain of η. Let us call such events as *domain-restricted events*. A run composed only of domain-restricted events is called a *domain-restricted run*.

Let us define the *range of a run* ξ to be the union of the ranges of all substitutions σ such that $(\eta, \sigma, lp) \in \mathit{Events}(\xi)$ for some η and lp. (Note that by range of a substitution σ, we mean the set $\{\sigma(x) \mid x \in \mathfrak{T}_0 \text{ and } \sigma(x) \text{ is defined}\}$.) If we consider a domain-restricted run ξ of length at most r, then it is clear that the range of ξ has at most $r \cdot p \cdot (w+2)$ terms. Now $T(r)$ contains $r \cdot p \cdot (w+2)$ nonces and the same number of keys and agent names. Therefore there exists at least one *injective substitution* from the range of ξ to $T(r)$.

Fix one such substitution τ_ξ for each such bounded-domain run $\xi \in \mathcal{R}_r(\mathsf{Pr})$. (It is the *renaming map associated with* ξ.) For any such run $\xi = e_1 \cdots e_k$ with $e_i = (\eta_i, \sigma_i, lp_i)$ for each $i \leq k$, define $\tau_\xi(\xi)$ to be the run $\tau_\xi(e_1) \cdots \tau_\xi(e_k)$ where $\tau_\xi(e_i) = (\eta_i, \tau_\xi \circ \sigma_i, lp_i)$ for each $i \leq k$ (for each $x \in \mathfrak{T}_0$, $(\tau_\xi \circ \sigma_i)(x)$ is defined to be $\tau_\xi(\sigma_i(x))$).

Now for every domain-restricted run $\xi \in \mathcal{R}_r(\mathsf{Pr})$, it is a simple matter to check that for any prefix ξ' of ξ, $A \in Ag$ and $t \in \mathcal{T}$, we have $t \in (\mathit{infstate}(\xi'))_A$ iff $\tau_\xi(t) \in (\mathit{infstate}(\tau_\xi(\xi')))_A$. Also t is leaked in ξ iff $\tau_\xi(t)$ is leaked in $\tau_\xi(\xi)$. From this it easily follows that $\tau_\xi(\xi)$ is in fact a run of Pr (and so belongs to $\mathcal{R}_r^{T(r)}(\mathsf{Pr})$) and that it is leaky if and only if ξ is leaky.

Thus we have shown that if there is a leaky run in $\mathcal{R}_r(\mathsf{Pr})$, then there is also a leaky run in $\mathcal{R}_r^{T(r)}(\mathsf{Pr})$. $\square$

From the above discussion we conclude the following:

Theorem 4.2 *The problem of checking for a given protocol* Pr *and a given bound* r *whether there is a well-typed leaky run of* Pr *of length bounded by* r*, is decidable.*

5 Decidability for good runs

In this section, we define the notion of a *good run* and prove some basic properties of good runs. We also prove that the problem of checking whether there is a good leaky run of a given *context-explicit protocol* is decidable.

Good runs formalise a notion of "well-behavedness" of runs. The essential property of good runs is the following:

- Given a context-explicit protocol $\mathsf{Pr} = (\mathsf{C}, \eta)$ and a good run $e_1 \cdots e_k$ of Pr, none of the e_i's is enabled by another event which is an instance of a later communication in η.

Definition 5.1 *Suppose* $\mathsf{Pr} = (\mathsf{C}, \eta)$ *is a context-explicit protocol and* $\xi = e_1 \cdots e_k$ *is a run of* Pr*. For* $i, j \leq k$*,* e_j *is called a* good successor *of* e_i *(and* e_i *a* good predecessor *of* e_j*) in* ξ *iff* $i < j$ *and at least one of the following conditions holds:*

- $e_i \rightarrow_\ell e_j$.
- e_i *is a send event,* e_j *is a receive event, and* $EST(e_i) \cap EST(e_j) \neq \emptyset$.

For $i \leq k$*,* e_i *is called a* good event *in* ξ *iff either* $i = k$ *or there is some* $j > i$ *such that* e_j *is a good successor of* e_i*.* e_i *is called a bad event iff it is not a good event. A run* ξ *is called a* good run *iff all its events are good. A subsequence* $e_1 \cdots e_r$ *of* ξ *is called a* good path *iff for all* $j < r$*,* e_{j+1} *is a good successor of* e_j*.*

The following propositions list some useful properties of good runs.

Proposition 5.2 *Suppose* $\mathsf{Pr} = (\mathsf{C}, a_1 b_1 \cdots a_\ell b_\ell)$ *is a context-explicit protocol and* ξ *is a run of* Pr. *Then all good paths in* ξ *are of length at most* $2 \cdot \ell$.

Proof: For convenience, define the following notation: for all $i : 1 \leq i \leq \ell$, $c_{2\cdot i-1} \stackrel{\text{def}}{=} a_i$ and $c_{2\cdot i} \stackrel{\text{def}}{=} b_i$. Suppose $e_1 \cdots e_r$ is a good path in ξ with $e_i = (\zeta_i, \sigma_i, lp_i)$ for all $i \leq r$. Since for all $j \leq r$, e_j is an event of Pr, it is clear that there exists some $i_j \leq 2 \cdot \ell$ such that $\zeta_j(lp_j) = c_{i_j}$.

We now show that for all $j < r$, $i_j < i_{j+1}$, using the fact that e_{j+1} is a good successor of e_j. There are two cases to consider:

$e_j \rightarrow_\ell e_{j+1}$*:* In this case it is clear that $\zeta_j = \zeta_{j+1}$, $\sigma_j = \sigma_{j+1}$ and $lp_{j+1} = lp_j + 1$. Now ζ_j is a role of Pr and hence a subsequence of $c_1 \cdots c_{2\cdot\ell}$. Thus c_{i_j} occurs earlier in $c_1 \cdots c_{2\cdot\ell}$ than $c_{i_{j+1}}$ and hence $i_j < i_{j+1}$.

