Automatic Analysis of Recursive Cryptographic Protocols

Diploma Thesis submitted by Klaas Ole Kürtz
supervised by Prof. Dr. Thomas Wilke
and PD Dr. Ralf Küsters

Research Group Theoretical Computer Science
Department of Computer Science
Christian-Albrechts-Universität zu Kiel, Germany

July 2006
Abstract

In recent years, formal methods have been developed to analyze and verify cryptographic protocols. We will focus on protocols that rely on iteration or recursion. These protocols typically use special security tokens – such as numbers used only once, called nonces, or keys generated by a principal – to achieve their security assertions.

The recursion depth of the computations in such protocols and thus the number of fresh tokens occurring in a run of a protocol is not explicitly bounded by the protocol’s description. Therefore, we need a mechanism to provide the protocol’s principals with the ability to generate an unbounded number of fresh tokens.

In this thesis we will extend the model of selecting theories introduced by Truderung – in this model recursive protocols can be analyzed in the presence of a Dolev-Yao intruder. We will present an extended model that allows the principals to generate fresh tokens, and we will show decidability with respect to a bounded number of sessions. In the proof, attacks on such protocols will be represented by a special graph structure introduced by Truderung called ADAG; we will prove our decidability result by bounding the size of ADAGs. In the protocol model and in the ADAGs the modeling of fresh tokens will be based on an infinite set of constants in the signature.
Contents

1 Introduction and Fundamentals 1
   1.1 Cryptographic Protocols .................................................. 2
   1.2 Analysis of Cryptographic Protocols ....................................... 4
   1.3 Horn Clauses and Selecting Theories ....................................... 10
   1.4 Structure of this Thesis .................................................. 10

2 Protocol Model and Main Result 11
   2.1 A Running Example ..................................................... 11
   2.2 Preliminaries .......................................................... 13
   2.3 The Formal Model ..................................................... 16
   2.4 Features of the Model .................................................. 21
   2.5 Modeling the Example Protocol .......................................... 24
   2.6 Main Result .......................................................... 27

3 The Graph of an Attack 28
   3.1 The Intruder Theory ..................................................... 28
   3.2 The Theory of a Protocol ................................................ 29
   3.3 Stage Theory ........................................................ 37
   3.4 The Graph of an Attack .................................................. 43

4 Scaling Down ADAGs 52
   4.1 Preliminaries ........................................................ 52
   4.2 Constructing Simple ADAGs ............................................... 58
   4.3 Bounding the Number of Goals ........................................... 68
   4.4 Bounding the Size of an ADAG .......................................... 75
   4.5 Proof of the Main Result ................................................ 86

5 Conclusion 87

List of Figures ii
Index iii
Bibliography v
1 Introduction and Fundamentals

The use of cryptographic protocols is widely spread – for example, e-commerce and online banking are typically secured by protocols using cryptographic methods as encryption or digital signatures, promising security properties as secrecy or authentication. The design of such protocols is complex and error-prone, hence, an analysis of their security properties is essential. But as the analysis itself is likewise complex and laborious, the overall analysis is usually split up into several more manageable tasks.

One of these tasks is a high-level analysis of the protocol’s logic, abstracting from most of the implementational and cryptographical details of the cryptographic methods used in a protocol. The main question on this level of abstraction is if an adversary can attack a protocol’s security assertions if he has control over the network and can manipulate messages, but has no means like cryptanalysis – this type of adversary is called Dolev Yao intruder. A symbolic approach on this level has allowed models and tools to be develop, which have become useful and practical for the (automatic) analysis of cryptographic protocols [DY83, FHG98, RT01].

While most models can only handle protocols of a simple structure, we will examine a broader class of protocols including ones that can only be modeled by iteration or recursion. These protocols often rely on special security tokens such as numbers used only once, called nonces, or fresh keys generated by principals of the protocol [Mea01, SB05]. As the size of the messages exchanged in such a protocol and thus the recursion depth of the principals’ computations is not explicitly bounded by the protocol’s description, we need a mechanism to provide the principals with the possibility to generate an unbounded number of nonces and fresh keys.

In this thesis we will extend a model introduced by Truderung [Tru05b], which uses special Horn theories – called selecting theories – to model the principal’s computations. In this model secrecy can be decided for recursive protocols in the presence of a Dolev-Yao intruder; we will extend this model to handle the generation of fresh tokens and show decidability of secrecy of a protocol (with respect to a bounded number of sessions). In the decidability proof, attacks on such protocols will be represented by a special graph structure called ADAG introduced in [Tru05a], and we will be able to present a decision algorithm by bounding the size of ADAGs. In the protocol model and in our ADAGs the modeling of fresh tokens will be based on an infinite set of constants along the lines of [KW04].
1.1 Cryptographic Protocols

1.1.1 Cryptographic Primitives

In this thesis we will use cryptographic algorithms such as encryption schemes, digital signatures or hashing algorithms. As we will abstract from the cryptographical and implementational details of these algorithms, we refer the reader to introductory texts like [Sti95] or [DK02], the latter also has a chapter about cryptographic protocols. We will call such algorithms (cryptographic) primitives.

One important primitive we refer to are nonces, which is an abbreviation for number used once. We assume that during computations each computing party can generate numbers that were not generated by this party before. This allows us, for instance, to tag messages, i.e., to distinguish between two messages which have the same content, by appending nonces to the messages.

The Handbook of Applied Cryptography [MVO96] defines nonces as

…] a value used no more than once for the same purpose. It typically serves to prevent (undetectable) replay.

The term nonce is most often used to refer to a »random« number in a challenge-response protocol […]. Three main classes of time-variant parameters are […] random numbers, sequence numbers, and time-stamps. Often, to ensure protocol security, the integrity of such parameters must be guaranteed.

In this thesis, the generation of nonces and the generation of fresh keys will both be referred to as generation of fresh tokens, and we will abstract from the details of the implementation as described in Section 1.2.5

1.1.2 Cryptographic Protocols

In computer science, a communication protocol is a convention for the communication between two or more computing parties called principals. The Handbook of Applied Cryptography [MVO96] defines it as

…] a multi-party algorithm, defined by a sequence of steps precisely specifying the actions required of two or more parties in order to achieve a specified objective.

One has to distinguish between the service offered by a protocol (»a specified objective«), i.e., the purpose and benefit when using this protocol, and the description of the internal structure (»sequence of steps«), i.e., the way in which this protocol tries to accomplish its purpose. The definition of a protocol may include several levels, from the syntax of electronic signals to the semantics of applications.
Cryptographic protocols apply cryptographic methods such as encryption algorithms and digital signatures to offer a specific security service. Their internal structure can often be described on an abstract level, i.e., as a sequence of abstract messages under certain conditions. Their service may, for example, include secrecy, authentication, key exchange or agreement, secured data transport, anonymity, non-repudiation, or fairness. In this thesis we will use the term «security of a protocol» – if not stated otherwise – for secrecy, i.e., that an adversary is not able to derive information that is meant to be kept secret by a protocol.

Today the use of cryptographic protocols on modern communication networks is widely spread – for example, Transport Layer Security (TLS, formerly SSL) and Secure Shell (SSH) are common cryptographic protocol used to secure connections over the internet. Although partly out-of-date, the Clark-Jacob library [CJ97] gives a good survey of cryptographic protocols.

Bellare and Goldwasser state [GB01]:

Classical cryptography is concerned with the problem of [secure] communication between users by providing privacy and authenticity. The need for an underlying infrastructure for key management leads naturally into the topic of key distribution. For many years this is all there was to cryptography.

One of the major contributions of modern cryptography has been the development of advanced protocols. These protocols enable users to electronically solve many real world problems, play games, and accomplish all kinds of intriguing and very general distributed tasks. Amongst these are zero-knowledge proofs, secure distributed computing, and voting protocols.

### 1.1.3 Recursive Cryptographic Protocols

Most of the cryptographic protocols used in real world applications can be abstracted to the mathematical level and then be described by simple means: The protocols consist of a certain sequence of messages exchanged by the principals, where each principal that receives a message replies with a single message that can be computed from the previous messages without the need of complex methods as iteration or recursion.

This is often based on the fact that the cryptographic protocol is separated from the application using it. For example, the standard situation for TLS is «only» to secure communication between two principals by agreeing upon a set of cryptographic algorithms used and then exchanging key material for these primitives. Any application may use this secured communication channel and transmit arbitrary data over it, abstracting from the details of the implementation of TLS.

But some protocols, called recursive protocols, contain complex actions or data structures: On the one hand, there are, for instance, group protocols providing services not only for two principals, but for a potentially unbounded number of principals. Such protocols
often contain data structures such as lists of the principals involved or rely on a server which sends messages to each principal.

On the other hand, there are protocols that integrate application logic and cryptographic methods: For example, there are protocols based on web services and WS-Security that allow the modeling of business processes. During such a business process, a whole set of data is exchanged between several parties, which have to encrypt or digitally sign parts of the messages or data, or which may only be allowed to read or to add data etc.

One important example of a recursive protocol is the Internet Key Exchange (IKE) protocol \[\text{Kau05} \text{, Zh099}\], for more examples see \[\text{SB05}\]. We will use a simple recursive protocol for illustration and as an example protocol, the Recursive Authentication Protocol \[\text{BO97} \text{, Pau97}\].

1.2 Analysis of Cryptographic Protocols

1.2.1 Different Levels and Approaches

When cryptographic protocols are used in the real world, their security depends on factors on different levels. One can distinguish between different types of weaknesses:

1. **User level**: When one uses software containing cryptographic primitives or cryptographic protocols, there are several possible problems like weak or inscribed passwords, unconditional trust exploited by social engineering or phishing, and running security-critical applications on insecure systems.

2. **Implementation level**: Implementing cryptographic algorithms and protocols is extremely error-prone – there may be numerous ways to attack implementations on hardware (e.g., power consumption) or software level (stack-overflow, deadlocks, timing attacks, weak pseudo-random number generators etc.).

3. **Cryptographic Level**: Some cryptographic protocols are designed by simply applying an arbitrary encryption, signature, or hashing scheme. While some of these schemes are simply outdated or weak (for example, DES), the security of other schemes depends on their usage – e.g., the stream cipher RC$_4$ should be used with a pseudo-random key and the first output bytes should be discarded. Naturally, weak or misused cryptographic primitives lead to serious flaws in protocols, the best-known example is the Wired Equivalent Privacy (WEP) protocol which uses RC$_4$ in an insecure way.

