Looking for Diamonds in the Desert — Extending Automatic Protocol Generation to Three-Party Authentication and Key Agreement Protocols*

Adrian Perrig
perrig@cs.berkeley.edu

Dawn Song
dawnsong@cs.berkeley.edu

Computer Science Department
University of California, Berkeley

Abstract

In this paper, we describe our new results in developing and extending Automatic Protocol Generation (APG), an approach to automatically generate security protocols. We explore two-party mutual authentication and key agreement protocols, with a trusted third party (TTP) which shares a symmetric key with each of the two principals. During the process, we experienced the challenge of a gigantic protocol space (even after we limit the protocol complexity and size, and after constraining message format and sending order, we estimate the state space at 10^{12} protocols!). Facing this challenge, we develop more powerful reduction techniques for the protocol generator. We also develop new pruning theorems and probabilistic methods of picking goal orderings for the protocol screener, Athena, which greatly improve the efficiency and worst-case performance of Athena.

In our first experiment, APG found new protocols for two-party mutual authentication with a TTP using symmetric keys. In our second experiment, APG also found new protocols for three different sets of security properties for two-party authentication and key agreement. Our new list of security properties for key agreement also uncovered an undocumented deficiency in the Yahalom protocol.

*We gratefully acknowledge support for this research from several US government agencies. This research was suported in part by the Defense Advanced Research Projects Agency under DARPA contract N6601-99-28913 (under supervision of the Space and Naval Warfare Systems Center San Diego), by the National Science foundation under grant FD99-79852, and by the United States Postal Service under grant USPS 1025 90-98-C-3513. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied of the US government or any of its agencies, DARPA, NSF, USPS.

1 Introduction

1.1 Motivation for Automatic Protocol Generation

The current process of designing a security protocol is usually ad-hoc and involves little formalism and mechanical assistance. Such a design process is not only slow but also error-prone. Evidence shows that even when security protocols are designed with care and examined intensely (even over time of years), they can still be fundamentally flawed. A classic example is the Needham-Schroeder public-key mutual authentication protocol [NS78], in which Gavin Lowe discovered a flaw 18 years later [Low96]. Due to the lack of formalism and mechanical assistance, manually designed protocols often contain undocumented assumptions and hence can lead to implementation errors. Further, simply because a manual design process cannot search a large number of candidates, it may not find the optimal protocol for the given system requirements.

Hence, it is desirable to explore a different approach, *automatic protocol generation* (APG for short). With APG, the protocol designer specifies the desired security requirements, such as authentication and secrecy, and system specification, e.g., symmetric or asymmetric encryption/decryption, low bandwidth. A *protocol generator* then generates *candidate* security protocols which satisfy the system requirements. In the final step, a *protocol screener* analyzes the candidate protocols, discards the flawed protocols, and outputs the correct protocols that satisfy the desired security properties.

The approach of APG is fully automatic and provides high confidence and high quality of protocols. Since the input specifications are written in a well-defined specification language and the APG is fully automatic, there are no hidden assumptions. The protocol screener has a powerful engine which can automatically generate a proof of cor-

rectness if a protocol is correct, or a counterexample otherwise. The user-defined system requirements includes a metric function which specifies the cost or overhead of a protocol. APG tries to generate correct protocols with minimal cost with respect to a metric function hence suits the system requirements the best.

1.2 Results in this Paper

In this paper, we describe our new results in developing and extending Automatic Protocol Generation (APG) to explore two-party mutual authentication and key agreement protocols with TTP using symmetric keys. During the process, we experienced the challenge of a gigantic protocol space (even after we limit the protocol complexity and size, and after constraining message format and sending order, we estimate the protocol space at 10^{12} protocols!). Facing this challenge, we develop powerful reduction techniques for the protocol generator. We also develop new pruning theorems and probabilistic methods of picking goal orderings for the protocol screener, Athena, which greatly increase the efficiency and worst-case performance of Athena. In our first experiment, APG found new protocols for two-party mutual authentication with TTP using symmetric keys. In our second experiment, APG also found new protocols for three different sets of security properties for two-party authentication and key agreement in the same setting. Our new list of security properties for key agreement also discovered an undocumented deficiency in the Yahalom protocol [CJ97]. In fact, APG found protocols which have the same cost as the Yahalom but stronger security properties. APG also found protocols which have the same security properties as Yahalom but with a smaller cost.

The remainder of this paper is organized as follows. In Section 2 we briefly review the main concepts and methods of automated protocol generation. Section 3 describes the reduction techniques for the protocol generator. We describe improvements to the protocol screener Athena in Section 4. Section 5 describes our new results in automatically generate two-party mutual authentication protocols, where both parties share a symmetric key with a trusted third party. In the same setting we also analyze protocols for key agreement, which provide key secrecy, freshness, and key confirmation.

2 APG Overview

The approach of Automatic Protocol Generation (APG) was firstly introduced by Perrig and Song [PS00]. As shown in Figure 1, APG consists of three stages. First, the protocol designer enters the *security requirements* and *system specification* into the *protocol generator* (PG). Second, based on the system specification, PG generates *candidate protocols* along with *verification conditions* (VC) for each protocol. Finally, the *protocol screener* (PS) verifies all VC's of each protocol. If all VC's of a protocol are correct, PS outputs it as a *correct protocol*. In the remainder of this Section, we explain each of these components in more detail.

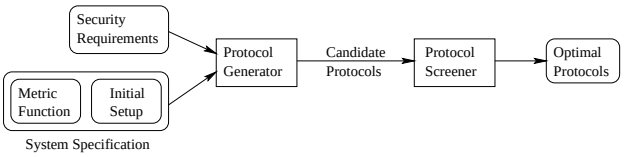

Figure 1. APG overview

2.1 Specification of System Requirements and Security Properties

The system requirements are specified as a metric function and a specification of the initial setup. The initial system configuration defines which cryptographic primitives are available to the principals and what keys each principal possesses. For example, in a PKI environment, all protocol parties know their own private key and the public keys of the other principals, whereas in a symmetric-key environment the principals have shared secret keys. Hybrid systems are also possible.

The metric function corresponds to the cost or overhead of the protocol. An example for metric design is to make the metric correspond to the time overhead of the protocol. In a system such as a smart-card, encryption can be fast while the bandwidth between the card and the card reader may be low, in which case the metric function specifies a low cost for encryption, whereas the cost of sending and receiving messages is high.

The metric function is required to monotonically increase as the protocol complexity increases. This requirement is necessary during the protocol generation phase, where all the protocols up to a maximum cost threshold are generated.

The metric function defines an ordering among the protocols generated. The screener analyzes the protocols in the order of increasing cost,[1] hence the first correct protocol has a minimal cost-value with respect to the metric function. Given a specification of security properties and system requirements, we say that a protocol is *optimal* if it has the lowest cost-value with respect to the metric function among protocols that satisfy the security properties and the system requirements.