$act(e_j) \in Send$*,* $act(e_{j+1}) \in Rec$ *and* $EST(e_j) \cap EST(e_{j+1}) \neq \emptyset$*:* It is clear now that c_{i_j} is a send action and $c_{i_{j+1}}$ is a receive action, and also that $c_{i_{j+1}-1}$ is a send action with $term(c_{i_{j+1}-1}) = term(c_{i_{j+1}})$. Thus it follows that there exist t and t' belonging to $EST(c_{i_j})$ and $EST(c_{i_{j+1}-1})$ respectively, such that $\sigma_j(t) = \sigma_{j+1}(t')$. But from the definition of context-explicit protocols, it follows that $t = t'$ and $i_j = i_{j+1} - 1$. This shows that $i_j < i_{j+1}$.

From this it follows that there is a sequence $i_1 < \cdots < i_r \leq 2 \cdot \ell$ such that for all $j \leq r$, $\zeta_j(lp_j) = c_{i_j}$. This suffices to prove that $r \leq 2 \cdot \ell$. □

Lemma 5.3 *Suppose* $\mathsf{Pr} = (\mathsf{C}, a_1 b_1 \cdots a_\ell b_\ell)$ *is a context-explicit protocol and* ξ *is a good run of* Pr. *Then* $|\xi| \leq 2^{2\cdot\ell+1} - 1$.

Proof: Suppose $\xi = e_1 \cdots e_k$. Since ξ is a good run of Pr, all the events e_i ($i \leq k$) are good. This means that for all $i < k$, there is some $j : i < j \leq k$ such that e_j is a good successor of e_i. It easily follows that for all $i < k$, there is a good path from e_i to e_k. For all $i : 0 \leq i \leq 2 \cdot \ell$, define the set E_i to be the set of events e occurring in ξ such that the shortest good path from e to e_k is of length i. From Proposition 5.2 we know that all good paths of ξ are of length at most $2 \cdot \ell$. Thus the set of events occurring in ξ is partitioned by the sets $E_0, \ldots, E_{2\cdot\ell}$. From Proposition 3.7, we can conclude that for every receive event e occurring in ξ there is at most one send event e' in ξ such that $EST(e) \cap EST(e') \neq \emptyset$. Further for every event e there is at most one e' such that $e' \rightarrow_\ell e$. Thus every event occurring in ξ has at most two good predecessors, and thus for all $i < 2 \cdot \ell$, $|E_{i+1}| \leq 2 \cdot |E_i|$. Thus it is easy to see by induction that for all $i \leq 2 \cdot \ell$, $|E_i| \leq 2^i$, and that $|\xi| \leq 2^{2\cdot\ell+1} - 1$. □

Lemma 5.3 and Theorem 4.2 immediately imply the following theorem.

Theorem 5.4 *The problem of checking for a given context-explicit protocol* Pr *whether there is a* good leaky run *of* Pr *is decidable.*

6 Reduction to good runs

In this section we prove the main result of this paper — if a context-explicit protocol has a leaky run then it has a *good* leaky run. Thus the secrecy problem for context-explicit protocols is reduced to searching whether there is a good leaky run, and an appeal to Theorem 5.4 yields Theorem 3.8.

6.1 How to eliminate terms

Suppose T is a set of terms and u is a term such that $u \in \overline{T}$. Can we remove a term t (with the property that $EST(t) \cap EST(u) = \emptyset$) from T but add a set of *atomic terms* T' such that it is still the case that $u \in \overline{(T \setminus \{t\}) \cup T'}$? The following lemmas show that under some additional assumptions this is possible. They will be crucially used later in the reduction to good runs. We split the task mentioned above into two parts, first handling the case when $u \in \mathsf{analz}(T)$ and then considering what happens when $u \in \overline{T}$.

Lemma 6.1 *Suppose* $T = (\mathsf{analz}(S_1 \cup \{t\}) \setminus \mathsf{analz}(S_1)) \cap \mathcal{T}_0$ *for some* $S_1, S_2 \subseteq \mathcal{T}$ *and* $t \in \mathcal{T}$.

1. *Let* u *be a term and let* π *be an* analz*-proof of* $S_1 \cup S_2 \cup \{t\} \vdash u$ *such that for all* $k \in ST(S_1 \cup \{t\}) \cap K$ *for which* $\overline{k}$ *labels a non-root node of* π, $\overline{k} \in \mathsf{analz}(S_1 \cup \{t\})$. *Then* $u \in (\mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)) \cup \mathsf{analz}(S_1 \cup S_2 \cup T)$.

2. *Let* u *be a term such that* $u \in \mathsf{synth}((\mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)) \cup \mathsf{analz}(S_1 \cup S_2 \cup T))$ *and* $EST(u) \cap EST(t) = \emptyset$. *Then* $u \in \overline{S_1 \cup S_2 \cup T}$.

Proof:

1. Suppose π is an analz-proof of $S_1 \cup S_2 \cup \{t\} \vdash u$. We prove by structural induction that for every subproof ϖ of π with root labelled $S_1 \cup S_2 \cup \{t\} \vdash w$, w belongs to $(\mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)) \cup \mathsf{analz}(S_1 \cup S_2 \cup T)$. Suppose ϖ is a subproof of π with root labelled $S_1 \cup S_2 \cup \{t\} \vdash w$ such that for all proper subproofs ϖ_1 of ϖ the statement of the lemma holds. Then we prove that it holds for ϖ as well. We only consider the cases when the rule applied at the root of ϖ is Ax_a or $\mathsf{decrypt}$. The other cases can be handled by a routine application of the induction hypothesis.