4. **Logical level**: Even if one abstracts from all other levels mentioned so far by assuming that the cryptographic primitives used are perfectly secure, there are numerous possible weaknesses in the logic of a protocol itself. As described in the next section, one major example is Lowe’s attack on the Needham-Schroeder protocol \[\text{NS78} \text{, Low95}\], for more examples see, e.g., \[\text{CJ97}\].
Although the security of an actual protocol execution depends on all these levels, the complexity of the analysis necessitates to split the analysis.

The user level (1) typically eludes any analysis. Applications using cryptographic methods are, in practice, analyzed on the implementational level (2), but rarely systematically: These examinations are very laborious, they have to be done manually and do always remain partial. On the cryptographical level (3), many of the cryptographic primitives used in protocols have been thoroughly analyzed or are still analyzed; but naturally, the work on this level will never be finished, either. Finally, as there are various potential problems on the logical level (4) of protocol design, it is also necessary to analyze the logic level of protocols.

There are two main approaches on the logical level: The first one links the security of a protocol on the logical level to the security of the cryptographic primitives, i.e., linking level three and four, while the latter one completely abstracts from the details of cryptographic primitives, i.e., solely analyzing the logical level.

- In the computational model, messages are viewed as bit strings, and the cryptographic operations applied on messages are considered as functions on bit strings. An adversary is typically modeled by a probabilistic polynomial-time Turing machine; and security is proven by reducing the security of a protocol to security of the cryptographic primitives. This usually yields security results which express the probability or computational complexity of attacks like in [BR93]. In this model, the complex proofs can often only be done manually, which is error-prone [CBH05].

- On the other hand, the symbolic approach abstracts from many of these details. Messages are modeled by a term algebra, and so the cryptographic primitives are only modeled by a set of axioms, assuming perfect cryptography. The intruder is able to modify terms in an idealized way. Thus, security properties can be described in formal logics. This enables a (semi-)automatic analysis by model checking, theorem proving or similar methods – if the problem is decidable at all, what is not necessarily the case. We will give references in the next two sections.

- In addition to these two separate approaches, first results were achieved to link both the symbolic and the computational approach, see, e.g., [AR00, BPW03, CW05].

In this thesis we will analyze protocols on the logical level, pursuing the symbolic approach, i.e., we will abstract from cryptographical details as described in the next sections.

1.2.2 History on the use of Formal Methods

From the beginning of computer networks, protocols were developed, implemented, and tested by hand, i.e., by trial-and-error: A protocol was published (or some software using a protocol), and if a bug or a flaw was found in a protocol or software, a revised version was
developed. But similar to the verification of algorithms, formal methods were developed and began to be practical and useful to analyze the security of cryptographic protocols.

In 1978, Needham and Schroeder published an article entitled »Using Encryption for Authentication in Large Networks of Computers« [{NS}78]. They described the scenario of a network without a central authority that can ensure authentication etc., and they proposed a threat model:

We assume that an intruder can interpose a computer in all communication paths, and thus can alter or copy parts of messages, replay messages, or emit false material. While this may seem an extreme view, it is the only safe one when designing authentication protocols.

This description is the basis for today's formal analysis on the logical level of protocols. In 1983, Dolev and Yao presented the first method which allowed protocols to be modeled on this level, making it possible to formally analyze their secrecy properties with respect to the Needham and Schroeder threat model [{DY}83].

Ironically, an authentication protocol proposed by Needham and Schroeder [{NS}78] is today's best-known example for a protocol that was analyzed with formal methods – and failed: Although Denning and Sacco manually analyzed the protocols proposed by Needham and Schroeder in 1981 and found flaws in it [{DS}81], an important attack on the asymmetric Needham-Schroeder protocol was found by Lowe no earlier than 1995 [{Low}95].

Lowe then applied formal methods to analyze the flaw and to show that his extension of the protocol is secure [{Low}96]. Today, a modified version of the protocols proposed by Needham and Schroeder is widely used in the authentication protocol Kerberos [{SNS}88].

For a survey of the history of the symbolic analysis of cryptographic protocols, for numerous examples and for an outlook on open problems see [{Mea}95, {Mea}01, {Mea}03] and [{CS}02].

1.2.3 The Dolev-Yao Model

The symbolic approach of protocol analysis is based on the so-called Dolev-Yao model, which is a generic term for a set of assumptions and abstractions that has developed out of [{DY}83]. These assumptions mostly include the following ones, which will be used in our model, too:

1. A protocol run consists of messages exchanged by a set of principals. Each principal is defined by a set of consecutive actions including receiving and sending messages; such an action is called a protocol step.

2. The set of messages is created by a term algebra over a signature that contains constants for atomic keys, the names of principals, nonces etc., as well as function symbols for pairing and cryptographic primitives such as encryption or digital signatures.
3. The principals are connected via a network that is insecure, i.e., there is an intruder which has complete control over the network: He can read, manipulate, or delete each message sent by the principals, and he can create messages by combining (parts of) messages he read. This is often illustrated and modeled as »The intruder is the network«.

4. There can be several sessions of each principal, i.e., there may be multiple parallel runs of the protocol. There typically is a basic knowledge shared between the sessions of one principal, e.g., they all know the principal’s name, but further than that, the different sessions are typically not connected nor coordinated except by the network, i.e., by the intruder.

5. The cryptographic primitives work perfectly – for example, the only way to learn the plaintext from an encrypted message is to use the appropriate key. No form of cryptanalysis is possible.

Although these assumptions are not always realistic, they are necessary to analyze protocols on this level of abstraction. Many decision problems and formal methods were developed out of the Dolev-Yao model or inspired by it.

Previous Results and Restrictions

The general decision problem of secrecy of a protocol and many variants thereof are undecidable as shown, e.g., in [DLMS99, EG83, AC02]. But there are important variants of the problem that are decidable – also see Figure 1.1 for selected results: For a bounded total number of sessions, the secrecy problem was shown to be \( \text{NP}-\text{complete} \) [RT01, RT03], for a bounded message size and a bounded number of nonces it is \( \text{DEXP}-\text{complete} \) [DLMS04].

The decidability results for a bounded number of sessions have been extended to protocols that incorporate, e.g., properties of operators like XOR [CKRT03a, CLS03] or Diffie Hellman exponentiation [CKRT03b]. In addition, while most work on the formal analysis of cryptographic protocols has concentrated on protocols that offer secrecy, authentication or key exchange, there are other security properties that can be analyzed and decided [KKW05].

For decidable classes of protocols, several techniques were developed and have been applied for the (semi-)automatic analysis of secrecy of such protocols, including model checkers [MMS97], theorem provers [BS97], logic programming [CDL+99], and tree automata [Küs03].

In [Mea03], Meadows identified several emerging areas of research: open-ended protocols, new applications and threats (like anonymous communication or electronic commerce), high fidelity (i.e., lowering the level of abstraction), composability of protocols, and »getting it into the real world«. Our focus on recursive protocols falls in the first category.
1 Introduction and Fundamentals

1.2.4 Analyzing Recursive Cryptographic Protocols

So far, most research has concentrated on non-recursive protocols, i.e., protocols where the protocol steps can be expressed by simple rewrite rules. However, as there exists many recursive protocols, the relevance of their analysis is apparent – this has been pointed out by, e.g., [Mea01, FS00, Zho99]. Recursive protocols have been analyzed manually [PQ01] and (semi-)automatically [BS97, Mea00, SB05] – e.g., our running example, the Recursive Authentication Protocol, has been analyzed using the theorem provers CSP/PVS [BS97] and Isabelle/HOL [Pau97]. Furthermore, a general decision algorithm for an automatic analysis of recursive protocols was presented in [KW04, Tru05b].

Modeling of Fresh Tokens

In the case of a finite number of sessions and non-recursive protocols, the modeling of fresh tokens is quite easy – all nonces and keys used in a protocol run can simply be atomic messages, i.e., constants of the signature used to model terms [RT01]. This is due to the fact that in each session only a bounded number of nonces can be generated, so the total number of nonces is bounded in terms of the size of the protocol.

In models that do not have these restrictions, one has to introduce other means to model fresh tokens correctly. Some models allow an unbounded number of sessions – for example, the approach of multiset rewriting [CDL+99, DLMS04] uses existential quantification to model nonces, and the strand space model [FHG98] applies special restrictions on strands and bundles to guarantee freshness.

In [KW04] the number of sessions is bounded, but protocols can be recursive – therefore, an infinite signature is used to allow an unbounded set of fresh tokens, and the tree automata used to develop the decision algorithm are designed to cope with infinite signatures. We will adopt this method and use an infinite signature of anonymous constants (the name also originates from [KW04]), and we will modify the structures used in the proofs in [Tru05b] to cope with infinite signatures or unbounded subsets thereof.

Comparison of Different Approaches

Figure 1.1 compares different approaches and some results. When aiming at recursive protocols, one naturally has to consider unbounded message sizes. Hence, to maintain

<table>
<thead>
<tr>
<th>unbounded message size</th>
<th>unbounded number of session</th>
<th>recursive protocols</th>
<th>fresh tokens</th>
<th>decidability and references</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>yes</td>
<td>–</td>
<td>yes</td>
<td>undecidable [DLMS04]</td>
</tr>
<tr>
<td>–</td>
<td>yes</td>
<td>–</td>
<td>–</td>
<td>decidable [DLMS04]</td>
</tr>
<tr>
<td>yes</td>
<td>–</td>
<td>yes</td>
<td>yes</td>
<td>decidable [RT01]</td>
</tr>
<tr>
<td>yes</td>
<td>–</td>
<td>yes</td>
<td>–</td>
<td>decidable [KW04]</td>
</tr>
<tr>
<td>yes</td>
<td>–</td>
<td>yes</td>
<td>yes</td>
<td>decidable [Tru05b]</td>
</tr>
<tr>
<td>yes</td>
<td>–</td>
<td>yes</td>
<td>yes</td>
<td>decidable (this thesis)</td>
</tr>
</tbody>
</table>
decidability it is necessary to limit the total number of sessions and thus the number of principals. In [Küs03, KW04] transducers were used to model recursion. As stated in [KT06],

[...] the expressivity of these transducer-based models is orthogonal to the Horn theory model: While the transducer-based models allow [principals] to output messages of complex structure, in the Horn theory model only lists (or sets) of messages of a more simple structure can be produced. The main disadvantage of the transducer-based model is that, unlike the Horn theory model, messages cannot be tested for equality without losing decidability. This, as already observed in [KW04], immediately implies that security is undecidable in the transducer-based model with \texttt{XOR} (or Diffie Hellman exponentiation) since these operators allow for (implicit) equality tests between arbitrary messages.