The security properties are specified using predicates in a logic. Later in the paper, we describe in more detail how

[1] All protocols are sorted after the generation step, since the generation might not generate them in strictly increasing order

$$
\begin{aligned}
Message &::= Atomic \mid Encrypted \mid Concatenated \\
Atomic &::= \textit{Principalname} \mid \textit{Nonce} \mid Key \\
Encrypted &::= (Message, \textit{Key}) \\
Key &::= \textit{PublicKey} \mid \textit{PrivateKey} \mid \textit{SymmetricKey} \\
Concatenated &::= Message\ List
\end{aligned}
$$

Figure 2. Message grammar

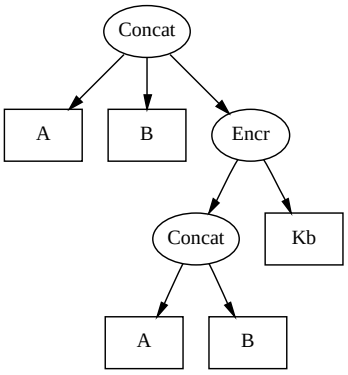

Figure 3. An example message tree

APG specifies properties such as authentication, key agreement, and key secrecy.

2.2 Protocol Representation

A protocol defines a sequence of actions of the participating parties. The actions include sending and receiving messages. We call each sending and receiving of a message one *round* of the protocol and the first message as the first round of the protocol. Messages are defined by the grammar shown in Figure 2. The grammar can be easily extended if needed.

Each message can be represented as a tree with the atomic messages as leaves and operations as intermediate nodes. Figure 3 shows an example for the message: $A, B, \{A, B\}_{K_B}$. We define the *depth* of a message as the depth of the tree representing that message. For example, in Figure 3 the depth of the message tree is 4.

2.3 The Protocol Generator

The purpose of the protocol generator is to generate candidate protocols that satisfy the specified system specification and to generate VCs according to the security requirements. Intuitively, the protocol space is infinite. Hence, we need a way to limit the number of candidate protocols generated while not omitting any potential optimal protocols.

Our primary method to solve this problem is to use *iterative deepening*, a standard search technique [RN95]. In each iteration, we set a *cost threshold* of protocols. We then search through the protocol space to generate all the protocols below the given cost threshold. After sorting the protocols, the protocol screener tests them in the order of increasing cost. If one protocol satisfies the desired properties, it is minimal with respect to the cost metric and the generation process stops. Otherwise, we increase the cost threshold and generate more protocols.

2.4 Protocol Screener

The automatic protocol screener needs to be sound and efficient. Given a candidate protocol, the protocol screener has to be able to examine the protocol and tell whether it is correct or not. Since the protocol generator generates thousands of candidate protocols, the protocol screener needs to be highly efficient to find the optimal protocol in a reasonable amount of time.

We use Athena [Son99], a recently introduced automatic checker for security protocol analysis, as the protocol screener. Athena has the ability to analyze protocol executions with any arbitrary protocol configuration. When Athena terminates, it either provides a proof that a protocol satisfies its specified property under any arbitrary protocol configuration if it is the case, or it generates a counterexample if the property does not hold. To be efficient, Athena also exploits many state space reduction techniques which highly reduces the state space.

3 Protocol Generation

The protocol generation for two-party authentication and key agreement protocols with TTP is particularly challenging, due to the vast protocol space. Our estimate shows that the space of protocols we search is on the order of 10^{12}, (see Appendix D for details on this estimate). Our protocol generator generates and analyzes 10000 protocols per second, which would take over three years to explore the entire space. Clearly, we need to apply powerful protocol space reduction techniques to make APG practical.

In this section, we first introduce some notions about reductions and then describe some specific reduction techniques we use in the protocol generator.

3.1 Notions

Early-pruning / late-pruning reduction. An *early-pruning* reduction prunes a protocol after inspecting only its initial rounds, i.e. the first and second round. On the

contrary, *late-pruning* reductions remove protocols after inspection of almost all the rounds. Clearly, early-pruning reductions are much more efficient since they prune the state-space exploration at an early stage.

Minimal correct protocol / decorated protocol. A *minimal correct protocol* is a correct protocol, which becomes flawed as soon as any message component is removed. An *decorated protocol* is derived from a minimal correct protocol by adding message terms to the protocol, such that the resulting protocol is still correct. It is desirable to prune the decorated protocols in the protocol generation process, since we are only interested in minimal correct protocols.

Lossless / quasi-lossless / lossy reduction. A protocol-space reduction technique is *lossless* if it does not prune any correct protocols. A *quasi-lossless reduction* may prune some decorated protocols, but never prune minimal correct protocols. A *lossy* reduction may also lose optimal protocols. From our experience, the quasi-lossless reductions are often early-pruning reductions and more aggressive than lossless reductions, which is very effective in limiting the search space. Ideally, we would like to use only lossless and quasi-lossless reductions.

3.2 Reduction Techniques

The reduction techniques consist of simple syntactical restriction rules and more sophisticated pruning algorithms. We assume that messages are well-typed. Namely, we do not consider type flaws. Furthermore, during protocol generation process, we do not consider permutations of message components of a concatenated message. In other words, two concatenated messages will be considered equivalent if they only differ in the ordering of their components. This highly reduces the protocol space although it is a lossy reduction. In Section 5 we show that this reduction does lose the optimal protocol in one of our experiments and we also show our method to fix this problem. For convenience, we use A to denote the initiator, B for the responder and S for TTP in the rest of this section.

Syntactical restriction rules

The following is some of the syntactical restriction rules we use in the protocol generator. We believe most of the syntactical restriction rules are quasi-lossless although we do not prove it here.

1. Sending-order restrictions. These rules prevent impossible sending orders, such as $A \rightarrow B, S \rightarrow A, S \rightarrow B, A \rightarrow B$. Such a sending sequence makes no sense, since S cannot know that A wants to initiate a protocol with B. Because the following rules ensure a sane sending order, it is intuitive to see that these reductions are lossless.
   - A always starts. B and S can only start sending if they have received a message.
   - Each principal must send and receive at least one message in the protocol.
   - No principal sends two messages consecutively to the same receiver without receiving any message in between.
2. Message-level restrictions. Since the following reductions operate on the message level, they prune messages in each round and are therefore early-pruning reductions.
   - In any concatenated message, there are no redundant message components, i.e., N_A, N_A.
   - No long-term initial keys are sent in a message.
   - Never use nested encryptions which use the same key twice consecutively, i.e. no $\{\{X\}_{K_A}\}_{K_A}$.
   - No useless encryptions. For example, don't send an encrypted message which only contains principal names.
3. Protocol-level restrictions. These reductions operate on the entire protocol and therefore fall into the category of late-pruning reductions.
   - Check that both parties receive the session key during the protocol in the case of key agreement.
   - Nonce-flow requirements for freshness. If a principal wants to authenticate another principal or check the freshness of a session key, it has to generate a nonce and receives it back as a different form.
   - The final message in a protocol should not contain any part which the recipient cannot decrypt.