   - Suppose ϖ is the following proof:

     $$\frac{}{S_1 \cup S_2 \cup \{t\} \vdash w}\,\mathsf{Ax}_a$$

     Then $w \in S_1 \cup S_2 \cup \{t\}$. If $w = t$ then $w \in \mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)$. If $w \in S_1 \cup S_2$ then $w \in \mathsf{analz}(S_1 \cup S_2 \cup T)$.

   - Suppose ϖ is the following proof:

$$\dfrac{\begin{matrix}(\varpi_1) \\ \vdots \\ S_1 \cup S_2 \cup \{t\} \vdash \{w\}_k\end{matrix} \qquad \begin{matrix}(\varpi_2) \\ \vdots \\ S_1 \cup S_2 \cup \{t\} \vdash \overline{k}\end{matrix}}{S_1 \cup S_2 \cup \{t\} \vdash w}\ \mathsf{decrypt}$$

By induction hypothesis $\{w\}_k$ belongs to $\mathsf{analz}(S_1 \cup \{t\}) \cup \mathsf{analz}(S_1 \cup S_2 \cup T)$ and $\overline{k}$ belongs to $\mathsf{analz}(S_1 \cup \{t\}) \cup \mathsf{analz}(S_1 \cup S_2 \cup T)$.

$\{w\}_k \in \mathsf{analz}(S_1 \cup S_2 \cup T)$*:* If $\overline{k} \in \mathsf{analz}(S_1 \cup S_2 \cup T)$ then w is in the same set as well and we are done. If on the other hand $\overline{k} \in \mathsf{analz}(S_1 \cup \{t\})$, then $\overline{k} \in K \cap (\mathsf{analz}(S_1) \cup (\mathsf{analz}(S_1 \cup \{t\}) \setminus \mathsf{analz}(S_1)))$. But this implies that $\overline{k} \in \mathsf{analz}(S_1 \cup T) \subseteq \mathsf{analz}(S_1 \cup S_2 \cup T)$ and hence w is also in the same set.

$\{w\}_k \in \mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)$*:* It is evident that $k \in ST(S_1 \cup \{t\})$. Thus by assumption $\overline{k} \in \mathsf{analz}(S_1 \cup \{t\})$ and hence w is also in the same set. Clearly $w \in ST(t)$ as well.

2. Let us denote by W the set $((\mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)) \cup \mathsf{analz}(S_1 \cup S_2 \cup T)) \cap ST(u)$. It is clear that $u \in \mathsf{synth}(W)$. Now $w \in ST(u)$ for every $w \in W$, and since $EST(u)$ and $EST(t)$ are disjoint it is also the case that $EST(w)$ and $EST(t)$ are disjoint. We prove below that $W \subseteq \overline{S_1 \cup S_2 \cup T}$; this suffices to prove that $u \in \overline{S_1 \cup S_2 \cup T}$.

   So suppose $w \in W$. Then $w \in \mathsf{analz}(S_1 \cup S_2 \cup T) \cup (\mathsf{analz}(S_1 \cup \{t\}) \cap ST(t))$. If $w \in \mathsf{analz}(S_1 \cup S_2 \cup T)$ we are done. Suppose $w \in \mathsf{analz}(S_1 \cup \{t\}) \cap ST(t)$. In this case, as observed above $EST(w) \cap EST(t) = \emptyset$, and hence from $w \in ST(t)$ it follows that $EST(w) = \emptyset$. This means that w is just a tuple of atomic terms. In this case it is clear that $w \in \mathsf{synth}(\mathsf{analz}(\{w\}) \cap \mathcal{T}_0)$. But then $\mathsf{analz}(\{w\}) \cap \mathcal{T}_0 \subseteq \mathsf{analz}(S_1 \cup \{t\}) \cap \mathcal{T}_0 \subseteq \mathsf{analz}(S_1 \cup T)$. This implies that $w \in \overline{S_1 \cup S_2 \cup T}$ and the proof is done.

□

The following lemma is vital in proving that if m is secret at a run ξ of a protocol Pr, then m is also secret at ξ', where ξ' is got by eliminating some events and renaming some atomic terms of ξ.

Lemma 6.2 *Suppose S is a set of terms and $T \subseteq \mathsf{analz}(S) \cap \mathcal{T}_0$. Suppose τ is a substitution with the property that for all $x \in \mathcal{T}_0 \setminus T$, $\tau(x) = x$ and for all $x \in T$, $\tau(x) \in S$. Then for all $t \in \mathsf{analz}(\tau(S))$, there exists $r \in \mathsf{analz}(S)$ such that $\tau(r) = t$.*

Proof: Suppose π is an analz-proof of $\tau(S) \vdash t$. We prove by structural induction that for every subproof ϖ of π with root labelled $\tau(S) \vdash w$, there exists $r \in \mathsf{analz}(S)$ such that $\tau(r) = w$. Suppose ϖ is a subproof of π with root labelled $\tau(S) \vdash w$ such that for all proper subproofs ϖ_1 of ϖ the statement of the lemma holds. Then we prove that it holds for ϖ as well. We only consider the cases when the rule applied at the root of ϖ is Ax_a or $\mathsf{decrypt}$. The other cases can be handled by a routine application of the induction hypothesis.

- Suppose ϖ is the following proof:

$$\frac{}{\tau(S) \vdash w}\ \mathsf{Ax}_a$$

Then $w \in \tau(S)$ which means that there exists $r \in S \subseteq \mathsf{analz}(S)$ such that $\tau(r) = w$.