On the other hand, properties of the \texttt{XOR} operator can be added to the model based on Horn theories, allowing decidability not for all protocols, but for a major class of protocols [KT06].

In this thesis we will examine recursive protocols with an unbounded message size and an unbounded number of fresh tokens, but we will limit the number of sessions – in fact, we will limit the number of principals and allow only one session per principal, but a bounded number of sessions per principal can simply be simulated by copying each principal.

1.2.5 Generation of Fresh Tokens as a Cryptographic Primitive

As described above, we abstract from the mathematical and implementational details of the cryptographic primitives used and make certain assumptions, e.g., for encryption schemes. As we view the generation of fresh tokens as a cryptographic primitive, we have to formulate assumptions for this primitive as well – similar assumptions are presented, e.g., in [CS02].

During this thesis we will abstract from the details of the generation of fresh tokens and use the following assumptions:

- If a principal $A$ generates a fresh token, no other principal is able to predict or derive that token other than by receiving this information from $A$, directly or indirectly.
- Two different principals will never generate the same token.
- There is an unbounded set of values for fresh tokens.

Of course, these assumptions are not realistic: As [MVO96] states, nonces are typically implemented by random numbers, sequence numbers, or timestamps; so there is, for example, usually no guaranteed uniqueness. Nevertheless, the possibilities involved here, e.g., for the incidental generation of equal random numbers on two different personal computers, are usually negligibly small.
1.3 Horn Clauses and Selecting Theories

In [Tru05b] and in our model sets of Horn clauses are used to describe the principals of a protocol. Horn clauses are named after Alfred Horn who analyzed special classes of clauses in [Hor51]. They are the basis for many deduction methods in artificial intelligence [Rob65] and furthermore, logic programming languages (e.g., PROLOG) are mostly based on Horn clauses [Llo84].

Kowalski gives an introduction to Horn clause logic in [Kow79] and states:

The majority of formalisms for computer programming bear greater resemblance to Horn clauses than they do to »non-Horn« clauses. In addition, most of the models of problem-solving which have been developed in artificial intelligence can be regarded as models for problems expressed by means of Horn clauses.

Thus, Horn theories – i.e., sets of Horn clauses – have proven to be a good choice for modeling the knowledge and the deduction capabilities of the principals and the intruder, there are various approaches to the verification of cryptographic protocols using Horn clauses [Blao97, VSS05, Tru05b].

Truderung introduces a special class of Horn theories called selecting theories [Tru05b], which are used to model the handling of messages by the protocol’s principals and the intruder. He uses three kinds of predicates in the Horn clauses: One is modeling the intruder’s knowledge, a second kind is used to describe the recursive processing of terms by a principal, and a third kind is modeling tree automata and provides a regular look-ahead when processing messages. Upon receiving a message a principal applies a rewrite rule. Then he uses a selecting theory to process the result, deriving new messages that are added to the intruder’s knowledge. The intruder is then able to process these messages by a special Horn theory, composing and decomposing terms to construct new messages.

As we will extend Truderung’s model in this thesis, we refer the reader either to Chapter 2 for the details of the extended model or to [Tru05b] for the basic model.

1.4 Structure of this Thesis

In Chapter 2 we will first provide an example protocol. Then we will present our model, and we will define a security notion for this kind of protocols. We will also show how to model the example protocol, and we will state our main result. In the following two chapters we will prove the decidability result: First, in Chapter 3 we will introduce a graph structure called ADAG which represents an attack on a protocol, i.e., a protocol is insecure if and only if an ADAG for this protocol exists. In Chapter 4 we show that we can always find ADAGS of a bounded size in terms of the size of the protocol, which leads to the possibility to decide security by checking all these bounded ADAGS in bounded time. In the last chapter we will conclude and show possible extensions of our model.
2 Protocol Model and Main Result

In this chapter we will extend the model of »Selecting Theories« developed in [Tru05b] and briefly introduced in Section 1.3 to be able to generate an unbounded number of fresh tokens and thus model nonces and generated keys more accurately. We will start with a running example, the Recursive Authentication Protocol [BO97, Pau97]. Then we will present preliminaries necessary for our model, the latter is then presented in Section 2.3. Afterwards we will show how to model the running example in Section 2.5 and present our main result in Section 2.6.

2.1 A Running Example

To illustrate the usage of our model, we will use the »Recursive Authentication Protocol« which was presented by Bull and Otway [BO97]. After the description of the protocol in this section we will give a formal model of the protocol in our extended model in Section 2.5.

The Recursive Authentication Protocol is an extended version of the authentication protocol by Otway and Rees [OR87]. The latter can only cope with two principals, while the recursive protocol allows an unbounded number of principals in one protocol run. Note that we will use a modified version provided by Paulson [Pau97] – amongst minor differences the original version uses exclusive or (XOR) instead of encryption, allowing a simple attack on the protocol [RS98].

For this protocol we assume that there is a set of principals \( \{P_1, \ldots, P_N\} \) who would like to communicate pairwise, i.e., \( P_n \) wants to communicate in a secure way with \( P_{n+1} \) for all \( n \in \{1, \ldots, N-1\} \).

In addition, there is a server \( P_S \) (with \( S = N + 1 \)) that, for each \( n \in \{1, \ldots, N\} \), shares a long-term key \( K_n \) for a symmetric encryption scheme with the principal \( P_n \). The server shall now generate new symmetric session keys \( K_{(n,n+1)} \) for the communication between \( P_n \) and \( P_{n+1} \) for all \( n \in \{1, \ldots, N\} \).

To start the protocol \( P_1 \) sends the following message to \( P_2 \), containing the names \( P_1 \) and \( P_2 \), a nonce \( \tilde{N}_1 \) generated by \( P_1 \) and a constant marking \( \perp \) the start of the protocol – using the abbreviation \( \text{hash}_K(x) \) for \( \langle \text{hash}(\langle K, x \rangle), x \rangle \):

\[
P_1 \to P_2: m_1 = \text{hash}_{K_1}(P_1, P_2, \tilde{N}_1, \perp).
\]
Principal $P_2$ sends the following message to $P_3$, again containing two names, a nonce $N_2$ generated by $P_2$ and the first request received from $P_1$:

$$P_2 \rightarrow P_3: m_2 = \text{hash}_{K_2}(P_2, P_3, \hat{N}_2, m_1).$$

Accordingly, $P_N$ receives the message $m_{N-1}$ and sends this message to the server $P_S$:

$$P_N \rightarrow P_S: m_N = \text{hash}_{K_N}(P_N, P_S, \hat{N}_N, m_{N-1}).$$

First the server processes $m_N$ without knowing the size of $m_N$ in advance, which depends on the number of the protocol’s principals. The server generates keys $K_{(n,n+1)}$ for $n \in \{1, \ldots, N\}$ and composes the following messages:

$$m'_N = \langle \{K_{(N-1,N)}, P_{N-1}, P_N, \hat{N}_N \}_{K_{(N,N)}}, \{K_{(N,S)}, P_N, P_S, \hat{N}_N \}_{K_N} \rangle,$$

$$\vdots$$

$$m'_n = \langle \{K_{(n-1,n)}, P_{n-1}, P_n, \hat{N}_n \}_{K_n}, \{K_{(n,n+1)}, P_n, P_{n+1}, \hat{N}_n \}_{K_n} \rangle,$$

$$\vdots$$

$$m'_2 = \langle \{K_{(1,2)}, P_1, P_2, \hat{N}_2 \}_{K_2}, \{K_{(2,3)}, P_2, P_3, \hat{N}_2 \}_{K_2} \rangle,$$

$$m'_1 = \{K_{(1,2)}, P_1, P_2, \hat{N}_1 \}_{K_1}.$$

Second the server sends all these messages to $P_N$, who will decrypt the message $m'_N$ and forward all other messages to his predecessor $P_{N-1}$. They will act in the same way, resulting in the following messages to be sent:

$$P_S \rightarrow P_N: \langle m'_1, \ldots, m'_N \rangle,$$

$$P_N \rightarrow P_{N-1}: \langle m'_1, \ldots, m'_{N-1} \rangle,$$

$$\vdots$$

$$P_3 \rightarrow P_2: \langle m'_1, m'_2 \rangle,$$

$$P_2 \rightarrow P_1: m'_1.$$

At the end of the protocol run, principals $P_n$ and $P_{n+1}$ share the key $K_{(n,n+1)}$ for all $n \in \{1, \ldots, N\}$.

### 2.1.1 Fresh Tokens

The Recursive Authentication Protocol is an example of a recursive protocol: The server gets a linked list of single requests, which is processed recursively. The size of the total message received by the server and thus the number of elements in the list is neither known in advance nor explicitly bounded by the protocol’s description, i.e., the server
should be able to process messages of arbitrary length (aside from real-world technical bounds as memory or network capacity).

During this process, the server has to generate keys for each request, hence, the server may have to generate an unbounded number of keys. In the previous section, we modeled this by defining a key for each pair of principals. But this may lead to re-use of keys: If a groups of principals (or the intruder) generate a message containing nested requests for symmetric keys for two principals, the server creates duplicate certificates, i.e., he uses the same key twice for both requests.

Consider the following request received by the server:

\[ m = \text{hash}_{K_B}(B, S, N_B', \text{hash}_{K_A}(A, B, N_A', \text{hash}_{K_B}(B, A, N_B, \bot))) \]  

The server will send the keys \( K_B(B, S) \) and \( K_B(B, A) \), but also keys for the communication between A and B initiated by A, i.e., he will send a key \( K_A(A, B) \). When using a model which has a bounded number of constants, we will usually only have one key \( K_A(A, B) \). The server has to send the following terms, sending the key \( K_A(A, B) \) twice to both A and B:

\[
\langle \{ K_A(A, B), A, B, N_A' \} K_A, \{ K_B(B, S), B, S, N_B' \} K_B \rangle,
\langle \{ K_A(A, B), B, A, N_A' \} K_A, \{ K_B(B, A), A, B, N_B \} K_B \rangle,
\langle \{ K_A(A, B), A, B, N_B \} K_B, \{ K_B(B, A), B, A, N_A \} K_A \rangle
\]

But this is not what a real server would do. Upon receiving this kind of request, a real server would generate a new key for each of the connections, ignoring the fact that some of these connections are intended for the same pair of principals.