Early-pruning by simple impersonation attack

A good strategy to speed up the APG process is to subject each protocol to a simple and efficient analysis, which filters out a large number of trivially flawed protocols. Previously, PG uses an impersonation attack to rule out a majority of protocols failed the authentication property. This attack tests whether an adversary M, who eavesdrops all the messages, can generate the same messages as a legal principal P, in which case M can impersonate P. In our previous experiments, PS (Athena) analyzes 10 protocols per second, but PG can perform a simple impersonation attack to prune protocols at 10000 per second. Although this technique was already efficient, it was a late-pruning reduction. Because of the gigantic protocol space we faced in the new experiments, we improved this reduction to be early-pruning by

using the following observation: a principal can only save itself from an impersonation attack by sending a message which its impersonator cannot generate. Therefore, if the number of principals which can be impersonated is larger than the number of rounds remaining, we can prune the current message. In the case of three-party and four-round authentication protocols, this reduction already starts pruning messages in the second round, and is therefore much more efficient than before.

4 Improvements on Athena

4.1 Overview of Athena

Athena is developed by Dawn Song [Son99, SBP00]. It uses the *Strand Space Model* (SSM) [THG98] to represent protocol executions. Athena uses a logic suitable for SSM to express various security properties, such as authentication, secrecy, and key agreement. It uses an automatic procedure for evaluating well-formed formulas in this logic. If the evaluation procedure terminates, it generates either a proof or a counterexample, depending on the validity of the formula.

Intuitively, the automatic procedure is an efficient exhaustive search for all possible protocol executions that satisfy certain constraints. Athena uses a special state structure, *semi-bundle*, to represent sets of protocol executions efficiently. When the search procedure terminates, it reaches a finite set of *leaf-states* which correspond to the possibly infinite set of all protocol executions satisfying given constraints. Hence, Athena can reason about security properties of a protocol in an arbitrary configuration and with unbounded number of concurrent runs by simply analyzing the finite set of *leaf states*.

In the rest of this section, we describe some new improvements on Athena which enables it to analyze thousands of the synthesized protocols.

4.2 New Pruning-theorems

Athena exploits several state space reduction techniques to reduce the state space that needs to be explored. Besides backward search and symbolic representation with SSM, it also has the advantage that it can systematically incorporate results from theorem proving through *pruning-theorems*. Pruning-theorems are statements about whether a state is a *contradictory state*. A state is a contradicotory state if no bundle can contain it, and hence, can be pruned from the search tree immediately. Pruning-theorems can be either specific to a particular protocol, or general theorems that are not restricted to any concrete example. In the latter case they can be proven once and for all and included in the core of the tool.

In [GT00], Guttman and Thayer introduce the notion of *transforming / transformed edges* and the theorems of *authentication tests*. The authentication theorems are effective as pruning-theorems to reduce the state space, although it has the limitation that it cannot allow certain terms to be encrypted (more specifically, the second clause of definition 4.2). In our experience of APG for three-party key agreement protocols, most of the generated protocols violate this condition. Hence, we extend the concept of authentication tests and prove some new theorems without the original limitation. These theorems play an important role in speeding up the analysis of the large number of the synthetic three-party key agreement protocols. We describe the new pruning-theorems in the following. Please refer to Appendix A for more details on SSM and Appendix C for the proofs of the new theorems.

Definition 4.1. *The edge $n_1 \Rightarrow^+ n_2$ is a transformed edge for $a \in A$ if n_1 is positive, n_2 is negative, $a \sqsubset term(n_1)$, and there is a new component t_2 of n_2 such that $a \sqsubset t_2$.*

Symmetrically, $n_1 \Rightarrow^+ n_2$ is a transforming edge for $a \in A$ if n_1 is negative, n_2 is positive, $a \sqsubset term(n_1)$, and there is a new component t_2 of n_2 such that $a \sqsubset t_2$. (As Definition 2.4 in [GT00]).

Definition 4.2. *$t = \{|h|\}_K$ is a test component for a in n if:*

1. *$a \sqsubset t$ and t is a component of n;*
2. *If t is a proper subterm of a component of n', then n' is not regular, for every $n' \in \Sigma$.*

As Definition 2.5 in [GT00].

Definition 4.3. *We define a relationship $a \ll_K m$, iff. a is an atomic message, and $m = \{t\}_K$, where a is a component of t.*

Proposition 4.1. *Every non-empty subset of the nodes in a bundle $\mathcal{C}$ has $\preceq_{\mathcal{C}}$-minimal members. (As Proposition A.6 in [GT00].)*

Lemma 4.2. *Let C be a normal bundle with $n \Rightarrow^+ n'$ in $\mathcal{C}$. If a uniquely originates from n, and in $term(n)$, a is only in subterm $t_1 \sqsubset n$ and $a \ll_K t_1$, and $a \sqsubset n', t_1 \not\sqsubset n'$. Then there exists a transforming edge $m \Rightarrow^+ m'$ for a and $t_1 \sqsubset m$, and there is a new subterm $t_2 \sqsubset m', a \sqsubset t_2$ and $t_1 \not\sqsubset t_2$.*

Moreover, if $K^{-1} \notin \mathsf{P}$, then there exists a transforming edge as described above which lies on a regular strand.

The proof sketch is in Appendix C.

Lemma 4.3. *Let C be a bundle with $n \Rightarrow^+ n' \in \mathcal{C}$. If a uniquely originates from n, and in $term(n')$, $a \ll_K t_1$ and $t_1 \sqsubset n', t_1 \not\sqsubset n$. Then there exists a transforming edge $m \Rightarrow^+ m'$, where $t_1 \sqsubset m', t_1 \not\sqsubset m, a \sqsubset m$.*

Moreover, if $K \notin \mathsf{P}$, then there exists a transforming edge as described above which lies on a regular strand.

The proof sketch is in Appendix C.

Lemma 4.4. *Let C be a bundle with a negative node $n \in \mathcal{C}$. If in $term(n)$, $a \ll_K t_1$ and $t_1 \sqsubset n, K \notin \mathsf{P}$. Then there exists a positive regular node $m \in \mathcal{C}$, where $t_1 \sqsubset m$.*

The proof is similar to the previous.