- Suppose ϖ is the following proof:

$$\frac{\begin{array}{c}(\varpi_1)\\ \vdots \\ \tau(S) \vdash \{w\}_k\end{array} \qquad \begin{array}{c}(\varpi_2)\\ \vdots \\ \tau(S) \vdash \overline{k}\end{array}}{\tau(S) \vdash w}\ \mathsf{decrypt}$$

By induction hypothesis there exist $r', r'' \in \mathsf{analz}(S)$ such that $\tau(r') = \{w\}_k$ and $r'' = \overline{k}$. It is clear that r' is of the form $\{r\}_{k'}$ with $\tau(r) = w$ and $\tau(k') = k$, and r'' is of the form k''. We need to prove that $r \in \mathsf{analz}(S)$.

  – Suppose $k' \in T$. It then follows that $k' \in K_0$ and hence it follows that $k' = \overline{k'}$ and that $\overline{k'} \in \mathsf{analz}(S)$ (since $T \subseteq \mathsf{analz}(S)$). Coupled with the fact that $\{r\}_{k'}$ belongs to $\mathsf{analz}(S)$, we have that $r \in \mathsf{analz}(S)$.

  – Suppose $k' \notin T$. From the definition of τ we see that $k' = k$. Thus $\{r\}_k$ belongs to $\mathsf{analz}(S)$.

    If $k'' \in T$, then since $\tau(T) \subseteq S \subseteq \mathsf{analz}(S)$ it follows that $\overline{k} \in \mathsf{analz}(S)$. If $k'' \notin T$, from the definition of τ it follows that $k'' = \overline{k}$, and thus it is again clear that $\overline{k} \in \mathsf{analz}(S)$ (since $T \subseteq \mathsf{analz}(S)$ and $\overline{k} = k'' \in T$).

    Coupled with $\{r\}_k \in \mathsf{analz}(S)$, this implies that $r \in \mathsf{analz}(S)$, as desired.

□

6.2 Reduction to good runs

In this subsection we proceed to prove the reduction to good runs using the properties proved in the previous subsection. We briefly present the key ideas in the proof before proceeding to the formal presentation. Suppose a given context-explicit protocol has a leaky run ξ. If it is not good, we define a new run by eliminating the latest bad event of ξ. This loses some information — more specifically, the messages occurring later than this event may no longer be constructible. Here is where the results of the previous subsection come in handy. We know that the eliminated event is a bad event and hence does not have encrypted subterms in common with the later events. With some effort we can apply Lemma 6.1 to this case, so all the later messages do not need all of t but just the set T of new terms occurring in t learnt by the intruder during the latest bad event. If the intruder is provided with this information in his initial state then we can show that even after eliminating the latest bad event we have a run. Recall that we let n_0 and k_0 stand for the intruder's knowledge, so adding T to the intruder's state is formally achieved by mapping all of T to $\{\mathsf{n}_0, \mathsf{k}_0\}$. This new run is then shown to be leaky by applying the results of the previous subsection. After the latest bad event is eliminated, it is easy to see that none of the later events become bad. Some of the good events occurring before this bad event might become bad after its elimination. But clearly the index of the latest bad event of this new run decreases. We can now repeat the above process till we get a good run.

Lemma 6.3 *Suppose* $\mathsf{Pr} = (\mathsf{C}, a_1b_1 \cdots a_\ell b_\ell)$ *is a context-explicit protocol which has a leaky run. Then it also has a* good *leaky run.*

Proof: We fix the following notation for the rest of the proof. Fix a leaky run $\xi = e_1 \cdots e_k$ of Pr, none of whose *proper* prefixes is leaky. Let $e_j = (\eta_j, \sigma_j, lp_j)$ for $j \leq k$. For any $j \leq k$, $t_j = \mathit{term}(e_j)$. For any $j : 1 \leq j \leq k$, ξ_j denotes $e_1 \cdots e_j$, s_j denotes $\mathit{infstate}(\xi_j)$ and T_j denotes $(s_j)_I$. For $i, j : 1 \leq i \leq j \leq k$, ξ_j^{-i} denotes $e_1 \cdots e_{i-1}e_{i+1} \cdots e_j$ if $i < j$ and ξ_{i-1} if $i = j$, s_j^{-i} denotes $\mathit{infstate}(\xi_j^{-i})$ and T_j^{-i} denotes $(s_j^{-i})_I$. We also denote $\mathsf{init}(\mathsf{Pr})$ by s_0 and $(s_0)_I$ by T_0.

Suppose ξ is not a good run. This means that there is a bad event in ξ. Let r be the index of the latest bad event in ξ (i.e. $r = \mathit{max}(\{i \leq k \mid e_i \text{ is a bad event of } \xi\})$). Notice that by definition e_k is a good event, and hence $r < k$. Define T to be $(\mathsf{analz}(T_r) \setminus \mathsf{analz}(T_{r-1})) \cap \mathcal{T}_0$. Since ξ_r is not leaky, it follows that no $m \in T$ is secret at ξ_{r-1}. Thus it has to be the case that $T \subseteq NT(e_r) \subseteq N \cup K_0$.