Thus, our goal is to allow the server to generate an unbounded number of fresh tokens so that a verification of such a protocol does not have to rely on a finite set of constants defined in the signature of that protocol.

### 2.2 Preliminaries

In this section we will introduce basic notations and preliminaries necessary for our model.

#### 2.2.1 Signatures and Terms

The messages used in our model will be terms over a formal term algebra, and we will extend the normal (finite) signature with an infinite set of constants, which will contain constants called anonymous constants. In addition, we will use three different kinds of variables: regular variables, anonymous variables, and fresh variables. The main idea is that
the anonymous and fresh variables are typed variables, i.e., the only values that can be assigned to them are anonymous constants. Furthermore, we assume that the substitutions we use cannot assign the same anonymous constant to two different fresh variables. Thus, we can guarantee freshness of anonymous constants.

**Definition 2.1.** Let $\Sigma$ be a finite signature, i.e., a set of function symbols with an arity. By $\Sigma_i$ we denote the subset of all function symbols of arity $i$ in $\Sigma$. Symbols with arity zero are called constants. Let $\Gamma = \{c_1, c_2, \ldots\}$ with $\Gamma \cap \Sigma = \emptyset$ denote a countably infinite set of constants which we call anonymous constants.

Let $X, Y, Y^*$ be disjoint finite sets of variables: The elements $x \in X$ will be called regular variables, elements $y \in Y$ will be called anonymous variables, and variables $y^* \in Y^*$ are called fresh variables. We use $V = X \cup Y \cup Y^*$.

The set of terms over a signature $\Sigma$ and a set of variables $V$ will be denoted by $T(\Sigma, V)$; we will need the set $T_V = T(\Sigma \cup \Gamma, X \cup Y \cup Y^*)$. Terms are called ground if they do not contain any variables, neither regular nor anonymous nor fresh ones. A term is called linear if each variable occurs at most once in it. Flat terms are of the form $f(t_1, \ldots, t_m)$ with $f \in \Sigma_m$ and $t_1, \ldots, t_m$ are constants or variables. We will call terms simple if they are linear or flat, possibly both.

For a term $t \in T_V$, let $\text{sub}(t)$ denote the set of subterms and $\text{depth}(t)$ the depth of that term, i.e., for all $c \in \Sigma_0 \cup \Gamma \cup V$ and for all $f \in \Sigma_n$ (for some $n \in \mathbb{N}_{>0}$ and terms $t_1, \ldots, t_m \in T_V$) we recursively define:

\[
\text{sub}(c) = \{c\}, \quad \text{sub}(f(t_1, \ldots, t_m)) = \{f(t_1, \ldots, t_m)\} \cup \bigcup_{j=1}^{m} \text{sub}(t_j),
\]

\[
\text{depth}(c) = 0, \quad \text{depth}(f(t_1, \ldots, t_m)) = 1 + \max \{ \text{depth}(t_j) \mid j \in \{1, \ldots, m\}\}. \tag{2.3}
\]

Let $\text{var}(t)$ denote the set of all variables occurring in $t$, i.e., $\text{var}(t) = \text{sub}(t) \cap V$. For each $V' \subseteq V$ we write $\text{var}_{V'}(t)$ for $\text{var}(t) \cap V'$, e.g., the fresh variables of a term $t$ are denoted by $\text{var}_{Y^*}(t)$.

### 2.2.2 Anonymous Substitutions

In order to cope with anonymous and fresh variables, anonymous substitutions are defined, which map variables to terms, anonymous and fresh variables to anonymous constants, and different fresh variables to different anonymous constants.

**Definition 2.2.** A (ground) substitution is a mapping $\sigma : X \rightarrow T(\Sigma \cup \Gamma, X)$ from regular variables to (ground) terms. A substitution $\sigma$ is extended to a substitution from terms to
terms \( \hat{\sigma} : T(\Sigma \cup \Gamma, X) \rightarrow T(\Sigma \cup \Gamma, X) \) in the natural way:

\[
\hat{\sigma}(t) = \sigma(t) \quad \text{for } t \in X, \tag{2.5}
\]

\[
\hat{\sigma}(t) = f(\hat{\sigma}(t_1), \ldots, \hat{\sigma}(t_m)) \quad \text{for } t = f(t_1, \ldots, t_m) \text{ with } f \in \Sigma_m. \tag{2.6}
\]

A function \( \sigma : X \cup Y \cup Y^* \rightarrow T_Y \) is called \emph{anonymous substitution} if the following two conditions hold:

\[
\sigma(y) \in \Gamma \quad \text{for all } y \in Y \cup Y^* \quad \text{and} \tag{2.7}
\]

\[
\sigma(y^*_1) \neq \sigma(y^*_2) \quad \text{for all } y^*_1, y^*_2 \in Y^* \text{ with } y^*_1 \neq y^*_2. \tag{2.8}
\]

An anonymous ground substitution is defined analogously, and both can be extended to a mapping on terms as above.

In the rest of this thesis, we will simply use \( \sigma \) and the denotation \( \text{anonymous substitution} \) for both the substitution \( \sigma \) and the extended anonymous substitution \( \hat{\sigma} \).

\textbf{Example 2.3.} Let \( t = f(y^*_1, y^*_2) \) for a function symbol \( f \in \Sigma_2 \) and two distinct fresh variables \( y^*_1 \neq y^*_2 \), an anonymous substitution \( \sigma \) cannot map \( t \) to \( f(c, c) \) with \( c \in \Gamma \), instead we have \( \sigma(t) = f(c_1, c_2) \) for some \( c_1, c_2 \in \Gamma \) and \( c_1 \neq c_2 \).

\textbf{2.2.3 Registers}

When processing a message, our principals will have access to a special memory containing a bounded number of anonymous constants, called \emph{register}. The principals can read constants from that memory, move constants within the memory and generate anonymous constants at each single position in the register.

\textbf{Definition 2.4.} A tuple \( \kappa = (\kappa_1, \ldots, \kappa_Z) \in (\Gamma \cup Y \cup Y^*)^Z \) for \( Z \in \mathbb{N}_{>0} \) will be called a \emph{register} where \( Z \) is the \emph{register size}. We will denote \( \kappa_i \) by \( \kappa[i] \). We define \( \text{var}_Y(\kappa) = \{\kappa_1, \ldots, \kappa_Z\} \cap Y \) and \( \text{var}_{Y^*}(\kappa) \) analogously, let \( \text{var}(\kappa) = \text{var}_Y \cup \text{var}_{Y^*} \).

If we have \( \kappa \in (Y \cup Y^*)^Z \), we call it a \emph{register of variables}, if otherwise we have \( \kappa \in \Gamma^Z \), it is called a \emph{ground register}. We will use a special term, called \emph{simple register}, which is defined as \( \kappa_s = (y_1, \ldots, y_Z) \in Y^Z \) with pairwise distinct anonymous variables \( y_1, \ldots, y_Z \). We will later use \( \kappa[i] \) to denote the \( i \)th element in a register.

Let \( \Gamma = \{c_1, c_2, \ldots\} \), then we define \( \kappa^{(n)} = (r_{n.Z+1}, \ldots, r_{n.Z+Z}) \) for each \( n \in \mathbb{N} \).

\textbf{2.2.4 Horn Theories and Proofs}

As stated in Section 1.3, we will use \emph{Horn theories} to model both the protocol’s principals and the intruder, allowing recursive computations for the handling of messages.
Definition 2.5. Let $P$ be a set of unary predicate symbols and $R$ be a set of binary predicate symbols. For $p \in P$ and $t \in T_V$ we say $p(t)$ is an atomic (ground) formula if $t$ is a (ground) term. For $r \in R$ and a (ground) register $\kappa$ let $r(t, \kappa)$ also be an atomic (ground) formula. We will use the term (Horn) fact for atomic ground formulas.

A Horn theory $\Phi$ is a finite set of Horn clauses of the form $[a_1, \ldots, a_m \rightarrow a_0]$ where $a_0, \ldots, a_m$ are atomic formulas. In addition we assume that fresh variables do only appear in $a_0$. ◁

Definition 2.6. Let $\Phi$ be a Horn theory, and let $A$ and $B$ be sets of facts. Let $\Pi = [b_1, \ldots, b_m]$ be a finite sequence of facts with $B \subseteq \{b_1, \ldots, b_m\}$. We further assume that there is a set $\{C_1, \ldots, C_m\}$ of pairwise disjoint subsets of $\Gamma$, let $C = \bigcup_{j=1}^m C_j$. We assume that the following holds for each $j \in \{1, \ldots, m\}$:

1. Either $b_j$ is an assumed fact (a-fact), i.e., $b_j \in A$ and $C_j = \emptyset$; or
2. $b_j$ is a constructed fact (c-fact), i.e., there is a clause $\varphi = [t_1, \ldots, t_n \rightarrow t_0] \in \Phi$ and an anonymous substitution $\sigma_j$ such that there exists $\{t'_1, \ldots, t'_n\} \subseteq \{b_1, \ldots, b_{j-1}\}$ with $\sigma(t_0) = b_j$ and $\sigma(t_i) = t'_i$ for all $i \in \{1, \ldots, n\}$; with $C_j = \sigma(\text{var} \gamma^*(t_0))$. We call $b_j$ constructed by $\varphi$ or obtained by $\varphi$.

Such a sequence $\Pi$ is called (Horn) proof of $B$ with respect to $\Phi$ assuming $A$ assigning $C$. ◁

If there is a Horn proof of $B$ with respect to $\Phi$ assuming $A$ which assigns $C$, we write $A \vdash^C\Phi B$, calling $A$ the assumption and $B$ the conclusion of the proof. For a single atomic formula $b$ we write $A \vdash^C\Phi b$ instead of $A \vdash^C\Phi \{b\}$. If $A \vdash^C\Phi B$ holds for some $C \subseteq \Gamma$, we will also omit $C$ and write $A \vdash^C\Phi B$.