4.3 Probabilistic methods for picking goal orderings

In Athena, the search procedure is based on *goals* and *goal-bindings*. Intuitively, *goals* are the received messages. For any goal, there has to be a sender who sends the goal the first time. *Goal-binding* is to bind to the goal to its earliest sender. If a state contains a goal which cannot be bound, the state is a contradictory state and is pruned from the search. Otherwise, after the goal-binding of a goal, a set of next-states will be generated. Every state has a set of goals. Theoretically, the search procedure can choose the goals with any arbitrary ordering. An optimized ordering, however, would lead to a much smaller state space because it can prune many contradictory states at an early stage, while a poor goal ordering can result in a very large state space or even non-termination. Athena has some built-in heuristics on how to choose the goal orderings. For example, messages which are encrypted with a secret key have a higher ranking than others. These heuristics work well in general. But in the process of checking the synthesized protocols, the heuristics can perform poorly on some uncommon cases. In order to prevent the worst-case performance on the heuristics, we apply probabilistic methods for picking the goal orderings. Basically, the search procedure uses the ranking from the heuristics to assign the probability of picking each goal in a state. This alleviates the worst-case performance on the heuristics. It is worth to point out that probabilistic methods are used in artificial intelligence for searching of plans, but seldom used in theorem proving for searching of proofs.

5 Automatic generation of protocols for two-party authentication and key agreement with TTP using symmetric keys

In this section, we first introduce how to specify security properties in Athena. Then we describe our results in automatic generation of protocols for two-party authentication with TTP. Finally we show our findings in automatic generation of protocols for two-party authentication and key agreement with TTP.

For convenience, we use A to represent an arbitrary legitimate initiator, B for an arbitrary legitimate responder, S for the legitimate TTP, I for the intruder, N_a for the nonce generated by A, N_b for the nonce generated by B, K_{AS} for the symmetric key between A and S, K_{BS} for the symmetric key between B and S, and K_{ab} for the session key generated by S. The experiments are done on a Pentium III PC with 256M of memory. All the experiments have the same metric functions: all the atomic operations have the same unit cost. Figure 4 shows the GUI of APG we used in the experiments.

5.1 Specification of security properties

Athena uses a logic on strand space model to specify security properties. It contains nodes, strands and bundles as literals, noted as n, s, c respectively, and atomic proposition $s \in c$. Well-formed formulae (wffs) are composed from atomic propositional formulae with the usual connectives and quantifiers over the bundle variables. Please refer to Appendix B for more details.

We use the notation $Resp(\vec{x})$ to represent the responder strand with the parameter list $\vec{x}$, and $Init(\vec{x})$ to represent the initiator strand in the paper. For example, the formula

$$\forall C. Resp[A, B, S, N_a, N_b, K_{ab}] \in C \implies Init[A, B, S, N_a, N_b, K_{ab}] \in C$$

means that for any bundle C, if strand $Resp$ with the binding $A, B, S, N_a, N_b, K_{ab}$ is in C, then the strand $Init$ with the same binding must also be in C.

5.2 Authentication with TTP using symmetric keys

The specification of the authentication properties are as follows:

If A completes the protocol run with B in a session with N_a and N_b, then there must be a unique responder protocol run with the same binding of parameters. The same holds for B also.

1. $$\forall C. Init[A, B, S, N_a, N_b] \in C \implies Resp[A, B, S, N_a, N_b] \in C$$

   The uniqueness of the protocol run of B is achieved by the freshness of N_b generated in the protocol run.

2. $$\forall C. Resp[A, B, S, N_a, N_b] \in C \implies Init[A, B, S, N_a, N_b,] \in C$$

   The uniqueness of the protocol run of A is achieved by the freshness of N_a generated in the protocol run.

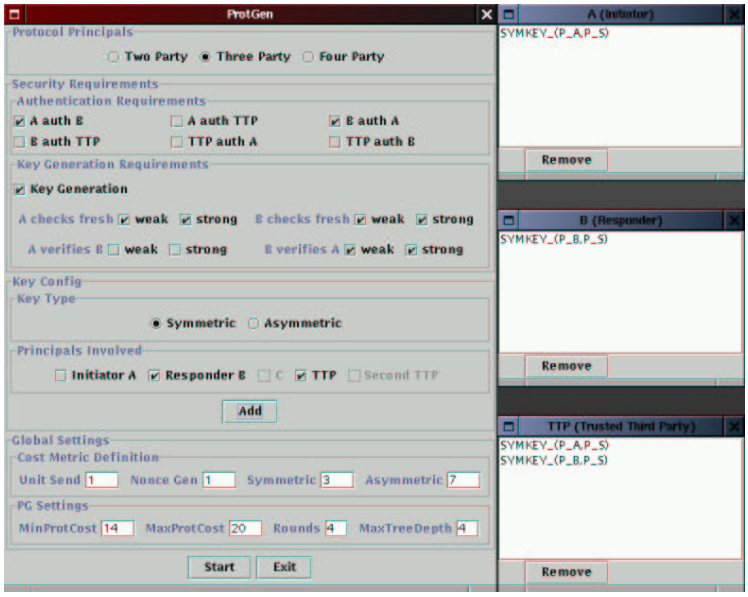

Figure 4. APG GUI

After analyzing $140,275$ protocols, the protocol generator output 473 candidate protocols. Using less than 10 minutes, Athena found two optimal protocols:

1. $A \rightarrow B : N_A, A$
   $B \rightarrow S : \{N_A, N_B, A\}_{K_{BS}}, B$
   $S \rightarrow A : \{N_A, N_B, A, B\}_{K_{AS}}$
   $A \rightarrow B : N_B$

2. $A \rightarrow B : N_A, A$
   $B \rightarrow S : \{N_A, N_B, A, B\}_{K_{BS}}, B$
   $S \rightarrow A : \{N_A, N_B, B\}_{K_{AS}}$
   $A \rightarrow B : N_B$

Interestingly, APG missed the following real optimal protocol:

$A \rightarrow B : N_A, A$
$B \rightarrow S : \{A, N_A, N_B\}_{K_{BS}}, B$
$S \rightarrow A : \{N_A, N_B, B\}_{K_{AS}}$
$A \rightarrow B : N_B$

The reason is because APG ignores the permutation of components in a concatenated message. So APG generated the following protocol instead and Athena pruned it because it is flawed.

$A \rightarrow B : N_A, A$
$B \rightarrow S : \{N_A, N_B, A\}_{K_{BS}}, B$
$S \rightarrow A : \{N_A, N_B, B\}_{K_{AS}}$
$A \rightarrow B : N_B$

Obviously, the reduction on ignoring the permutation of components in a concatenated message is a lossy reduction. But it is such a powerful reduction that we cannot remove it. Hence, we add a new rule to APG to solve this problem. Basically, the protocol generator would detect when the protocol contains two messages with the same format. When this occurs, the protocol generator will change the order of the concatenated components in one of the two messages to make it a different format and output a new protocol. After this fix, APG was able to find the real optimal protocol.

