# A First Step towards the Automatic Generation of Security Protocols*

Adrian Perrig
perrig@cs.berkeley.edu

Dawn Song
dawnsong@cs.berkeley.edu

Computer Science Department
Carnegie Mellon University and University of California, Berkeley

## Abstract

*This paper describes automatic protocol generation (APG for short), a novel mechanism to generate security protocols automatically. With APG, the protocol designer inputs the specification of the desired security properties and the system requirements. The system requirements include a metric function which specifies the cost or overhead of protocol primitives, which defines an ordering over protocols with respect to the metric function. Based on this ordering, APG explores the protocol space and outputs the correct protocol which has minimal cost with respect to the metric function, as well as satisfies the security properties and system requirements.*

*The APG approach has several advantages over the current protocol design process. It is fully automatic, and hence, more efficient than a manual process. The protocols generated by APG offer higher confidence, because they are verified by a powerful protocol analyzer. Another significant advantage is that, because APG search through the protocol space in the order of increasing cost with respect to the metric function, APG generates correct protocols with minimal cost which ideally suit the system requirements. Furthermore, APG is flexible in the sense that it can handle different security properties and different system requirements.*

*To gain experience with APG, we conduct a case study on the automatic generation of two-party, mutual authentication protocols. In one experiment, APG generates authentication protocols that are simpler than the standard protocols documented in the literature (i.e., ISO standards [Int93]). In another experiment, the automatic protocol generation generates different protocols with minimal cost for varying requirements, hence demonstrating its capability to produce high quality protocols.*

## 1 Introduction

The exponential growth of the Internet and electronic commerce brings not only prosperity, but also vulnerability. Numerous attacks pose a real challenge to different aspects of security mechanisms. Among these security mechanisms, security protocols play an essential role. They use cryptographic primitives as building blocks to achieve security goals such as authentication, confidentiality, and integrity. New applications and systems eagerly demand security protocols suitable to their system requirements. Unfortunately, designing security protocols is a delicate task and experience shows that security protocols are notoriously hard to get right [BAN89, Low96]. This naturally raises

*This research was done while the authors were at Carnegie Mellon University. This publication was supported in part by Contract Number 102590-98-C-3513 from the United States Postal Service. The contents of this publication are solely the responsibility of the author and do not necessarily reflect the official views of the United States Postal Service.

the question whether the current security protocol design process is satisfactory. If the answer is no, how can we improve it?

The current process of finding a solution is usually ad-hoc and involves little formalism, and almost no mechanical assistance. Such a design process is not satisfactory for the following reasons:

- Error-prone. Security protocols are intricate and attackers are powerful.
  - If the designer has limited experience, it is likely that the security protocol is flawed. 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.
  - Due to the lack of formalism and mechanical assistance, manually designed protocols often suffer from the fact that they contain undocumented assumptions. Since the implementation might not respect these assumptions the resulting protocols might be insecure/flawed.
  - Without proofs, these protocols cannot give any guarantee on satisfying the desired security properties, and hence degrade the level of confidence.
- Non-optimal. Such designed protocols may contain unnecessary operations. Simply because the designer cannot search a large number of candidates, she may not find the optimal protocol for the given system requirements. Moreover, conservative designers might put unnecessary operations just to play safe.
- Inefficient and Expensive. The current design process is often slow. It can be a serious bottleneck of the project and severely delay product development. If the protocol is flawed, high costs might incur, due to forced redesign, update plans, or liability claims.

In this paper, we present *automatic protocol generation*, APG for short, a mechanism which addresses these shortcomings. With automatic protocol generation, the protocol designer specifies the desired security properties, such as authentication and secrecy, and system requirements, i.e., 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.

Our approach provides several advantages:

- Automatic. The protocol designer specifies the security properties and the system requirements. The remaining process is fully automatic.
- High Confidence. Since the input specifications are written in a well-defined specification language and the automatic protocol generation is fully mechanical, there are no hidden assumptions, in contrast to the manual design process. The protocol screener has a powerful engine which can automatically generate a proof if a protocol is correct, or a counterexample otherwise.
- High Quality. 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 the metric function hence suits the system requirements the best.
- Flexible. The approach works for different security properties, system requirements, and attacker models.