Note that we use squared brackets to denote sequences, e.g., $\Pi = [b_1, \ldots, b_m]$, and that we will denote the $i$th element of a sequence $S$ by $S[i]$, e.g., we have $\Pi[i] = b_i$.

Remark 2.7. In this thesis we will use algorithms to construct Horn proofs. This may lead to duplicate facts in the same proof. While this is no problem for a-facts and c-facts constructed by clauses without fresh variables, this may lead to two identical facts both being constructed by the same clause containing a fresh variable: This would contradict the assumption that the sets $C_i$ are pairwise disjoint. Therefore, for the rest of the thesis we assume that each fact is added to a proof only once. ◁

We have now defined all the preliminaries we need to describe our protocol model.

2.3 The Formal Model

2.3.1 Messages

We will start with the term algebra defining the set of messages that can be used in our protocols.
2 Protocol Model and Main Result

Definition 2.8. Let $\Sigma$ be a finite signature containing the following symbols:

- a constant $\$\$ modeling the intruder’s initial knowledge,
- a constant $\cent$ modeling the secret,
- a binary function symbol $\langle, \rangle$ for pairing,
- a binary function symbol $\{, \}$ for symmetric encryption,
- a binary function symbol $\{[, ]\}$ for asymmetric encryption,
- an unary function symbol $\text{hash}(\cdot)$ for hashing, and
- additional constants depending on the protocol, e.g., for keys.

Let $K \subseteq \Sigma_0$ be the set of keys used in this protocol. These keys are constants, both for symmetric as for asymmetric encryption. We assume that there exists a bijection $\cdot^{-1}$ on the set of keys which maps every public (private) key $k$ to its corresponding private (public) key $k^{-1}$.

Let $\Gamma$ be an infinite set of constants. Ground terms over $\Sigma \cup \Gamma$ are called messages.

For the rest of this thesis, we will consider $\Sigma$, $K$, and $\Gamma$ to be fixed sets. We will omit the pairing symbols if possible – for instance, we will write $I(x, y)$ instead of $I(\langle x, y \rangle)$, or $\langle x_1, \ldots, x_4 \rangle$ instead of $\langle x_1, \langle x_2, \langle x_3, x_4 \rangle \rangle \rangle$. As used above, we will denote $\langle \text{hash}(\langle K, x \rangle), x \rangle$ by $\text{hash}_K(x)$.

2.3.2 Selecting Theories

Now the main ingredient of our model is defined – special Horn theories called selecting theories. We will use them to process messages: When a message $t$ is sent over the network, there will be a predicate symbol $r$ such that a principal’s processing of that message is handled by a selecting theory applied on the fact $r(t)$.

We will take three different kinds of predicate symbols and accordingly we define three types of clauses, whose function can informally be described as follows:

1. Using push clauses we can process messages recursively, moving from a term in the message to one of its subterms (or some of its subterms in parallel). There will be a register where we can store anonymous constants as explained above, and we will be able to generate anonymous constants during each push step.

2. Send clauses allow principals to send messages to the network – and thus directly to the intruder, as proposed by the Dolev-Yao model.

3. Pop clauses will provide a regular lookahead when processing messages as they can simulate runs of a nondeterministic bottom-up tree automata when viewing the message terms as trees: They assign pop symbols from a set of predicate symbols $Q$ to subterms of messages, allowing the use of these symbols in other clauses. Note that we will be able to assign pop symbols to anonymous constants.
For the resemblance between pop clauses and tree automata we refer the reader to [Tru05a].

**Definition 2.9.** Let \( Q \) and \( R \) be disjoint sets of predicate symbols. The elements of \( Q \) are assumed to be unary predicate symbols and are called *pop symbols*, the elements of \( R \) are binary predicate symbols called *push symbols*. Let \( I \not\subseteq Q \cup R \) be an unary predicate symbol, which will model the intruder’s knowledge.

Let \( \kappa, \kappa' \) be registers of variables with \( \text{var}(\kappa) \subseteq Y \) and \( \text{var}(\kappa') \subseteq \text{var}(\kappa) \cup Y^* \). Let \( m \in \mathbb{N}, j \in \{1, \ldots, m\} \), pop symbols \( q, q_1, \ldots, q_m \in Q \), push symbols \( r, r' \in R \), function symbols \( f \in \Sigma_m \cup Y \) and \( g \in \Sigma_m \), variables \( x_1, \ldots, x_m \in X \). Let \( t_f = g(x_1, \ldots, x_m) \), and let \( t_I \in T(\Sigma, X) \) be a simple term, let \( s_I \in T(\Sigma, X \cup Y) \) with \( \text{var}(s_I) \subseteq \text{var}(t_I) \cup \text{var}(\kappa) \).

We define the following three types of Horn clauses:

\[
\begin{align*}
\text{pop clauses} & \quad q_1(x_1), \ldots, q_m(x_m) \rightarrow q(f(x_1, \ldots, x_m)), \quad (2.9a) \\
\text{push clauses} & \quad q_1(t_f), \ldots, q_m(t_f), r(t_f, \kappa) \rightarrow r'(x_f, \kappa'), \quad (2.9b) \\
\text{send clauses} & \quad q_1(t_I), \ldots, q_m(t_I), r(t_I, \kappa) \rightarrow I(s_I). \quad (2.9c)
\end{align*}
\]

Pop, push, and send clauses are special Horn clauses. Hence, a finite set \( \Phi \) of such clauses is a special Horn theory called *selecting theory using anonymous constants over \( (Q, R) \) with register size \( Z \).*

We remark that by using an anonymous variable \( f \in Y \) (and thus, \( m = 0 \)) we are able to assign pop symbols to anonymous constants.

**Example 2.10.** Consider the following selecting theory \( \Phi \) containing three clauses over \( Q = \{q\} \) and \( R = \{r\} \) with register size \( Z = 1 \), regular variables \( x, x' \), an anonymous variable \( y \) and a fresh variable \( y^* \):

\[
\begin{align*}
q(x) & \rightarrow q(y) \quad \text{assign a pop symbol to an anonymous constant}, \quad (2.10) \\
r(x, x') & \rightarrow r(x', y^*) \quad \text{descend in the right component}, \quad (2.11) \\
q(x) & \rightarrow I(\langle x, y \rangle) \quad \text{send a fresh anonymous constant}. \quad (2.12)
\end{align*}
\]

Now consider a term \( t = \langle \langle c_1, c_2 \rangle, \langle d, c_3 \rangle \rangle \) with anonymous constants \( c_1, c_2, c_3 \in \Gamma \) and \( d \in \Sigma_0 \). Then we have \( r(t, c_4) \models_{\Phi}^{\langle c_5, c_6 \rangle} I(\langle c_3, c_6 \rangle) \) as the following Horn proof shows:

\[
\begin{align*}
r(\langle c_1, c_2 \rangle, \langle d, c_3 \rangle), & \quad \text{a-fact} \quad (2.13) \\
r(\langle d, c_3 \rangle, c_5) & \quad \text{c-fact by applying (2.11) on (2.13)} \quad (2.14) \\
r(c_3, c_6) & \quad \text{c-fact by applying (2.11) on (2.14)} \quad (2.15) \\
q(c_3) & \quad \text{c-fact by applying (2.12)} \quad (2.16) \\
I(\langle c_3, c_6 \rangle) & \quad \text{c-fact by applying (2.12) on (2.15) and (2.16)} \quad (2.17)
\end{align*}
\]

But we will never be able to construct a fact \( I(\langle d, c' \rangle) \) for any \( c' \) because we cannot construct the fact \( q(d) \).
2 Protocol Model and Main Result

Figure 2.1: The run of a protocol: The intruder derives a message $m_1$, a principal applies a protocol step $t_1 \rightarrow r_1(s_1)$ and derives a message $m_2$ sent to the intruder and so on.

2.3.3 Principals and Protocols

Now we can define principals which mainly consists of a rewrite rule which is then processed by a selecting theory.

**Definition 2.11.** A protocol step $\tau$ over $(Q, R)$ is of the form $(t, r, s)$, where $t, s \in T(\Sigma, X)$ are terms and $r \in R$ is a push symbol.

A principal $\vec{\tau}$ over $(Q, R)$ is a finite sequence $[\tau_1, \ldots, \tau_N]$ of protocol steps such that, for every $n \in \{1, \ldots, N\}$ and $\tau_n = (t_n, r_n, s_n)$ we have

$$\text{var}(s_n) \subseteq \bigcup_{i=1}^{n} \text{var}(t_i). \quad (2.18)$$

A protocol over $(Q, R)$ with register size $Z$ is a pair $\mathbb{P} = (P, \Phi)$ consisting of a finite set of principals $P$ and a selecting theory $\Phi$ using anonymous constants over $(Q, R)$ with register size $Z$.

By abuse of notation we will write $t \rightarrow r(s)$ instead of $(t, r, s)$ even if $r(s)$ is no atomic formula because of the arity of $r$.

We will now explain the behavior of principals in our model – see Figure 2.1 for illustration, with the intruder theory $\Phi_I$ as introduced in the next section.

Let $\Phi$ be a selecting theory with anonymous constants over $(Q, R)$. Assume that we have a principal which has a protocol step $t \rightarrow r(s)$ with $r \in R$, and he is receiving a term $\sigma(t)$ for a ground substitution $\sigma$. Let $\kappa$ be a register containing fresh anonymous constants, i.e., constants not used so far in the run of the protocol. The principal will now send each term $s'$ to the network for which $r(\sigma(s), \kappa) \vdash_{\Phi} I(s')$ holds – for a set of fresh anonymous constants $C$ not used so far and not used to construct any other term $s''$. 
In fact, there could be an infinite number of messages \( s' \) as there are infinitely many possibilities to choose each anonymous constant – but we will show that for analyzing security it is sufficient to check a bounded number of messages. Note that the number of messages sent to the network in one protocol step is not bounded in terms of the size of the protocol, even if we only consider messages that do not contain any anonymous constant.