5.3 Authentication and key agreement with TTP using symmetric keys

The security properties for key agreement is much more complex, including key freshness, key confirmation and key secrecy. We first give a complete list of the specifications of different security properties. We then describe the optimal protocols found by APG with different sets of desired security properties.

Our list of formal specifications of the security properties are more complete than the ones in [SM95]. For example, we distinguish between weak key freshness and strong key freshness, while [SM95] does not consider the strong key freshness. In fact, the Yahalom protocol [CJ97] has been proven to have key freshness property by [DNL99]. But actually Athena proved that Yahalom only has the weak key freshness, not the strong key freshness property.

5.3.1 Specification of security properties

1. Authentication between the initiator and the responder

   If A completes the protocol run with B in a session with N_a and N_b, then there must be a unique responder protocol run with the same binding of parameters. The

same holds for B also.

$$\forall C.Init[A, B, S, N_a, N_b, *] \in C \implies Resp[A, B, S, N_a, N_b, *] \in C$$

The uniqueness of the protocol run of B is achieved by the freshness of N_b generated in the protocol run.

$$\forall C.Resp[A, B, S, N_a, N_b, *] \in C \implies Init[A, B, S, N_a, N_b, *] \in C$$

The uniqueness of the protocol run of A is achieved by the freshness of N_a generated in the protocol run.

2. Weak key freshness with server authentication

   If A completes a protocol run and accepts a session key K_{ab} in a session with B and nonce N_a, then there must be a unique protocol run of server S with K_{ab} for the session between A and B with nonce N_a. The same holds for B also.

$$\begin{aligned} \forall C.Init[A, B, S, N_a, *, K_{ab}] \in C &\implies \\ Srv[A, B, S, N_a, *, K_{ab}] \in C & \\ \forall C.Resp[A, B, S, *, N_b, K_{ab}] \in C &\implies \\ Srv[A, B, S, *, N_b, K_{ab}] \in C & \end{aligned}$$

   The uniqueness of the protocol run of S is achieved by the freshness of K_{ab} generated in the protocol run.

3. Strong key freshness with server authentication

   If A completes a protocol run and accepts a session key K_{ab} in a session with I and nonce N_a, then there must be a unique protocol run of server S with K_{ab} for the session between A and I with nonce N_a. The same holds for B also.

$$\begin{aligned} \forall C.Init[A, I, S, N_a, *, K_{ab}] \in C &\implies \\ Srv[A, I, S, N_a, *, K_{ab}] \in C & \\ \forall C.Resp[I, B, S, *, N_b, K_{ab}] \in C &\implies \\ Srv[I, B, S, *, N_b, K_{ab}] \in C & \end{aligned}$$

   The uniqueness of the protocol run of S is achieved by the freshness of K_{ab} generated in the protocol run.

   This property is stronger than the weak key freshness property, because in this property even when A (or B) is having a session with I, I cannot trick A (or B) to accept a key that is not freshly generated by S for the session.

4. Weak key confirmation between the initiator and the responder

   If A completes the protocol run and accepts the session key K_{ab} with B in a session with N_a, then there must be a responder protocol run with the same binding of parameters. The same holds for B also.

$$\begin{aligned} \forall C.Init[A, B, S, N_a, *, K_{ab}] \in C &\implies \\ Resp[A, B, S, N_a, *, K_{ab}] \in C & \\ \forall C.Resp[A, B, S, *, N_b, K_{ab}] \in C &\implies \\ Init[A, B, S, *, N_b, K_{ab}] \in C & \end{aligned}$$

5. Strong key confirmation between the initiator and the responder

   If A completes the protocol run and accepts the session key K_{ab} with B in a session with N_a, N_b, then there must be a unique responder protocol run with the same binding of parameters. The same holds for B also.

$$\forall C.Init[A, B, S, N_a, N_b, K_{ab}] \in C \implies Resp[A, B, S, N_a, N_b, K_{ab}] \in C$$

   The uniqueness of the protocol run of B is achieved by the freshness of N_b generated in the protocol run.

$$\forall C.Resp[A, B, S, N_a, N_b, K_{ab}] \in C \implies Init[A, B, S, N_a, N_b, K_{ab}] \in C$$

   The uniqueness of the protocol run of A is achieved by the freshness of N_a generated in the protocol run.

6. Key secrecy

   If A completes the protocol run and accepts the session key K_{ab} with B, then the intruder cannot learn K_{ab}. The same holds for B also.

$$\forall C.Init[A, B, S, N_a, N_b, K_{ab}] \in C \implies \neg F[K_{ab}] \in C$$

$$\forall C.Resp[A, B, S, N_a, N_b, K_{ab}] \in C \implies \neg F[K_{ab}] \in C$$

   where $F[K_{ab}]$ is an attacker strand with K_{ab} as its term, meaning that the attacker can get K_{ab} in cleartext.

5.3.2 New protocols

We did experiments using APG to find optimal protocols which satisfy three different sets of the security properties as shown in Table 1. The protocol generator output more than $11,000$ candidate protocols. Athena took less than two hours to analyze them. We list our findings as follows. Property-set 1 is the strongest among the three. Protocol-set S1 has a higher cost than S2 and S3 while S2 and S3 have the same costs. Interestingly, Yahalom protocol, shown in Figure 5, has the same costs as protocol-set S1 but only achieves the same security properties as protocol-set S2. In fact, the Yahalom protocol was generated by the protocol generator but Athena found a counterexample of the strong

Yahalom :

$$A \to B : N_A, A$$
$$B \to S : \{N_A, N_B, A\}_{K_{BS}}, B$$
$$S \to A : \{SK_{AB}, A\}_{K_{BS}}, \{SK_{AB}, N_A, N_B, B\}_{K_{AS}}$$
$$A \to B : \{SK_{AB}, A\}_{K_{BS}}, \{N_B\}_{SK_{AB}}$$

Figure 5. Yahalom Protocol [CJ97]

key freshness property in Yahalom. The counterexample simply shows that if B is a responder in a protocol run with the intruder I as the initiator, I can easily make B to accept an old key as the "fresh " session key generated by S. In some applications, if the session key is also used for some other purposes, e.g. the hash of the session key is used as a transaction ID (of course, such a usage of session keys might not be recommended at the first place), the strong key freshness could be important.