The remainder of this paper is organized as follows. We begin with an overview of the general framework and requirements of APG. Next, we present a case study of automatic generation of authentication protocols. In our case study, we show how we deal with the arising difficulties to make APG feasible. Following the case study, we discuss some of the limitations of APG, as well as the insights and results of the case study and the interesting directions for further research. Finally, we summarize the contributions of this paper.

# 2 General Framework and Requirements for APG

In this section, we first give an overview of the general framework of APG. Then, we state the requirements of each component and give more details about our design and implementation of these components.

## 2.1 Overview

At a high level, APG is, as Figure 1 shows, a pipeline composed of an automatic protocol generator and an automatic protocol screener. In general, the process of APG has four stages. First, the protocol designer specifies the desired security properties and system requirements as input. Second, the protocol generator searches through the protocol space and generates candidate protocols that satisfy the system requirements. Third, the protocol screener analyzes the candidate protocols. Finally, the flawed protocols are discarded and the correct ones that satisfy the desired security properties are output.

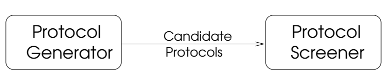


Figure 1: APG overview

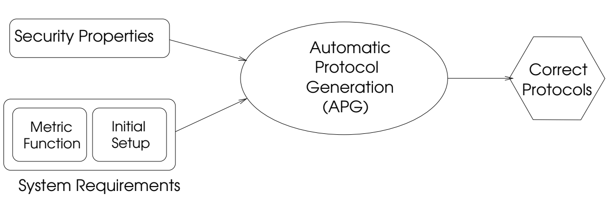


Figure 2: APG data flow

## 2.2 Input Specification

Our specification language defines the security properties, including authentication, secrecy and other properties related to electronic commerce. 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 be monotonically increasing as the protocol complexity is increasing. 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.

## 2.3 Notation

We use the following notation to describe protocols in this paper.

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

$A, B$ are the principals

$N_A$ is a nonce generated by $A$

$K_A$ denotes $A$'s public key

$K_A^{-1}$ denotes $A$'s private key

$K_{AB}$ denotes the secret (symmetric) key which is shared between $A$ and $B$

$\{M\}_{K_B}$ is the encryption of message $M$ with $B$'s public key

## 2.4 Protocol Representation

A protocol defines a sequence of actions of the participating parties. The actions include sending and receiving messages. Messages are defined by the following grammar. The grammar can be easily extended if needed.

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

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.

## 2.5 The Protocol Generator

The purpose of the protocol generator is to generate candidate protocols that satisfy the specified system requirements. Intuitively, the protocol space is infinite. Hence, we need a way to limit the number

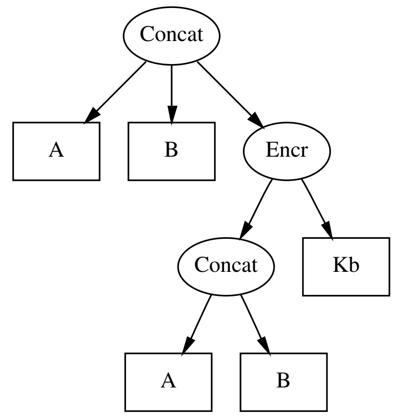


Figure 3: Example of a message tree for the message: $A, B, \{A, B\}_{K_B}$

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 can stop. Otherwise, we increase the cost threshold and generate more protocols.

It is intuitive and our experiments confirmed that the number of protocols generated is exponential in the value of the cost threshold. Hence, the protocol generator can easily generate millions of protocols. Since elaborate protocol verification takes on the order of 1 second per protocol, it would be impractical for the protocol screener to analyze all of these protocols. To make APG practical we use efficient reduction techniques and heuristics to prune invalid candidate protocols early to reduce the number of candidate protocols passed to the protocol screener. Most of the generated protocols contain severe security flaws which can be detected with a simple and more efficient verification algorithm. Each security property that is supported by the system is therefore accompanied by a pruning algorithm (which efficiently discards most severely flawed protocols) and a verification condition for the screener. We give more detail about the pruning algorithms in the case study in Section 3.

### 2.6 The 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. The protocol screener is sound if when it claims that a protocol satisfies certain security properties, it is true that the protocol really does satisfy these properties. 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.