Let τ be a substitution which maps every $n \in T \cap N$ to n_0, every $k \in T \cap K_0$ to k_0 and is identity on all the other terms in $\mathcal{T}_0$. (Recall that n_0 and k_0 are fixed constants in the intruder's initial state.) For all $j \leq k$, we define e'_j to be $(\eta_j, \tau \circ \sigma_j, lp_j)$, where $(\tau \circ \sigma_j)(t) = \tau(\sigma_j(t))$ for all t. We define $\xi' = e'_1 \cdots e'_k$. Analogous to the notations based on ξ, we define the notations t'_j, ξ'_j, s'_j, T'_j, $(\xi')_j^{-i}$, $(s')_j^{-i}$ and $(T')_j^{-i}$ based on ξ'. Note that $t'_j = \tau(t_j)$, $\xi'_j = \tau(\xi_j)$, $T'_j = \tau(T_j)$ and so on.

We now show that $(\xi')_k^{-r})$ is a run of Pr and that it is leaky; but the index of the latest bad event (if any) in $(\xi')_k^{-r}$ is less than r, and hence we can repeat the process on the new run, eventually obtaining a good run.

We now prove that $(\xi')_k^{-r}$ is a run of Pr and that it is leaky, thus concluding the proof of the theorem.

Claim: $(\xi')_k^{-r}$ is a run of Pr.

Proof of Claim: Since ξ is a run, it follows that $NT(e_i) \cap ST(\mathsf{init}(\mathsf{Pr})) = \emptyset$ for all $i \leq k$, and that $NT(e_i) \cap NT(e_j) = \emptyset$ for all $i < j \leq k$. Since $T \subseteq NT(e_r)$ it follows that $T \cap NT(e_q) = \emptyset$ for all $q \neq r$. It thus follows that $NT(e'_q) = NT(e_q)$ for all $q \neq r$. It is now easy to see that for all $i \leq k, i \neq r$, $NT(e'_i) \cap ST(\mathsf{init}(\mathsf{Pr})) = \emptyset$ and that for all $i < j \leq k, i, j \neq r$, $NT(e'_i) \cap NT(e'_j) = \emptyset$. Thus $(\xi')_k^{-r}$ satisfies the unique origination property. We concentrate on proving that all its events are enabled at the end of the preceding events.

By definition of bad events it follows that $e_r \neq e_k$ and for all $q : r < q \leq k$, e_q is not a good successor of e_r. This implies in particular that for all $q : r < q \leq k$, $\neg(e_r \rightarrow_\ell e_q)$. From this it also follows that for all $q : r < q \leq k$, $\neg(e_r \overset{+}{\rightarrow}_\ell e_q)$, i.e., $e_r \notin LP(e_q)$.

- We first consider the case when e_r is a receive event. Then by Proposition 2.14, $T_r = T_{r-1}$ and thus $T = \emptyset$. Then it is clear that τ is the identity map on terms. Hence $\xi' = \xi$. It suffices to prove that ξ_k^{-r} is a run of Pr. Firstly it is clear that ξ_{r-1} is a run of Pr. Consider a q such that $r < q \leq k$. Since all events in $LP(e_q)$ occur in ξ_{q-1} and $e_r \notin LP(e_q)$, it follows that all events in $LP(e_q)$ occur in ξ_{q-1}^{-r}.

  Now if e_q is a receive event, then since $T_r = T_{r-1}$ it is clear that $T_{q-1}^{-r} = T_{q-1}$ and hence $t_q \in T_{q-1}^{-r}$. This suffices to show that e_q is enabled at ξ_{q-1}^{-r}. If e_q is a send event, then since instantiations of roles of Pr are send-admissible, e_q is enabled at ξ_{q-1}^{-r}.

- Let us now consider the case when e_r is a send event. We first show that ξ'_{r-1} is a run of Pr. Since $T \subseteq NT(e_r)$ and since $NT(e_r) \cap ST(s_{r-1}) = \emptyset$, τ does not affect any term occurring in ξ_{r-1}. Hence it follows that for all $q < r$, $t_q = t'_q$, $s_q = s'_q$, and $T_q = T'_q$. Thus for all $q < r$, e'_q is enabled at ξ'_{q-1}. This means that ξ'_{r-1} is a run of Pr.

  We now show that for all $q : r < q \leq k$, e'_q is enabled at $(\xi')_{q-1}^{-r})$. We first note that for any $i < j \leq k$, $e_i \rightarrow_\ell e_j$ iff $e'_i \rightarrow_\ell e'_j$, $e_i \in LP(e_j)$ iff $e'_i \in LP(e'_j)$, and $EST(e_i) \cap EST(e_j) \neq \emptyset$ iff $EST(e'_i) \cap EST(e'_j) \neq \emptyset$. These statements immediately follow from the definitions.

  Fix a q such that $r < q \leq k$. There are two cases to consider:

  - If e_q is a receive event, then it is clear that $t_q \in \mathsf{synth}(U)$ where $U = \mathsf{analz}(T_{q-1}) \cap ST(t_q)$. Consider some $u \in U$ and an analz-proof π of $T_{q-1} \vdash u$. It is clear that for all keys k, if $k \in (s_0)_A$ for some $A \in Ag$ then $\overline{k} \in (s_0)_B$ for some $B \in Ag$. Further for any index i, if $k \in NT(e_i)$, then $k \in K_0$ and hence $k = \overline{k}$. So we can say that for any $k \in K$, if $k \in (s_i)_A$ for some $A \in Ag$ then $\overline{k} \in (s_i)_B$ for some $B \in Ag$. Further note that if $k \in ST(s_i)$ then $k \in (s_i)_A$ for some $A \in Ag$, and therefore $\overline{k} \in (s_i)_B$ as well, for some $B \in Ag$. Now since ξ_{q-1} is not leaky, it follows that whenever $k \in ST(s_r)$ for some $r < q$ and $\overline{k} \in \mathsf{analz}(T_{q-1})$ then $\overline{k} \in \mathsf{analz}(T_r)$. Thus T_{r-1}, $T_{q-1} \setminus T_r$, t_r, T, u and π play the role of S_1, S_2, t, T, u, and π respectively in item 1 of Lemma 6.1 and we get $u \in (\mathsf{analz}(T_r) \cap ST(t_r)) \cup \mathsf{analz}(T_{q-1}^{-r})$. Thus $t_q \in \mathsf{synth}((\mathsf{analz}(T_r) \cap ST(t_r)) \cup \mathsf{analz}(T_{q-1}^{-r} \cup T))$. Now since e_r is not a good predecessor of e_q, $EST(t_q) \cap EST(t_r) = \emptyset$. Thus the conditions of item 2 of Lemma 6.1 are fulfilled, and hence $t_q \in \overline{T_{q-1}^{-r} \cup T}$. Applying Proposition 2.9 and using the fact that $\tau(T) \subseteq T_0$, we conclude that $t'_q = \tau(t_q) \in \overline{\tau(T_{q-1}^{-r}) \cup \tau(T)} = \overline{(T')_{q-1}^{-r}}$. Hence e'_q is enabled at $(\xi')_{q-1}^{-r}$.
  - If e_q is a send event then e'_q is also a send event. Now since the instantiations of roles of Pr are send-admissible it immediately follows that $t'_q \in \overline{(T')_{q-1}^{-r}}$. Hence e'_q is enabled at $(\xi')_{q-1}^{-r}$.

This proves that $(\xi')_k^{-r}$ is a run of Pr.

Claim: $(\xi')_k^{-r}$ is leaky.

Proof of Claim: We first prove that some m which is secret at ξ_{k-1} belongs to $\mathsf{analz}(T_k^{-r} \cup T)$. If e_r is a receive event, then by Proposition 2.14 it follows that $T_k = T_k^{-r}$ and hence there is some m which is secret at ξ_{k-1} and which belongs to $\mathsf{analz}(T_k^{-r})$. (This follows from the fact that ξ is itself leaky). Suppose now that e_r is a send event. Consider an analz-proof of $T_k \vdash m'$ for some m' which is secret at ξ_{k-1}. Let π be a subproof of this proof with the property that the root of π is labelled by some m which is secret at ξ_{k-1} and none of the m'' labelling the nonroot nodes of π is secret at ξ_{k-1}. Then it is clear that T_{r-1}, $T_k \setminus T_r$, t_r, T, m and π play the role of S_1, S_2, t, T, u and π respectively in item 1 of Lemma 6.1 (if $\overline{k}$ labels a node of π and if $k \in ST(s_r)$ then since $\overline{k}$ is not secret at ξ_{k-1} it follows that $\overline{k} \in \mathsf{analz}(T_r)$) and we get $m \in (\mathsf{analz}(T_r) \cap ST(t_r)) \cup \mathsf{analz}(T_k^{-r} \cup T)$. But since ξ_r is not leaky, $m \notin \mathsf{analz}(T_r)$. Thus $m \in \mathsf{analz}(T_k^{-r} \cup T)$. From this it follows that $\tau(m) \in \mathsf{analz}((T')_k^{-r})$.

We now prove that $\tau(m)$ is secret at $(\xi')_{k-1}^{-r}$. Since m is secret at ξ_{k-1} and $T \subseteq \mathsf{analz}(T_r) \subseteq \mathsf{analz}(T_{k-1})$, it follows that $m \notin T$. Therefore $\tau(m) = m$. Since m is secret at ξ_{k-1}, it is clear that $m \notin \mathsf{analz}(T_{k-1})$. Now we observe that $T \subseteq \mathsf{analz}(T_r) \cap \mathcal{T}_0 \subseteq \mathsf{analz}(T_{k-1}) \cap \mathcal{T}_0$. Further, for all $x \in \mathcal{T}_0 \setminus T$, $\tau(x) = x$ and for all $x \in T$, $\tau(x) \in T_{k-1}$. Thus T_{k-1}, T and τ satisfy the conditions of Lemma 6.2, and we thus see that whenever $t \in \mathsf{analz}(T'_{k-1})$ there exists $r \in \mathsf{analz}(T_{k-1})$ with $\tau(r) = t$. When $t = m$, it immediately follows that $r = m$ as well (since $m \notin T$). This coupled with the fact that $m \notin \mathsf{analz}(T_{k-1})$ implies that $m \notin \mathsf{analz}(T'_{k-1})$. From this it follows that $m \notin \mathsf{analz}((T')_{k-1}^{-r})$ as well, and thus that $\tau(m) = m$ is secret at $(\xi')_{k-1}^{-r}$. This concludes the proof that $(\xi')_k^{-r}$ is leaky.

We have thus proved the reduction to good runs. □

Lemma 6.3 and Theorem 5.4 immediately yield us the following theorem.

Theorem 6.4 *The problem of checking for a given context-explicit protocol* Pr *whether there is a leaky run of* Pr *is decidable.*

7 Discussion

A naive algorithm to check secrecy based on the proofs presented here would proceed by trying to guess a good leaky run of a given context-explicit protocol (presented as a sequence of actions). Since good runs are of bounded length, and further the bound on their length also determines a bounded universe of atomic terms from which to construct terms, we have a NEXPTIME algorithm. But we conjecture that this can be improved to get a DEXPTIME algorithm. We believe that it would be an algorithm which tries to incrementally build a good leaky run and use an appropriately intelligent backtracking strategy so as not to explore more than an exponential number of states.