- Protocol-Set S1 satisfies the first set of the security properties:

  1.
  $$A \to B : N_A, A$$
  $$B \to S : \{N_A, N_B, A\}_{K_{BS}}, B$$
  $$S \to A : \{SK_{AB}, N_B\}_{K_{BS}}, \{SK_{AB}, N_A, N_B, B\}_{K_{AS}}$$
  $$A \to B : \{SK_{AB}, N_B\}_{K_{BS}}, \{N_B\}_{SK_{AB}}$$

  2.
  $$A \to B : N_A, A$$
  $$B \to S : \{N_A, N_B, A\}_{K_{BS}}, B$$
  $$S \to A : \{SK_{AB}, N_B\}_{K_{BS}}, \{SK_{AB}, N_A, N_B, B\}_{K_{AS}}$$
  $$A \to B : \{SK_{AB}, N_B\}_{K_{BS}}, \{\{SK_{AB}, N_B\}_{K_{BS}}\}_{SK_{AB}}$$

  Note that APG considers the two protocols have the same cost because in the last message in the second protocol, $\{SK_{AB}, N_B\}_{K_{BS}}$ is just a bit string that A received from S and therefore it is equivalent cost-wise with N_B in the last message of the first protocol from A's point of view.

- Protocol-Set S2 satisfies the second set of the security properties:

  1.
  $$A \to B : N_A, A$$
  $$B \to S : \{N_A, N_B, A\}_{K_{BS}}, B$$
  $$S \to A : \{SK_{AB}\}_{K_{BS}}, \{SK_{AB}, N_A, N_B, B\}_{K_{AS}}$$
  $$A \to B : \{N_B\}_{SK_{AB}}, \{SK_{AB}\}_{K_{BS}}$$

  2.
  $$A \to B : N_A, A$$
  $$B \to S : \{N_A, N_B, A\}_{K_{BS}}, B$$
  $$S \to A : \{SK_{AB}, N_A, N_B, B, \{SK_{AB}\}_{K_{BS}}\}_{K_{AS}}$$
  $$A \to B : \{N_B\}_{SK_{AB}}, \{SK_{AB}\}_{K_{BS}}$$

- Protocol-Set S3 satisfies the third set of the security properties:

$$A \to B : N_A, A$$
$$B \to S : \{N_A, N_B, A\}_{K_{BS}}, B$$
$$S \to A : \{SK_{AB}, N_B\}_{K_{BS}}, \{SK_{AB}, N_A, N_B, B\}_{K_{AS}}$$
$$A \to B : N_B, \{SK_{AB}, N_B\}_{K_{BS}}$$

6 Conclusion

As the protocol complexity increases, the protocol space grows exponentially. To analyze complex authentication and key-agreement protocols with three parties and four rounds, we needed to add aggressive state reduction techniques to the protocol generator. The increased number of protocols also put a heavy burden on Athena, our protocol screener. We also extended Athena with new pruning theorems and a probabilistic method to pick goal orderings which improves the overall efficiency and worst-case performance of Athena.

It is clear that the task of APG is increasingly difficult as we attack more complex protocols. The reward, however, is to find novel and highly efficient protocols with new properties. Especially in the second experiment we did, APG found different optimal protocols for three different sets of security properties. The difference in these protocols are so subtle that it is difficult for a person to invent them for each different security property. This nicely illustrates the necessity and usefulness of APG.

Acknowledgements

We would like to thank Joshua Guttman and Javier Thayer for introducing the authentication test theorems to us. We would also like to thank Doug Tygar and George Necula for his encouragement and advice on this work.

References

[CGP99] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature. http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz, 1997. Version 1.0.

[DNL99] Ben Donovan, Paul Norris, and Gavin Lowe. Analyzing a library of security protocols using casper and fdr. In *Workshop on Formal Methods and Security Protocols*, 1999.

[DY89] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1989.

[GT00] Joshua D. Guttman and F. Javier Thayer. Authentication tests. In *To appear in Security and Privacy Symposium'00*, 2000.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In

	A auth B	B auth A	Weak key freshness (A&B)	Strong key freshness (A&B)	Weak key conf (A to B)	Key secrecy	Optimal protocols
1	X	X	X	X	X	X	S1
2	X	X	X		X	X	S2
3	X	X	X	X		X	S3

Table 1. Three sets of security properties

Tools and Algorithms for the Construction and Analysis of Systems, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[Low97] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 31–43, 1997.

[Mea95] C. Meadows. Formal verification of cryptographic protocols: A survey. In *Advances in Cryptology - Asiacrypt '94*, volume 917 of *Lecture Notes in Computer Science*, pages 133–150. Springer-Verlag, 1995.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[PS00] Adrian Perrig and Dawn Song. A first step towards the automatic generation of security protocols. In *Network and Distributed System Security Symposium*, February 2000.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence, 1995.

[SBP00] Dawn Song, Sergey Berezin, and Adrian Perrig. Athena, a new efficient automatic checker for security protocols. *Submitted to Journal of Computer Security*, 2000.

[SM95] Paul Syverson and Catherine Meadows. Formal requirements for key distribution protocols. In *Advances in Cryptology – EUROCRYPT '94, Lecture Notes in Computer Science*, volume 950, 1995.

[Son99] Dawn Song. Athena: An automatic checker for security protocol analysis. In *Proceedings of the 12th Computer Science Foundation Workshop*, 1999.

[THG98] F.Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of 1998 IEEE Symposium on Security and Privacy*, 1998.

[WL93] T. Y. C. Woo and S. S. Lam. A semantic model for authentication protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1993.

A The strand space model

Actions. The set of actions *Act* that principals can take during an execution of a protocol include external actions such as *send* and *receive*, and user-defined internal actions such as *debit*, *credit*, etc.. In the rest of the paper, we will only use *send* and *receive* for simplicity.

Events. An *event* is a pair $\langle$*action, argu*$\rangle$, where *action* is in *Act*, and *argu* is in (A) and is the argument of the action. For simplicity, we denote $\langle$*send, a*$\rangle$ and $\langle$*receive, a*$\rangle$ respectively as *signed terms* $\langle +a \rangle$ and$\langle -a \rangle$. We represent the set of finite sequences of signed terms as $(\pm A)^*$.

Strands and Strand Spaces. A *protocol* defines the sequence of events for each role of the participant. A *strand* represents a sequence of actions of an instance of a role.

A strand space is a set Σ with a trace mapping tr: $\Sigma \rightarrow (\pm A)^*$.

1. A *node* is a pair $\langle s, i \rangle$, with $s \in \Sigma$ and *i* an integer satisfying $1 \leq i \leq$ length(tr(s)). We say $n = \langle s, i \rangle$ belongs to the strand *s*, denoted as $n \in s$. Clearly, every node belongs to a unique strand. The set of nodes is denoted by N.

2. If $n = \langle s, i \rangle \in N$, then index$(n) = i$ and strand$(n) = s$. If $(tr(s))_i = \langle \sigma a \rangle$, where σ is one of the symbols $+, -$, then term$(n) = a$.

3. If $n_1, n_2 \in N$, then $n_1 \rightarrow n_2$ means that term$(n_1) = +a$ and term$(n_2) = -a$. This represents that n_1 sends a message *a* and n_2 receives the message.