There are several existing tools for semi-automatic and automatic protocol analysis, such as the NRL Analyzer [Mea94], the Interrogator Model [Mil95], FDR [Low96], Mur$\varphi$ [MMS97], and Brutus [CJM98]. Athena is a recently introduced checker for security protocol analysis [Son99]. Comparing to other existing automatic tools, Athena has the following two main advantages:

- Athena has the ability to analyze protocol executions with any arbitrary configuration. Most of other existing automatic tools can only reason about finite state space, which implies that they can only analyze protocol executions with certain configurations, such as two initiators and two responders. In contrast, when Athena terminates, it provides a proof that a protocol satisfies its specified property under any arbitrary protocol configuration, or it generates a counterexample if the property does not hold.

- Athena exploits many state space reduction techniques which result in a highly reduced state space.

For these reasons, we choose to use Athena as the protocol screener. During this project, Athena has verified tens of thousands of protocols and has established itself as a highly efficient and robust tool for automatic protocol analysis.

The following is a brief overview of how Athena works. Athena uses an extension of the recently proposed Strand Space Model [THG98] to represent protocol execution. Athena incorporates a new logic that can express security properties including authentication, secrecy and properties related to electronic commerce. An automatic procedure enables Athena to evaluate well-formed formulae in this logic. For a well-formed formula, if the evaluation procedure terminates, it will generate a counterexample if the formula is false, or provide a proof if the formula is true. Even when the procedure does not terminate when we allow any arbitrary configurations of the protocol execution, (for example, any number of initiators and responders), termination could be forced by bounding the number of concurrent protocol runs and the length of messages, as is done in most existing automatic tools.

Athena also exploits several state space reduction techniques. Powered with techniques such as backward search and symbolic representation, Athena naturally avoids the state space explosion problem commonly caused by asynchronous composition and symmetry redundancy. Athena also has the advantage that it can easily incorporate results from theorem proving through unreachability theorems. By using the unreachability theorems, it can prune the state space at an early stage, hence, further reduce the state space explored and increase the likely-hood of termination. These techniques dramatically reduce the state space that needs to be explored.

## 3 Case Study: Automatic Generation of Authentication Protocols

In order to gain preliminary experience with APG, we perform a case study with automatic generation of two-party mutual authentication protocols. Authentication protocols are among the most widely used and intensely studied security protocols. Their complexity is suitable for an initial case study, and they are known to be notoriously difficult to design correctly and hence a good challenge [BAN89, Low96].

We use the agreement properties proposed by Gavin Lowe for authentication protocols as the formal definition of the authentication property [Low97]. A protocol guarantees a participant $B$ *agreement* for a certain binding $\vec{x}$ if each time a 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 $A$ as initiator using $\vec{x}$, supposedly with $B$.

In this section, we first discuss the assumptions we make in the case study. Then, we explain the difficulties we encountered in the case study and describe our enhancement techniques to overcome the difficulties. Finally, we summarize the experiment results and our findings.

### 3.1 Assumptions

We initially analyzed how many first messages the initiator $A$ can send, with a given depth of the message tree. Subsequently, we refer to the two protocol principals as the *initiator* $A$ and the *responder* $B$. Initially the initiator knows the following atomic message components: $A, B, K_A, K_A^{-1}, K_B, N_A$. With a message depth of 4 the initiator can generate about one thousand messages; and with a depth of 6, it can generate about 8 million messages. If a two-party mutual authentication protocol uses three rounds, considering 1000 possible messages in each round would leave us with $10^9$ protocols. A protocol screener which analyzes 20 protocols per second, would take over 1 year.

After the initial estimate, we decide to make certain assumptions to keep the protocol space small. We do not intend to prove that these assumptions do not eliminate the potential optimal protocol. But we believe these assumptions are intuitively reasonable. We list all the assumptions we make in the case study as the following:

- Message components are typed. This allows any participant to distinguish a nonce from a principal name, for example.
- In any concatenated message, there are no redundant message components, i.e., $N_A, N_A$.
- No initial keys are sent in a message, since it does not make sense to send a private key, and we assume every principal knows all the public keys. Session keys generated during the protocol run do not fall into this category of authentication protocols.
- We assume that the initiator's name needs to be in the first message in a understandable format to the responder, so that the responder knows who to reply to. (This assumption might not be necessary when the initiator and the responder have a link between them that is only used to communicate between the two parties, although this case is not very general so we do not consider it here.)
- We do not consider permutations of the message components of a concatenated message.