Most well-formed protocols occurring in the literature ([CJ97]) can be easily transformed into "equivalent" context-explicit protocols by the use of simple tagging schemes. For example, the Needham-Schroeder protocol presented in Example 2.4 can be modified to the following form:

$$\begin{array}{llcll}
\text{Message 1.} & A & \rightarrow & B & : \{1, A, x\}_{pubk_B} \\
\text{Message 2.} & B & \rightarrow & A & : \{2, x, y\}_{pubk_A} \\
\text{Message 3.} & A & \rightarrow & B & : \{3, m, y\}_{pubk_B}
\end{array}$$

The new nonce m has been added to Message 3 to make sure that the second condition in the definition of context-explicit protocols, namely that every message has a unique nonce occurring in all its encrypted components, is met. We have also added number tags to ensure the the first condition, which ensures that encrypted components across messages are not unifiable.

An alternative approach is to transform the semantics of protocols rather than protocols themselves. Tagging schemes similar to those we have proposed can be introduced in the run-generation mechanism to capture a notion of "well behaved runs". Here we should note that while non-unifiability across communications can be ensured by the use of a finite number of tags (which can be determined from the protocol specification), adding a new nonce to each message is quite an expensive operation. The trouble with maintaining nonces is that they need to be unguessable for most purposes. But we claim that the proofs of our decidability proofs never require this property of the nonces that are added as tags. An analysis of our decidability proof shows that the only place where the assumption on nonces is used is in the proof of Proposition 3.7, and there the only property used is the fact that distinct events generate distinct new nonces. Thus it suffices to use some other more efficient notion like *sequence numbers* or *play identifiers* and achieve the effect needed for our decidability proof to go through. This discussion makes it clear that we can efficiently associate a *tagged event* with every event of a given well-formed protocol.

Such simple schemes do not always work though. Consider for example the Woo and Lam protocol π_1 from [WL94]:

$$\begin{array}{llcll}
\text{Message 1.} & A & \rightarrow & B & : A \\
\text{Message 2.} & B & \rightarrow & A & : x \\
\text{Message 3.} & A & \rightarrow & B & : \{A, B, x\}_{k_{AS}} \\
\text{Message 4.} & B & \rightarrow & S & : \{A, B, \{A, B, x\}_{k_{AS}}\}_{k_{BS}} \\
\text{Message 5.} & S & \rightarrow & B & : \{A, B, x\}_{k_{BS}}
\end{array}$$

Here A passes $\{A, B, x\}_{k_{AS}}$ to B in message 3. B cannot decrypt the message and read its contents but just passes it on to the server S in message 4. Now according to the requirement of context-explicit protocols distinct tags have to be added to the component $\{A, B, x\}_{k_{AS}}$ occurring in message 3 and message 4. But such a tagging scheme cannot be implemented since B cannot replace the tags before passing on the message to S. Here we see that tagging conflicts with admissibility. It can be seen that this problem exists also with respect to other protocols with "blind relay" — for instance the Yahalom protocol, Denning-Sacco protocol, and Ottway-Rees protocol, as presented in [CJ97]. It has been noted however ([Low99]) that one can find protocols "equivalent" to the Woo-Lam protocol which avoid the problem of "blind relay" as above. For instance, A could send the message directly to the server instead of through B. Another point to be noted is that the tagging scheme used in [BP03] can be used to tag even protocols with blind relays.

In the informal discussion above we referred to an equivalence on protocols. We present a few definitions which formalise this notion. This notion has already been discussed in the literature in some form or other. (See [HL01], for instance).

We first define when a run of a given protocol is equivalent to a run of a different protocol.

Definition 7.1 *Given two sets of events, E and E', a one-one function $f : E \to E'$, and a set T of terms, we say that a run ξ is (f, T)-equivalent to ξ' if $\xi' = f(\xi)$ and for all prefixes ξ_1 of ξ and $A \in Ag$, $(\mathit{infstate}(\xi_1))_A \cap T = (\mathit{infstate}(f(\xi_1)))_A \cap T$.*

Definition 7.2 *A run $e_1 \cdots e_k$ of a protocol is said to be* honest *iff for all $j \leq k$ such that $act(e_j)$ is a receive action, there exists $i < j$ such that $act(e_i)$ is a send action and $act(e_j)$ is its matching receive.*

Thus, the intruder plays a relatively passive role in honest runs. The most it does is to block certain messages and deliver multiple copies of certain messages to the *intended recipient*.

Definition 7.3 *A protocol Pr' is a* safe transform *of another protocol Pr if there exists a one-one function f : $\mathit{Events}(\mathsf{Pr}) \to \mathit{Events}(\mathsf{Pr}')$ and a set $C \subseteq \mathsf{CT}(\mathsf{Pr}')$ such that (letting $T = \mathcal{T} \setminus C$):*

- *every honest run of Pr has an (f, T)-equivalent honest run of Pr', and*
- *every run of Pr' is (f, T)-equivalent to some run of Pr. (In particular, for every leaky run of Pr' there is a corresponding in the original protocol Pr.)*

The set C is the set of special constant terms used as tags in the messages of the original protocol. It is to be noted that for most of the tagging schemes of the kind outlined above, for every leaky run of the tagged version there is an equivalent leaky run in the original version. (Simply omitting the tags usually gives us a leaky run of the original protocol.) But the problem with protocols like the Woo-Lam protocol is that not all honest runs have corresponding tagged runs.

The above definitions are presented in full generality without considering the question of effectiveness. Hence the problem of checking whether a given well-formed protocol has a safe transform, or even the simpler one of checking whether a given protocol is a safe transform of another given protocol, might not be effectively solvable. But it is an easy matter to define transformations as functions from one protocol specification to another, and then we can handle effectiveness issues. But if we want to construct *efficient* safe transforms of well-formed protocols to context-explicit protocols, we might need to adapt our model slightly depending on the situation. For instance, we might add a new syntactic entity called *sequence number*, with the idea that we have available a sequence number generator which provides a distinct (but not necessarily nonguessable) number at each invocation, and use these entities in the syntactic transformation of protocols. Such transformations have the desired property of being easy to implement, while at the same time being amenable to useful theoretical discussion.