4. If $n_1, n_2 \in N$, then $n_1 \Rightarrow n_2$ means that n_1, n_2 occur in the same strand with index$(n_2) =$ index$(n_1) + 1$. This represents an event n_1 followed immediately by n_2 in the same strand.

5. A term *t originates* from a node $n \in N$ *iff sign(n)* $= +$; $t \sqsubseteq$ *term(n)*; and whenever n' precedes *n* on the same strand, $t \not\sqsubseteq$ *term(n')*.

6. A term *t uniquely-originates* from node *n iff t originates* on a unique $n \in N$. Nonces and other freshly generated terms are usually uniquely-originated.

We will also use N to refer to the directed graph (N, E) whose vertices are nodes and $E = (\rightarrow \cup \Rightarrow)$ is the set of edges that combines both types of relations $n_1 \rightarrow n_2$ and $n_1 \Rightarrow n_2$.

Bundles. A bundle represents the protocol execution under some configuration.

A bundle $C = (N_C, E)$ is a subgraph of N, where $E \subseteq (\rightarrow \cup \Rightarrow)$ is the set of the edges and $N_C \subseteq N$ is the set of nodes incident with the edges in E, and the following properties hold:

- C is non-empty and finite;
- If $n_1 \in C$ and *sign*$(n_1) = -$, then there is a unique n_2 such that $n_2 \rightarrow n_1 \in C$;
- If $n_1 \in C$ and $n_2 \Rightarrow n_1$, then $n_2 \Rightarrow n_1 \in C$;
- C is acyclic.

We say a strand $s \in C$ if for every node $n \in s$, $n \in C$.

Causal Precedence. Let S be a strand space, nodes n_1,$n_2 \in$ S. Define $n_1 \preceq_S n_2$ *iff* there is a sequence of zero or more edges of type $\rightarrow$ and $\Rightarrow$ leading from n_1 to n_2 in S. The relation $\preceq_S$ expresses a *causal precedence*.

Lemma A.1. *Suppose C is a bundle, then* $\preceq_C$ *is a partial order, i.e. a reflexive, antisymmetric, transitive relation. Every non-empty subset of the nodes in C has* $\preceq_C$*-minimal members.*

The proof of this lemma can be found in [THG98].

B The logic

B.1 Syntax

The syntax of terms consists of *strand constants* $(s, s_1, \dots)$ and *bundle variables* $(C, \dots)$. A strand constant may represent a partial strand, and in particular, a single node can be considered as a strand constant.

Propositional formulas are defined as follows:

- $s \in C$ is an (atomic) propositional formula;
- $\neg f_1$ and $f_1 \wedge f_2$ are propositional formulas if f_1 and f_2 are propositional formulas.

Finally, well-formed formulas (wffs) are:

- f, $\neg F_1$, $F_1 \wedge F_2$;
- $\forall C.\ f$, where f is a propositional formula, which doesn't contain any other variable than C.

Here f is a propositional formula, F_1 and F_2 are wffs, and C is a bundle variable.

Notice, that in a wff $\forall C.\ f$, the formula f needs to be propositional, and cannot contain any other variables than C. In addition, we allow only negative occurrences of atomic formulas in f that reference penetrator strands. An occurrence of an atomic formula ϕ in f is called *negative* if ϕ appears under the scope of odd number of negations. We also use the obvious abbreviations:

$$\begin{array}{rcl} f_1 \vee f_2 & \equiv & \neg(\neg f_1 \wedge \neg f_2) \\ f_1 \implies f_2 & \equiv & \neg f_1 \vee f_2 \\ f_1 \iff f_2 & \equiv & (f_1 \implies f_2) \wedge (f_2 \implies f_1) \\ \exists C.f & \equiv & \neg\forall C.\neg f \end{array}$$

B.2 Semantics

Let the set of nodes be N. For a given protocol p, define the set of all strands for this protocol (both regular and penetrator ones) as Σ_p; its execution traces form a set of bundles $\mathcal{D}_p$. A *model* M_p for a given protocol p is a tuple $M_p = (N,\ \Sigma_p,\ \mathcal{D}_p,\ \mathcal{I})$, where $\mathcal{I}$ is the interpretation of strand constants and bundle variables. We write $M' = M[\mathcal{I}(C) \leftarrow c]$ for some bundle c to denote a new model M' which is identical to M except that $\mathcal{I}'(C) = c$. The semantics of formulas in our logic is the following:

- $\mathcal{I}(s) \in \Sigma_p$ and $\mathcal{I}(C) \in \mathcal{D}_p$ are a strand and a bundle respectively, that are assigned to the constant s and the variable C by the interpretation $\mathcal{I}$.
- $M \models s \in C$ *iff* $\mathcal{I}(s) \in \mathcal{I}(C)$.
- If f is a propositional formula or a wff, then $M \models \neg f$ *iff* $M \not\models f$.
- If f_1 and f_2 are propositional formulas or wffs, then $M \models f_1 \wedge f_2$ *iff* $M \models f_1$ and $M \models f_2$.
- $M \models \forall C.f$ *iff* $\forall c \in \mathcal{D}_p.\ M[\mathcal{I}(C) \leftarrow c] \models f$.

B.3 Specify security properties in the logic

Our logic can express a variety of security properties, including ones for authentication, secrecy, and electronic commerce. In this paper we mainly focus on the authentication and secrecy. We use similar ways for representing security properties as in [THG98], however, we formulate them using well-formed formulas in our logic.

Authentication.

Gavin Lowe [Low97] proposed agreement properties for authentication protocols. A protocol guarantees an *agreement property* for a participant B (e.g. acting as a responder) for a certain vector of parameters $\vec{x}$, if each time the principal B completes a run of the protocol as a responder using $\vec{x}$, supposedly with A, then there is a unique run of the protocol with the principal A initiating a session with the same parameters $\vec{x}$, supposedly with B.

A weaker *non-injective agreement* does not ensure uniqueness, but requires only that each time a principal B completes a run of the protocol as responder using $\vec{x}$, supposedly with A, then there is a run of the protocol with the principal A as the initiator using $\vec{x}$, supposedly with B.

The non-injective agreement property can be specified in our logic as:

$$\forall C. Resp(\vec{x}) \in C \Longrightarrow Init(\vec{x}) \in C,$$

where $Resp(\vec{x})$ and $Init(\vec{x})$ are the responder and the initiator strands instantiated with parameters $\vec{x}$. For example, in the *NSL* protocol, the non-injective agreement property can be specified as

$$\forall C.\, Resp[A, B, N_a, N_b] \in C \Longrightarrow Init[A, B, N_a, N_b] \in C.$$

Here $\vec{x} = [A, B, N_a, N_b]$. Because of the freshness of the nonces generated in the protocol run, usually the agreement property can be proven after the non-injective agreement property is proven, with the argument that there cannot be two strands $Init(\vec{x}) \in C$ since the nonces in $Init(\vec{x})$ are uniquely originated from only one strand. Namely, in the *NSL* protocol N_a is uniquely-originated in the strand $Init[A, B, N_a, N_b]$.