The last point reduces the protocol space tremendously. Unfortunately, this optimization might result in missing a correct protocol. This case can occur if the generated protocol is vulnerable to a specific replay attack, where a message of round $i$ can be replayed for another message of round $j$ ($i \neq j$). In our current implementation of APG, the protocol is rejected and no permutation of the message components is considered. In the future, however, we could detect this case and try to repair the protocol through a message reordering.

### 3.2 Adding the Pruning Algorithm to the Protocol Generator

As we mentioned before, a naïve approach generates a large number of uninteresting candidate protocols. In Table 1 we show that the naïve approach generates tens of thousands of candidate protocols in the real experiments. Generating a large number of flawed candidate protocols risks to render APG impractical, since the running time of the protocol screener would be prohibitive. To deal with this problem, we define a pruning algorithm for each security property, which efficiently prunes the majority of the flawed protocols. This pruning algorithm can either operate on the message level or on the protocol level. A secrecy property, for instance, can be verified on the message level, since the secret value cannot be disclosed publically in any message. In the case of the authentication property, however, the pruning algorithm works on the protocol level, since it is difficult to define a message-level pruning algorithm which does not violate completeness (i.e. preserves correct protocols). To quickly discard flawed authentication protocols, we use an intruder module which checks for impersonation and simple replay attacks. As Table 1 shows, these two mechanisms reduce 98 percent of the candidate protocols in real experiments.

#### Impersonation attempt

We use two intruders to attack each protocol. The intruder $I_I$ tries to impersonate the initiator $A$, and the other intruder $I_R$ attempts to impersonate the responder $B$. Both intruders have the public keys of all the principals in their initial information. If symmetric encryption is used, the intruders certainly do not obtain any of the secret keys. Then, $I_I$ tries to start a session with $B$ impersonating as $A$. If $I_I$ can get $B$ to finish his session believing it is talking to $A$, then the protocol is simply broken. Similarly we can check whether $I_R$ can impersonate as $B$ to finish a session with $A$. The purpose for this attack is simply to check whether correct and necessary encryptions are used. It does not involve any replay attack and multiple protocol run and hence is very efficient.

#### Preventing simple replay attacks

Now we look at a simple replay attack. After a protocol session of an initiator $A$ and a responder $B$, an intruder $I$ stores all the messages sent in the session. Then, $I$ tries to re-send the packets to $B$ to impersonate as $A$. If $I$ can trick $B$ to finish its session believing it is talking to $A$, then the protocol is flawed and is discarded. Similarly, $I$ can launch the simple replay attack to $A$ as well. The purpose for this attack is just to check whether nonces are used in a correct way. The intruder does not try to encrypt or decrypt messages or alter the received messages and hence is very efficient.

### 3.3 Testing and Improving the Protocol Screener

There are two main challenges for the protocol screener. First, the protocol screener needs to be sound. If the protocol screener outputs a flawed protocol, the automatic protocol generation is not trustworthy. Second, the protocol screener has to be efficient because potentially the protocol generator could generate thousands of candidate protocols.

Hence, one point worth mentioning is that this research also serves a second purpose: a real test for Athena. As far as shown in previous literature, most of the automatic tools for protocol analysis have only been tested with a handful testing protocols and the testing protocols are mainly existing human-designed protocols. The candidate protocols generated by the protocol generator are purely machine-generated from a large protocol space, and hence, could potentially contain more misbehavior and difficult errors. Therefore, this is a good test for the soundness of both design and implementation of Athena. During the experiments, the protocol generator generates thousands of candidate protocols. Therefore, this is also a good test for the performance of Athena. As a sanity check, we also manually analyzed the protocols which Athena proved correct. We were not able to find errors in these protocols. On average, Athena verifies around 5 protocols per second, based on a 400 MHz Pentium II Linux workstation.

### 3.4 Summary of the Experiment Results

#### Effectiveness of the Reduction Techniques

In this experiment, we use a simple, linear metric function. Each operation has a unit-cost. The cost value of a protocol is the sum of the costs of all the protocol operations and components. We choose UNIT_ELEMENT_COST$= 1$

(cost to send a nonce or a principal name), and $\mathsf{NEW\_NONCE\_COST} = 1$ (cost to generate a new nonce). For symmetric-key protocols $\mathsf{SYM\_ENCRYPTION\_COST} = 3$ (cost to encrypt a message with a symmetric key), and for asymmetric-key protocols $\mathsf{ASYM\_ENCRYPTION\_COST} = 3$ (cost to encrypt a message with an asymmetric key).