**Remark 2.12.** It is also possible to model non-recursive protocol steps of the form \( t \rightarrow I(s) \): For each such step \( \tau \), we take an unused push symbol \( r_\tau I \in R \) and add the following send clause to the selection theory:

\[
r_\tau I(x, k_\ast) \rightarrow I(x) .
\]  
(2.19)

This allows us to express \( t \rightarrow I(s) \) by \( r \rightarrow r_\tau I(s) \).  
\( \triangleright \)

**Remark 2.13.** For the rest of this thesis we will consider the sets of predicate symbols \( Q \) and \( R \) as well as the register size \( Z \) to be fixed, i.e., when referring to a protocol we will omit »over \( (Q, R) \) with register size \( Z \)«.  
\( \triangleright \)

### 2.3.4 The Intruder

Our threat model is the same as presented in Section **1.2**. Thus, we will describe an intruder that is able to read, modify, delete, replay or create all messages sent over the network. But, according to the level of abstraction of the Dolev-Yao model, he is not able to apply any form of cryptanalysis or similar, e.g., he is only able to read and decompose messages that are not encrypted at all or encrypted with keys he knows. These abilities of the intruder, i.e., to decompose messages and generate new messages, are also modeled by a Horn theory.

**Definition 2.14.** The intruder theory \( \Phi_I \) is defined as follows for \( x, x_1, x_2 \in X \) and each key \( k \in K \):

\[
\begin{align*}
I((x_1, x_2)) & \rightarrow I(x_1), & I(\{x\}_k), I(k) & \rightarrow I(x), \\
I((x_1, x_2)) & \rightarrow I(x_2), & I(\{x\}_k), I(k^{-1}) & \rightarrow I(x), \\
I(x_1), I(x_2) & \rightarrow I((x_1, x_2)), & I(x), I(k) & \rightarrow I(\{x\}_k), \\
I(x) & \rightarrow I(\text{hash}(x)), & I(x), I(k) & \rightarrow I(\{x\}_k). 
\end{align*}
\]  
(2.20a)
(2.20b)
(2.20c)
(2.20d)

For a set of messages \( A \), let \( I(A) = \{ I(t) \mid t \in A \} \). We define that the intruder can derive a message \( t \) from \( A \) if \( I(A) \vdash_{\Phi_I} I(t) \).  
\( \triangleright \)

As later stated in Lemma **3.1** this derivation can be divided into two phases: decomposition, clauses **(2.20a)** and **(2.20b)**, followed by composition, clauses **(2.20c)** and **(2.20d)**.

Note that we have no special clauses for anonymous constants – for the intruder, these constants are not distinguishable from regular constants. By the principal described in **2.4.1**
we enable the intruder to use anonymous constants, but this is possible within the means presented so far.

### 2.3.5 Attacks and Security

**Definition 2.15.** Let $P = (P, \Phi)$ be a protocol with $M$ principals $\bar{\tau}^1, \ldots, \bar{\tau}^M$, where for each $m \in \{1, \ldots, M\}$ the principal $\bar{\tau}^m$ has $N_m$ protocol steps $\tau^m_1, \tau^m_2, \ldots, \tau^m_{N_m}$.

An execution scheme $\pi$ for $P$ is a sequence $[\pi_1, \ldots, \pi_{N_m}]$ of protocol steps if there is a function $f: \{1, \ldots, N\} \rightarrow \{1, \ldots, M\}$ that witnesses for each protocol step in which principal this protocol step occurs, i.e., has the following property:

$$f^{-1}(m) = \{p_1, \ldots, p_{N_m}'\}$$

with $p_1 < p_2 < \cdots < p_{N_m}'$ and $N_m' \leq N_m$ for all $m \in \{1, \ldots, M\}$, and $\pi_{p_n} = \tau^m_n$ for all $n \in \{1, \ldots, N_m\}$.

To complete our model we define the security of a protocol – keep in mind the illustration in Figure 2.1.

**Definition 2.16.** Let $(\pi, \sigma)$ be a pair with a ground substitution $\sigma$ and an execution scheme $\pi = [t_n \rightarrow r_n(s_n)]_{n=1}^N$ for a protocol $P$, and set $t_{N+1} =$.

We take the pairwise disjoint ground registers $\kappa^{(1)}, \ldots, \kappa^{(N)}$ we defined earlier and set $C_0$ to contain all anonymous constants in these $N$ registers. We further assume that $\{C_0, \ldots, C_N\}$ is a set of pairwise disjoint subsets of $\Gamma$.

Let there be another set $\{T_1, \ldots, T_N\}$ of sets of messages sent to the intruder during that step, i.e., $T_n$ is a set of facts of the form $I(t)$ for messages $t$, such that the following conditions hold:

$$r_n(\sigma(s_n), \kappa^{(n)}) \vdash^{C_n} I(t_n) \quad \text{for all } n \in \{1, \ldots, N\} \quad (2.21a)$$

$$\{I(c)\} \cup T_1 \cup \cdots \cup T_n \vdash^{\Phi_t} I(\sigma(t_{n+1})) \quad \text{for all } n \in \{0, \ldots, N\}. \quad (2.21b)$$

Then $(\pi, \sigma)$ is called an **attack** on $P$. A protocol is called **insecure** if there is an attack on it, otherwise it is called **secure**.

This concludes the definition of our model. In the next section we will single out some features of our model, whereas in Section 2.5 we will show how to model the Recursive Authentication Protocol.

## 2.4 Features of the Model

### 2.4.1 Fresh Tokens for the Intruder

In the real world, an intruder can obtain an arbitrary number of fresh tokens by simply generating them randomly. Our intruder, on the other hand, is not yet able to generate fresh anonymous constants.
We present one possible way to allow the intruder to generate constants: a principal whose only purpose is to provide the intruder with fresh tokens. We leave it up to the user of our protocol model to add such a principal if appropriate.

This affects the security of a protocol – i.e., a protocol without this principal may be secure in our model, while adding the principal makes it insecure. But as stated above, this change to the notion of »secure« is usually justified, as a real intruder is able to generate fresh tokens.

This principal has the following protocol steps, with \( \star \) being a regular variable not used elsewhere and \( \star \) being a new push symbol:

\[
x \rightarrow r \langle x \rangle .
\] (2.22)

The following clauses are added to the selecting theory with \( y^\star \) being a fresh variable and \( y_z \) being an anonymous variable for each \( z \in \{1, \ldots, Z\} \):

\[
\begin{align*}
& r \langle x_0, x_1 \rangle, (y_1, y_2, \ldots, y_Z) \rightarrow r \langle x_1, (y^\star, y_2, \ldots, y_Z) \rangle, \\
& r \langle c, x_1 \rangle, (y_1, y_2, \ldots, y_Z) \rightarrow I(y_1).
\end{align*}
\] (2.23) (2.24)

If the intruder wants to generate \( i \) anonymous constants, he constructs the following message:

\[
\langle c, \ldots, c \rangle \quad \text{(i+1) times}
\] (2.25)

Upon receiving this message the new principal replies with \( i \) messages each containing a fresh anonymous constant.

Note that our principals are not able to distinguish between different anonymous constants because there is no possibility to check anonymous constants for inequality. Hence, it would also be sufficient to allow the intruder to derive only one fresh anonymous constant.

### 2.4.2 Challenges to the Intruder

As later used in Section 2.5, we can add a principal which models security by using an additional key \( K^\Lambda \) and the following clause for the selecting theory with an unused regular variable \( x^\Lambda \):

\[
\langle x^\Lambda, \{ x^\Lambda \}_K \rangle \rightarrow I(\$).
\] (2.26)

Any principal can now claim a whole term \( t \) to remain secret by sending \( \{ t \}_K \) to the intruder: If the intruder is able to derive \( t \), then he is able to send \( \langle t, \{ t \}_K \rangle \) to the additional principal, which replies with \( \$ \).
On the other hand, the intruder is not able to extract \( t \) from \( \{ t \}_{K_A} \) because of the assumption that the key \( K_A \) is not used anywhere else in the protocol.

2.4.3 Non-Flat Push Clauses

When modeling protocols it is useful to allow non-flat push clauses, i.e., clauses of the following form with \( t \in T(\Sigma, X) \) and \( x \in \text{var}(t) \) (and predicate symbols and registers like in \( \Sigma \)):

\[
q_1(t), \ldots, q_m(t), r(t, \kappa) \rightarrow r'(x, \kappa').
\]  

(2.27)

We will show how to transform such a clause to a sequence of flat push clauses, e.g. allowing the use of non-flat push clauses when modeling protocols. However, non-flat push clauses have to be linear, i.e., each variable in \( t \) occurs exactly once in \( t \). We will also use this method later in Section 3.4.

Let \( P = (P, \Phi) \) be a protocol, and let \( \varphi \) be a non-flat push clause, i.e., for a linear term \( t \) we have \( \varphi = [q_1(t), \ldots, q_m(t), r(t, \kappa) \rightarrow r'(x, \kappa')] \).

As \( x \in \text{sub}(t) \), there is a sequence \( [t_1, \ldots, t_m] \) of terms with \( t = t_1, t_m = x \) with \( t_{j+1} \in \text{sub}(t_j) \) and \( \text{depth}(t_{j+1}) = \text{depth}(t_j) - 1 \) for all \( j \in \{1, \ldots, m-1\} \). We now choose terms \( s_1, \ldots, s_{m-1} \) such that for all \( j \in \{1, \ldots, m-1\} \) we can construct \( t_j \) from \( s_j \) by substituting \( x \in \text{var}(s_j) \) with \( t_{j+1} \).

If, for instance, \( t = \langle x', (\{x\}_k, x'') \rangle \), we would have \( m = 4 \) and the following terms:

\[
\begin{align*}
  t_1 &= \langle x', (\{x\}_k, x'') \rangle, & s_1 &= \langle x', x \rangle, \\
  t_2 &= (\{x\}_k, x''), & s_2 &= (x, x''), \\
  t_3 &= \{x\}_k, & s_3 &= \{x\}_k, \\
  t_4 &= x.
\end{align*}
\]

Then we take \( m - 2 \) push symbols \( r_2, \ldots, r_{m-1} \in R \) not used before and replace \( \varphi \in \Phi \) by the following \( m - 1 \) clauses:

\[
\begin{align*}
q_1(t), \ldots, q_m(t), r(s_1, \kappa) &\rightarrow r_2(x, \kappa'), \\
 r_2(s_2, \kappa) &\rightarrow r_3(x, \kappa_*), \\
 &\vdots \\
 r_{m-2}(s_{m-2}, \kappa_*) &\rightarrow r_{m-1}(x, \kappa_*), \\
 r_{m-1}(s_{m-1}, \kappa_*) &\rightarrow r'(x, \kappa_*).
\end{align*}
\]  

(2.28) - (2.32)

In these clauses we use \( \kappa_* \) from Definition 2.4 to just copy an unmodified version of the register from one fact to the next one.
As the predicate symbols \( r_2 \) to \( r_{m-1} \) are not used elsewhere, this transformation does only affect the internal structure of the selecting theory; i.e., when modeling protocols with non-flat push clauses and applying this transformation, this has no effect on the protocol’s security.