Secrecy.

A value v is *secret* in a strand set $\mathcal{W}$ if for every bundle C that contains $\mathcal{W}$ there is no way for the intruder to receive v in cleartext; that is, the strand $\mathrm{F}[v]$ does not appear in any C:

$$\forall C.\, \mathcal{W} \subseteq C \implies \neg \mathrm{F}[v] \in C.$$

For example, when $\mathcal{W}$ contains only a single responder strand $Resp(\vec{x})$, we can specify the secrecy property as:

$$\forall C.\, \{Resp(\vec{x})\} \subseteq C \implies \neg \mathrm{F}[v] \in C.$$

C Proofs of Pruning Theorems

Lemma C.1. *Let C be a normal bundle with $n \Rightarrow^+ n'$ in $\mathcal{C}$. If a uniquely originates from n, and in $term(n)$, a is only in subterm $t_1 \sqsubset n$ and $a \ll_K t_1$, and $a \sqsubset n', t_1 \not\sqsubset n'$. Then there exists a transforming edge $m \Rightarrow^+ m'$ for a and $t_1 \sqsubset m$, and there is a new subterm $t_2 \sqsubset m', a \sqsubset t_2$ and $t_1 \not\sqsubset t_2$.*

Moreover, if $K^{-1} \notin \mathsf{P}$, then there exists a transforming edge as described above which lies on a regular strand.

Proof sketch. Consider the set of nodes Φ in $\mathcal{C}$ as the set of nodes satisfying the following property: for each node $n_i \in \Phi, a \sqsubset n_i, t_1 \not\sqsubset n_i$. Because $n' \in \Phi$, Φ is not empty. By Proposition 4.1, there is at least a $\preceq$-minimal member, denoted as n_0. Because $n \notin \Phi, n \neq n_0$.

Because a uniquely originates from n, there must exist an edge $n_0' \Rightarrow^+ n_0$ in $\mathcal{C}$ and $a \sqsubset n_0'$. Thus, $n_0' \notin \Phi$. Therefore, $t_1 \sqsubset n_0'$. By definitions, $n_0' \Rightarrow^+ n_0$ is a transforming edge for a.

Now we need to prove that if $K^{-1} \notin \mathsf{P}$, then $n_0' \Rightarrow^+ n_0$ lies on a regular strand.

Suppose otherwise: Then $n_0' \Rightarrow^+ n_0$ lies either on a D- or and E-strand. First, suppose $n_0' \Rightarrow^+ n_0$ lies on a D-strand, then the key edge of this D-strand cannot be K^{-1} because we have assumed that $K^{-1} \notin \mathsf{P}$. Suppose $n_0' = \{|h|\}_{K'}$, $K' \neq K^{-1}$. So $t_1 \sqsubset h$, which contradicts that $t_1 \not\sqsubset n_0$. So $n_0' \Rightarrow^+ n_0$ cannot lie on a D-strand.

Second, suppose that $n_0' \Rightarrow^+ n_0$ lies on an E-strand, then $n_0 = \{|h'|\}_{K''}$. So $t_1 \not\sqsubset h'$, which contradicts that $t_1 \sqsubset n_0'$. So $n_0' \Rightarrow^+ n_0$ cannot lie on an E-strand. strand.

Hence, $n_0' \Rightarrow^+ n_0$ lies on a regular strand. □

Lemma C.2. *Let C be a bundle with $n \Rightarrow^+ n' \in \mathcal{C}$. If a uniquely originates from n, and in $term(n')$, $a \ll_K t_1$ and $t_1 \sqsubset n', t_1 \not\sqsubset n$. Then there exists a transforming edge $m \Rightarrow^+ m'$, where $t_1 \sqsubset m', t_1 \not\sqsubset m, a \sqsubset m$.*

Moreover, if $K \notin \mathsf{P}$, then there exists a transforming edge as described above which lies on a regular strand.

Proof sketch. Define the set of nodes Φ in $\mathcal{C}$ as the set of nodes containing t_1 as a subterm: for each node $n_i \in \Phi, t_1 \sqsubset n_i$. Because $n' \in \Phi$, Φ is not empty. By Proposition 4.1, there is at least a $\preceq$-minimal member, denoted as n_0. Because $n \notin \Phi, n \neq n_0$.

Because a uniquely originates from n, there must exist an edge $n_0' \Rightarrow^+ n_0$ in $\mathcal{C}$ and $a \sqsubset n_0'$. Because $n_0' \notin \Phi$, $t_1 \not\sqsubset n_0'$. By definitions, $n_0' \Rightarrow^+ n_0$ is a transforming edge for a.

Now we need to prove that if $K \notin \mathsf{P}$, then $n_0' \Rightarrow^+ n_0$ lies on a regular strand.

Suppose otherwise: Then $n_0' \Rightarrow^+ n_0$ lies either on a D- or and E-strand. First, suppose $n_0' \Rightarrow^+ n_0$ lies on a D-strand, then $n_0' = \{|h|\}_{K'}$. So $t_1 \sqsubset h$, which contradicts that $t_1 \not\sqsubset n_0'$. So $n_0' \Rightarrow^+ n_0$ cannot lie on a D-strand.

Second, suppose that $n_0' \Rightarrow^+ n_0$ lies on an E-strand, then $n_0 = \{|h'|\}_{K''}$, $K'' \neq K$. So $t_1 \sqsubset h'$, which contradicts that $t_1 \not\sqsubset n_0'$. So $n_0' \Rightarrow^+ n_0$ cannot lie on an E-strand.

Hence, $n_0' \Rightarrow^+ n_0$ lies on a regular strand. □

D Estimate of the size of the protocol space

This estimate is based on the average number of messages that a principal can generate in a given round of the protocol. We observed that in a four-round three-party authentication and key agreement protocol (both A and B share a secret key with S), A generates over 100 different messages to B, which in turn generates 500 distinct messages on average and forwards them to S. Since S has two encryption keys, the number of messages it can generate is much higher and the average is 30000. In the final round both A and B can generate around 500 messages. The product of these average number of messages is on the order of 10^{12}. We validated this estimate with a different message flow (i.e. $A \rightarrow S \rightarrow B \rightarrow A$) and we reached the same conclusion. We would like to point out that we made these measurements during a protocol generation run which we used to generate the candidate protocols for our study. The maximum protocol cost was limited to 19 and the message depth was limited to 4. The above estimate therefore approximates the size of the searched space.