Table 1 shows the statistics for the protocol generation. The cost threshold is 10 for symmetric-key authentication protocols and 14 for asymmetric-key protocols. The column labeled "Generated" shows how many protocols were initially generated with the corresponding cost threshold without applying the intruder reduction. The table depicts the effectiveness of the impersonator and replay attacks. The column marked "I.A." shows the number of protocols that are eliminated by the impersonation attack. Similarly, the "R.A." column depicts the number of protocols that are vulnerable to the replay attack. The combination of the two attacks is quite efficient (shown in the "Combined" column) and leaves about 2% of candidate protocols for the symmetric case and 0.2% for the asymmetric case (shown in the "Candidate" column). The running time for the protocol generation is on the order of 1 second for every 2000 protocols generated which includes the pruning algorithm (this number is based on our Java implementation, executed by the JVM of the Sun JDK 1.1.7, running on a 400 MHz Pentium II Linux workstation).

### Our Findings of the Protocols

Continuing the experiment from the previous subsection, Athena analyzed the remaining candidate protocols and output 2 correct symmetric-key protocols, which have the minimum cost 10. Among the 110 asymmetric-key protocols, only 1 is correct and has the minimal cost 14. The three protocols are listed below:

- Symmetric-key mutual authentication protocols.

$$\begin{aligned} \mathsf{Protocol:}\quad & A \rightarrow B : N_A, A \\ & B \rightarrow A : \{N_A, N_B, A\}_{K_{AB}} \\ & A \rightarrow B : N_B \end{aligned}$$

$$\begin{aligned} \mathsf{Protocol:}\quad & A \rightarrow B : N_A, A \\ & B \rightarrow A : \{N_A, N_B, B\}_{K_{AB}} \\ & A \rightarrow B : N_B \end{aligned}$$

  The standard symmetric key mutual authentication protocol using random numbers is documented in ISO/IEC 9798 [Int93] as *ISO Symmetric-Key Three-Pass Mutual Authentication Protocol*:

$$\begin{aligned} \mathsf{Protocol:}\quad & A \rightarrow B : N_A, A \\ & B \rightarrow A : \{N_A, N_B, B\}_{K_{AB}} \\ & A \rightarrow B : \{N_A, N_B\}_{K_{AB}} \end{aligned}$$

  Our automatically-generated protocols are clearly simpler than the one listed as ISO standard with respect of serving the same purpose as mutual authentication protocol.

- Asymmetric-key mutual authentication protocols.

$$\begin{aligned} \mathsf{Protocol:}\quad & A \rightarrow B : \{N_A, A\}_{K_B} \\ & B \rightarrow A : \{N_A, N_B, B\}_{K_A} \\ & A \rightarrow B : N_B \end{aligned}$$

  This protocol happens to be the same as the fixed version of Needham-Schroeder protocol [Low96], except for that the last message is not encrypted. This is because we do not have the secrecy requirement in the security property specification.

Although intuitive, another interesting result from the statistics is that the number of correct protocols comparing to the protocols that have the same cost is very low. For example, in this case study, the ratio of generated protocols to correct protocols is around $10^{-4}$. This ratio decreases when we increase the cost threshold.

| Type | Max Cost | Generated | I.A. | R.A. | Combined | Candidates | Correct |
|---|---|---|---|---|---|---|---|
| Symmetric | 10 | 19856 | 12098 | 18770 | 19449 | 407 | 2 |
| Asymmetric | 14 | 46518 | 46378 | 40687 | 46408 | 110 | 1 |

Table 1: Experiment Statistics for protocol generation. I.A. stands for impersonation attack and R.A. for replay attack

**Optimal Protocols**

In this experiment, we experiment with two extreme cases of the metric function to see how we can benefit from the automatic protocol generation to generate optimal protocols.