As mentioned earlier, the secrecy problem is undecidable in general. It can be seen that relaxing the conditions on well-formed protocols and context-explicit protocols allows us to code the halting problem for two-counter machines as a secrecy problem. The idea in the coding is to represent transitions of two-counter machines as roles of the protocol. The terms used in the protocol represent configurations of the two-counter machine, which are of the form (q, m, n) for some natural numbers m and n. The roles of the protocol look like the following:

$A?B\colon \{q, y, x\}_{k_{AB}}, \{x', x\}_{k_{AB}}$; $\quad A!B\colon (y')\ \{q', y', x'\}_{k_{AB}}, \{y, y'\}_{k_{AB}}$.

Note that the syntax restriction is not respected by this protocol as distinct communications have encrypted subterms – $\{q, y, x\}_{k_{AB}}$ and $\{q', y', x'\}_{k_{AB}}$, for instance – which are unifiable. The ability to generate new nonces allows us to code the natural numbers, and the unifiability of encrypted terms allows us to code the behavior of the machines which use the output configuration of one transition as the input configuration of another. This is the key to undecidability. (The report [RS03d] has more details.) Another point to note is that protocols like the above which code up machines are typically not even well-formed. In fact, it is still open whether the secrecy problem for well-formed protocols is decidable.

While we have studied only the secrecy problem here, we can observe that the technique may be used to prove the decidability of other security problems as well. For instance, suppose we wish to check whether there is a run of a protocol in which, at some event, the proposition A *has* m holds; that is, in the information state of A after this event, A has the (atomic) term $m \in \mathcal{T}_0$. If we come up with a suitable notion of good runs particular to each such property, then the reduction to good runs can be performed as described in the paper. This suggests that the exercise may well be carried out for a simple modal logic with an eventuality modality.

Acknowledgements

We thank the anonymous referees whose comments and suggestions have improved the presentation considerably.

References

[AC02] Roberto M. Amadio and Witold Charatonik. On name generation and set-based analysis in Dolev-Yao model. In *CONCUR 2002*, volume 2421 of *Lecture Notes in Computer Science*, pages 499–514, 2002.

[ALV02] Roberto M. Amadio, Denis Lugiez, and Vincent Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, 2002.

[AN95] Ross Anderson and Roger M. Needham. Programming Satan's computer. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–441, 1995.

[AN96] Martin Abadi and Roger M. Needham. Prudent engineering practices for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22:6–15, 1996.

[BP03] Bruno Blanchet and Andreas Podelski. Verification of Cryptographic Protocols: Tagging Enforces Termination. In Andrew D. Gordon, editor, *Proceedings of FoSSaCS'03*, volume 2620 of *Lecture Notes in Computer Science*, pages 136–152, 2003.

[CCM01] Hubert Comon, Véronique Cortier, and John C. Mitchell. Tree automata with One Memory, Set Constraints, and Ping-Pong Protocols. In *Proceedings of ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, 2001.

[CJ97] John Clark and Jeremy Jacob. A survey of authentication protocol literature. Electronic version available at `http://www.cs.york.ac.uk./~jac`, 1997.

[CMS00] Iliano Cervesato, Catherine A. Meadows, and Paul F. Syverson. Dolev-Yao is no better than Machiavelli. In P. Degano, editor, *Proceedings of WITS'00*, July 2000.

[DEK82] Danny Dolev, Shimon Even, and Richard M. Karp. On the Security of Ping-Pong Protocols. *Information and Control*, 55:57–68, 1982.

[DLMS99] Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. The undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[DY83] Danny Dolev and Andrew Yao. On the Security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.

[HL01] Men Lin Hui and Gavin Lowe. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Security*, 9(1, 2):3–46, 2001.

[HT96] Nevin Heintze and Doug Tygar. A model for secure protocols and their composition. *IEEE Transactions on Software Engineering*, 22:16–30, 1996.

[Low99] Gavin Lowe. Towards a completeness result for model checking of security protocols. *Journal of computer security*, 7:89–146, 1999.

[MS01] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 166–175, 2001.

[NS78] Roger M. Needham and Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of computer security*, 6:85–128, 1998.

[RS03a] R. Ramanujam and S.P. Suresh. A decidable subclass of unbounded security protocols. In Roberto Gorrieri, editor, *Proceedings of WITS'03*, pages 11–20, April 2003.

[RS03b] R. Ramanujam and S.P. Suresh. An equivalence on terms for security protocols. In Ramesh Bharadwaj, editor, *Proceedings of AVIS'03*, pages 45–56, April 2003.

[RS03c] R. Ramanujam and S.P. Suresh. Tagging makes secrecy decidable for unbounded nonces as well. In *Proceedings of 23rd FST&TCS*, volume 2914, pages 363–374, Mumbai, India, December 2003.

[RS03d] R. Ramanujam and S.P. Suresh. Undecidability of secrecy for security protocols. Electronic version available at `http://www.imsc.res.in/~jam`, 2003.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with finite number of sessions is NP-complete. *Theoretical Computer Science*, 299:451–475, 2003.

[Sur03] S.P. Suresh. *Foundations of Security Protocol Analysis*. PhD thesis, The Institute of Mathematical Sciences, Chennai, India, November 2003. Submitted to Madras University.

[WL94] Thomas Y.C. Woo and Simon S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, 28(3):24–37, 1994.