2.5 Modeling the Example Protocol

To illustrate our extension in comparison with [Tru05b], we give a formal model of the Recursive Authentication Protocol as described in Section 2.1. Our model presented here is slightly imprecise as the server does originally send a list of messages, which cannot be modeled by the means available – our server just sends each message on its own. Additionally, our principals do not forward the server’s messages to their predecessors. One can show that both imprecisions are not relevant for the security of the protocol.

We will start by defining the protocol in the basic model presented in [Tru05b] and then show how our extension of the model improves the analysis.

2.5.1 Basic Model of the Protocol

We assume that there are \( N + 1 \) principals \( \{P_1, \ldots, P_N, P_S\} \) using this protocol, with \( P_S \) being the server (again, \( S = N + 1 \)). Besides the constants and function symbols from Definition 2.8, the signature \( \Sigma \) contains the following constants:

- for each \( n \in \{1, \ldots, N\} \) the principal’s name \( P_n \), a nonce \( \tilde{N}_n \), and a key \( K_n \) which is shared between the principal and the server,
- for each possible pair of principals \( (n, m) \) with \( S \neq n \neq m \), a key \( K_{(n,m)} \) which can be sent to the principals by the server,
- the symbol \( \bot \) necessary for the initialization of the protocol, and
- a key \( K_\Lambda \) which is used for formulating challenges.

Let \( x_S, x_\Lambda, x_{(a,b)}, x_{(a,b,c,d)} \) be variables for \( a, b, c \in \{1, \ldots, N, S\} \) and \( d \in \{1, \ldots, 5\} \). We will now define the protocol steps and the selecting theory of the protocol:

Security

For modeling security as explained in Section 2.4.2, we first add a principal which uses the key \( K_\Lambda \), i.e., the principal has only one protocol step:

\[
\langle x_\Lambda, \{x_\Lambda\}_{K_\Lambda} \rangle \rightarrow I(\$).
\]

(2.33)

The following principals can now state »The term \( t \) should remain secret« by sending \( \{t\}_{K_\Lambda} \) to the intruder: As long as the key \( K_\Lambda \) is, besides in this principal, only used for
2 Protocol Model and Main Result

Initiator

Protocol steps for each $n \in \{1, \ldots, N\}, m \in \{1, \ldots, N, S\}$ (with $x = x_{(n,m)}$):

\[
\hat{x} \rightarrow I(\text{hash}_{\hat{x}_1}(P_n, P_m, \bar{N}_n, \perp)) ,
\]
\[
\{\hat{x}, P_n, P_m, \bar{N}_n\}_{K_{\hat{x}}} \rightarrow I(\{\hat{x}\}_{K_{\hat{x}}}) .
\]

The last step models that the term $\hat{x}$ sent to the principal in step 2 should remain secret during the run of the protocol – this is necessary because the initiator would use $\hat{x}$ as the key established by this protocol. Now if the intruder is able to derive $\hat{x}$, he will also be able to derive $\$ \$ because of the first principal, resulting in the protocol being called insecure.

Inner Principals

Protocol steps for each $n, m_1 \in \{1, \ldots, N\}$ and $m_2 \in \{1, \ldots, N, S\}$ (with $\bar{x}_I = x_{(n,m_1,m_2,l)}$):

\[
\text{hash}_{\bar{x}_1}(P_{m_1}, P_n, \bar{x}_2, \bar{x}_3) \rightarrow I(\text{hash}_{\bar{x}_1}(P_{m_1}, P_{m_2}, \bar{N}_n, \text{hash}_{\bar{x}_1}(P_{m_1}, P_n, \bar{x}_2, \bar{x}_3))) ,
\]
\[
\langle \{\bar{x}_4, P_{m_1}, P_n, \bar{N}_n\}_{K_{\bar{x}_1}}, \{\bar{x}_5, P_{m_2}, \bar{N}_n\}_{K_{\bar{x}_1}} \rangle \rightarrow I(\{\bar{x}_4\}_{K_{\bar{x}_1}}, \{\bar{x}_5\}_{K_{\bar{x}_1}}) .
\]

Both $\bar{x}_4$ and $\bar{x}_5$ correspond to keys received from the server and thus should remain secret, as again stated by the last send action.

Server

The server is modeled with only one protocol step:

\[
x_S \rightarrow r_S(x_S) .
\]

The following clauses are used as the selecting theory of the protocol for each $n, m_1 \in \{1, \ldots, N\}$ and $m_2 \in \{1, \ldots, N, S\}$:

\[
r_S(\text{hash}_{\bar{x}_1}(P_{m_1}, P_{m_2}, x_1, x_2)) \rightarrow r_S(x_2) ,
\]
\[
r_S(\text{hash}_{\bar{x}_1}(P_{m_1}, P_{m_2}, x_1, \text{hash}_{\bar{x}_1}(P_{m_1}, P_n, x_2, x_3))) \rightarrow I(\{K_{(m_1,n)}\}, \{P_{m_1}, P_n, x_1\}_{K_{\bar{x}_1}} ,
\]
\[
I(\{K_{(m_2,n)}\}, \{P_{m_2}, x_2\}_{K_{\bar{x}_1}} ,
\]
\[
I(\{K_{(m_1,n)}\}_{K_{\bar{x}_1}} , I(\{K_{(n,m_2)}\}_{K_{\bar{x}_1}}) ,
\]
\[
r_S(\text{hash}_{\bar{x}_1}(P_n, P_{m_2}, x_1, \perp)) \rightarrow I(\{K_{(n,m)}\}, \{P_n, P_{m_1}, x_1\}_{K_{\bar{x}_1}} ,
\]
\[
I(\{K_{(n,m)}\}_{K_{\bar{x}_1}}) .
\]
Additional Principal: Constants

An additional principal is necessary to allow the intruder to gain knowledge of the commonly used constants, i.e., the names of the principals and ⊥:

$$\zeta \rightarrow I(\bot, P_1, \ldots, P_N, P_S).$$  \hspace{1cm} (2.42)

2.5.2 Extending the Model of the Protocol

We will now modify the selecting theory presented in the previous section to use our extended model. We use $Z = 2$ as the register size. We still need most of the constants defined above, but not the keys $K_{(n,m)}$.

The protocol’s principals are modeled in exactly the same way, we only have to change the clauses of the selecting theory. Hence, in our model we use the following selecting theory, replacing the clauses (2.39) to (2.41):

$$r_S(\text{hash}_{K_i}(P_i, P_j, x_1, x_2), (y_1, y_2)) \rightarrow r_S(x_2, (y_2, y_*)), \quad I(\{y_1\}_{K_i})$$  \hspace{1cm} (2.43)

$$r_S(\text{hash}_{K_i}(P_i, P_j, x_1, \text{hash}_{K_j}(P_j, P_i, x_2, x_3)), (y_1, y_2)) \rightarrow I(\{y_2, P_j, P_i, x_1\}_{K_i}), \quad I(\{y_1, P_i, P_j, x_2\}_{K_i})$$  \hspace{1cm} (2.44)

$$r_S(\text{hash}_{K_i}(P_i, P_j, x_1, \bot), (y_1, y_2)) \rightarrow I(\{y_1, P_i, P_j, x_1\}_{K_i}).$$  \hspace{1cm} (2.45)

We also add a principal that is generating anonymous constants for the intruder as described in Section 2.4.1.

This model of the protocol is slightly more accurate than the one presented first. Indeed, we did not model the nonces used by the principals by anonymous constants. But this is not necessary because the total number of nonces used by the principals in the protocol is bounded by the number of principals.

But the key generation of the server is modeled using anonymous constants – thus, we eliminated the problem described in Section 2.1.1 i.e., the server will no longer »generate« the same key twice.

Note that in this protocol such a situation can only occur if there are multiple principals with the same name, which would be the case if we had $P_i = P_j$ in our model for $i \neq j$, or if the intruder would be able to modify or generate a message to the server such that it contains multiple requests for a single connection as shown in Section 2.1.1.
2.6 Main Result

Theorem 2.17 (Main Theorem). Let $P$ be a protocol using anonymous constants. For a bounded number of principals, there is an algorithm that decides in nondeterministic double exponential time if $P$ is secure.

To prove this theorem, we will later describe the algorithm which nondeterministically guesses a possible attack and checks if the guess is an attack. To do so, we present a structure called directed acyclic graph of the attack ($\text{ADAG}$) introduced in [Tru05a] and extended in [Tru05b], which represents an attack on a protocol.

The proof has two main steps: First we will show that a protocol is insecure if and only if there is an $\text{ADAG}$ for that protocol. Second we will prove that for each $\text{ADAG}$ we find one of at most double exponentially bounded size, resulting in the possibility to guess and examine each $\text{ADAG}$ in nondeterministic double exponential time.

More precisely, in Chapter 3 we will begin with transforming the definition of an attack presented above to a characterization more suitable for defining an $\text{ADAG}$. This transformation consists of two steps: First, we will merge the selecting theory of a protocol with the intruder theory, resulting in a Horn theory called the theory of a protocol (see 3.2) which can directly obtain the additions to the intruder’s knowledge from a protocol step $t \rightarrow r(s)$. Then we will extend this Horn theory to a selecting theory called stage theory (see 3.3), which is not modeling one protocol step at a time, but the whole run of the protocol in one application of the theory. The characterization of an attack by a stage theory is then related to an $\text{ADAG}$, which is a graph structure with special labeling functions, altogether restricted by a set of local properties defined in 3.4.