In the first case, we consider a smart-card, which has a built-in cryptographic accelerator and hence, can perform fast encryption/decryption operations. But the smart-card has a slow link to the card reader. In this case, we set the cost of encryption much lower than the bandwidth cost (UNIT_ELEMENT_COST in the specification). With this metric function, we find one symmetric-key authentication protocol with minimum cost:

$$\begin{aligned} \mathsf{Protocol:}\quad & A \to B : \{N_A, A\}_{K_{AB}} \\ & B \to A : \{N_A, N_B\}_{K_{AB}} \\ & A \to B : N_B \end{aligned}$$

In the second case, we consider a slow machine with a fast link, where the cryptographic operations are the bottleneck. In this case, we set the bandwidth cost much lower than the encryption cost in the metric function. Hence, we get the following two optimal symmetric-key protocols.

$$\begin{aligned} \mathsf{Protocol:}\quad & A \to B : N_A, A \\ & B \to A : \{N_A, N_B, A\}_{K_{AB}} \\ & A \to B : N_B \end{aligned}$$

$$\begin{aligned} \mathsf{Protocol:}\quad & A \to B : N_A, A \\ & B \to A : \{N_A, N_B, B\}_{K_{AB}} \\ & A \to B : N_B \end{aligned}$$

It is interesting to notice that the two protocols in the first case use one more encryption than the two protocols in the second case, while the messages are shorter. We can see a clear benefit from automatic protocol generation, since the protocols generated suit the system requirements ideally.

For the asymmetric-key protocol, in both cases, the automatic protocol generation finds the same protocol as the optimal protocol. The resulting protocol is the same as the asymmetric-key protocol listed in the previous subsection.

# 4 Discussion and Future Work

The approach of automatic protocol generation sounds attractive, but it is initially unclear whether it is feasible to generate meaningful and correct protocols automatically. One goal of this research is to try to answer this question. During the case study, we were able to generate correct authentication protocols automatically and some of them were documented before and are currently in use. The automatic protocol generation process for authentication protocols is efficient, usually only takes matter of seconds of running time. This illustrates that the approach of automatic protocol generation is feasible.

The case study is a proof of concept and shows that automatic protocol generation can accomplish simple tasks, but it says little about whether this approach will scale up to more complicated protocols. Since the protocol space grows exponentially with the number of parties and the number of messages, we expect that the number of candidate protocols, generated by the protocol generator in more complicated cases, can be orders of magnitudes larger than the numbers that appeared in the experiments. It is an interesting research direction to explore more powerful reduction techniques to make this approach scale.

The case study mainly covers the authentication security property. There are many other interesting security properties including properties related to electronic commerce, such as atomicity. We need to extend our system to handle these properties. For example, new reduction techniques are needed for the protocol generator. Athena terminated and successfully analyzed all the candidate protocols generated in the case study for authentication protocols. But for protocols requiring other properties, we might need to add new unreachability theorems to enhance Athena.

Currently, in the protocol analysis, we assume *perfect encryption*. The perfect-encryption assumption states that a ciphertext can only be decrypted if the decryption key is present, and similarly, a ciphertext can only be produced if the encryption key is present. Researchers have been exploring protocols which are resistant against stronger attacks, such as dictionary attacks. It is also interesting to try to strengthen the attacker model in the current approach to produce stronger protocols.

# 5 Conclusion

The main points of the paper are the following:

- We present the novel approach of automatic generation of security protocols. With a user-defined specification of security properties and the system requirements, including a system metric function, APG generates minimal protocols that satisfy the specified security properties and system requirements, minimal with respect to the metric function. This approach is a significant improvement over the current protocol design process, because it is more reliable, efficient, and produces protocols that suit the given system requirements ideally.

- We perform a case study on the automatic generation of two-party mutual authentication protocols for proof of concept and to gain experience with APG. During the case study, APG automatically generates protocols that are simpler than the documented standard ones. In two examples from the real world, APG is also able to generate different optimal protocols with respect to varying metric functions, and hence, demonstrate its benefit.


# 6 Acknowledgments

We would like to thank Doug Tygar for his valuable help and support during this project. We also thank George Necula and Sean Smith for their useful suggestions and discussions. We are also greatful to the anonymous reviewers for their feedback and suggestions.


# References


[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report 39, DEC Systems Research Center, February 1989.

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

[CJM98] E.M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *In Proceedings of the IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

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

[HT96] N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22(1):16–30, January 1996.

[Int93] International Standards Organization. *Information Technology - Security techniques — Entity Authentication Mech-*

*anisms Part 3: Entity authentication using symmetric techniques*, 1993. ISO/IEC 9798.

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

[Mea94] C. Meadows. A model of computation for the NRL protocol analyzer. In *Proceedings of the 1994 Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1994.

[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.

[Mil95] J. Millen. The Interrogator model. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 251–260. IEEE Computer Society Press, 1995.

[MMS97] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using mur$\varphi$. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1997.

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