Subsequently, in Chapter 4 we will show how to scale down $\text{ADAGs}$. In 4.2 an $\text{ADAG}$ will be transformed to simple $\text{ADAGs}$, i.e., we further restrict the structure of an $\text{ADAG}$. In 4.3 we scale down the number of a certain kind of nodes called goals, and finally in 4.4 we will show how to scale down the number of all nodes in an $\text{ADAG}$ by merging nodes until the whole $\text{ADAG}$ has a bounded size. Finally in 4.5 we can present an algorithm which decides security by nondeterministically guessing and checking $\text{ADAGs}$. 
3 The Graph of an Attack

3.1 The Intruder Theory

We will begin with the proof of the following lemma mentioned in the definition of the intruder in Section 2.3, which is based on [CJM98]:

**Lemma 3.1 (Intruder Theory).** Let $A$ and $B$ be sets of messages. It holds $A \vdash_{\Phi_I} B$ if and only if there exists a Horn proof $b_1, \ldots, b_n$ of $B$ with respect to $\Phi_I$ assuming $A$ such that there are numbers $l_1 \leq l_2 \leq n$ with the following properties:

- $b_1, \ldots, b_{l_1} \in A$ are a-facts,
- $b_{l_1+1}, \ldots, b_{l_2}$ are c-facts obtained by clauses $(2.20a)$ and $(2.20b)$, and
- $b_{l_2+1}, \ldots, b_n$ are c-facts obtained by clauses $(2.20c)$ and $(2.20d)$.

**Proof.** Let $A$ and $B$ be sets of messages. Clauses $(2.20a)$ and $(2.20b)$ will be called decomposition clauses, clauses $(2.20c)$ and $(2.20d)$ composition clauses.

One implication of the lemma directly holds: If there exists a Horn proof with the restrictions mentioned in the lemma, then we have $A \vdash_{\Phi_I} B$.

We show the other implication: We assume that $A \vdash_{\Phi_I} B$. This means there is a Horn proof $b_1, \ldots, b_n$ of $B$ with respect to $\Phi_I$ assuming $A$, we now show that there is a sequence $[b'_1, \ldots, b'_n] \subseteq \{b_1, \ldots, b_n\}$ which still is a Horn proof of $B$, but with $l_1 \leq l_2 \leq m$ as mentioned above.

By definition of a Horn proof, all the a-facts $b_i \in A$ occurring in the proof do not depend on other facts in the proof, hence, we can assume that we find an index $l_1$ with $b_1, \ldots, b_{l_1} \in A$ and $b_{l_1+1}, \ldots, b_n \not\in A$.

As long as we do not find an $l_2$ satisfying the condition in the lemma for $b_{l_1+1}, \ldots, b_n$, we repeat the following steps:

- First we find an index $i > l_1$ with $b_i$ being obtained by a composition clause, but $b_{i+1}$ being obtained by a decomposition clause – we call such a situation an inversion. Such an index exists because we have moved all facts from $A$ to the beginning of the Horn proof and we cannot find an index $l_2$. 
Merging the selecting theory

If the fact \( b_i \) be produced by a composition clause \( c_1 \rightarrow c_0 \) or \( c_1, c_2 \rightarrow c_0 \); and \( b_{i+1} \) be produced by a decomposition clause \( d_1 \rightarrow d_0 \) or \( d_1, d_2 \rightarrow d_0 \).

Consider two cases:

- If \( c_0 \neq d_1 \), then \( b_{i+1} \) does not rely on \( b_i \): We find an anonymous substitution \( \sigma \) and sets with \( \sigma(d_0) = b_{i+1} \), and with \( \sigma(d_1), \sigma(d_2) \in \{b_1, \ldots, b_{i-1}\} \) (analogously if there is no \( d_2 \)).

Then we can swap both steps: We set \( n' = n, b'_i = b_{i+1}, \) and \( b'_{i+1} = b_i, \) as well as \( b'_j = b_j \) for all \( j \neq i, i+1. \)

- If \( c_0 = d_1 \), then \( b_{i+1} \) is a fact which is already known before \( b_i \) is composed, i.e., we have \( d_0 \in \{c_1, c_2\} \) and we can delete step \( i+1 \) of the Horn proof: We set \( n' = n - 1, b'_j = b_j \) for all \( j \leq i, \) and \( b'_j = b_{j+1} \) for all \( j > i. \)

We now repeat these steps as long as there is an index \( i \) as above, using \( b'_1, \ldots, b'_m \) as the new Horn proof in the next iteration.

In every iteration we shorten the proof by one fact or eliminate an inversion, hence, we can have no more than \( n^2 \) iterations. As soon as there is no index \( i \) left, the proof meets the requirements mentioned in the lemma.

\[ \square \]

### 3.2 The Theory of a Protocol

In our model each of the intruder’s steps consists of composing a term and sending it to a principal, after which the principal applies a protocol step and uses the selecting theory to determine which terms are sent to the intruder.

As shown in Figure 3.1, we now merge two steps: After a principal applies the selecting theory \( \Phi \) of the protocol, the intruder uses his theory \( \Phi_1 \) to derive new messages. Because both \( \Phi \) and \( \Phi_1 \) are Horn theories, we can merge them into one Horn theory, the theory of a protocol.
In order to bound the ADAG in later steps of our proof, we introduce an optimization: If the selecting theory contains a send clause constructing a term \( s \), the intruder may be able to decompose this term – possibly because he already learned some keys used in the protocol – and derive a subterm \( s' \in \text{sub}(s) \). Therefore, we add an additional send clause to the theory of the protocol which directly sends the term \( s' \) if the intruder knows the necessary keys.

First we define a relation on the set of keys expressing accessibility: For a send clause \( \varphi \) and a set of keys \( K \) we need to know which subterms of the send term can be derived by the intruder after the clause \( \varphi \) has been applied, assuming the intruder already knew the keys in \( K \).

**Definition 3.2.** Let \( K \subseteq \text{sub}(t) \) be a set of keys, we will inductively define the relation \( \models_K \subseteq T_V \times T_V \). Let \( s \) and \( t \) be terms such that \( s \in \text{sub}(t) \), then we set:

\[
\begin{align*}
\models t \equiv_0 t, \\
\models \langle t, s \rangle \equiv_0 t, \\
\models \{ s \}_t \equiv_{\{ t \}} s, \\
\models \langle t, s \rangle \equiv_0 s, \\
\models \{ s \}_t \equiv_{\{ t^{-1} \}} s.
\end{align*}
\]

(3.1) (3.2)

If \( t_1 \models_{K_1} t_2 \) and \( t_2 \models_{K_2} t_3 \), we define \( t_1 \models_{K_1 \cup K_2} t_3 \), if not already \( t_1 \models_{K_3} t_3 \) for a set \( K_3 \subseteq K_1 \cup K_2 \).

For example, we have \( a \models_0 a, \{ a \}_k \models_{\{ k \}} a \), and \( \langle k_1, \{ \{ a \}_{k_1} \}_{k_2} \rangle \models_{\{ k_2^{-1} \}} a \).

In addition, we need to generalize the notion of push and send clauses:

**Definition 3.3.** A generalized push or send clause is a push or send clause where

- facts of the form \( I(k) \) for keys \( k \in K \) are allowed on the left-hand-side,
- only ground registers are allowed on the right-hand-side of a push clause, and
- simple terms are allowed in push clauses, i.e., we allow non-flat linear terms.

For a variable register \( \kappa \), a ground register \( \kappa' \), some \( n, m \in \mathbb{N} \), predicate symbols \( q_1, \ldots, q_m \in Q \), \( r, r' \in R \), keys \( k_1, \ldots, k_n \in K \), simple terms \( t_r, t_1 \in T(\Sigma, X) \) as well as \( s_1 \in T(\Sigma, X \cup Y) \) with \( \text{var}(s) \subseteq \text{var}(t) \cup \text{var}(\kappa) \), and a variable \( x \in \text{var}(t_r) \) we define:

\[
\begin{align*}
generalized \ \text{push clauses} & \quad \langle k_1, \ldots, k_n, q_1(t_r), \ldots, q_m(t_r), r(\ell_1, \kappa) \rangle \rightarrow r'(x, \kappa'), \\
generalized \ \text{send clauses} & \quad \langle k_1, \ldots, k_n, q_1(t_1), \ldots, q_m(t_1), r(t_1, \kappa) \rangle \rightarrow I(s_1).
\end{align*}
\]

(3.3 b, c)

Now we can define the theory of a protocol.

**Definition 3.4.** Let \( \mathbb{P} = (P, \Phi) \) be a protocol. Let \( r_1 \) be a new binary push symbol. Let \( c_1 \in \Gamma \) be an anonymous constant, leading to the unique register \( \kappa^{(1)} = (c_1, \ldots, c_1) \in \{ c_1 \}^2 \).

The **theory** \( \Phi_{\mathbb{P}} \) of the protocol \( \mathbb{P} \) consists of the clauses \( (3.4a) \) to \( (3.4b) \).
• intruder pop clauses for each key $k$

\[
\begin{align*}
I(x_1), I(x_2) &\rightarrow I((x_1, x_2)) , \\
I(x) &\rightarrow I(\text{hash}(x)) , \\
I(x), I(k) &\rightarrow I(\{x\}_k) ,
\end{align*}
\]  
(3.4 i a, b)

• the intruder send clause and generalized intruder push clauses for each key $k$

\[
\begin{align*}
r_I(x, \kappa_s) &\rightarrow I(x) , \\
r_I((x_1, x_2), \kappa_s) &\rightarrow r_I(x_1, \kappa^{(1)}) , \\
r_I((x_1, x_2), \kappa_s) &\rightarrow r_I(x_2, \kappa^{(1)}) , \\
I(k), r_I(\{x\}_k, \kappa_s) &\rightarrow r_I(x, \kappa^{(1)}) , \\
I(k^{-1}), r_I(\{\{x\}_k\}_k, \kappa_s) &\rightarrow r_I(x, \kappa^{(1)}) ,
\end{align*}
\]  
(3.4 ii a, b, c)

• each pop clause in $\Phi$

\[
q_1(x_1), \ldots, q_n(x_n) \rightarrow q(f(x_1, \ldots, x_n)) ,
\]  
(3.4 iii)

• each push clause in $\Phi$

\[
q_1(t), \ldots, q_n(t), r(t, \kappa) \rightarrow r'(x, \kappa') ,
\]  
(3.4 iv)

• for each send clause $[q_1(t), \ldots, q_n(t), r(t, \kappa) \rightarrow I(s)] \in \Phi$, each $s' \in \text{sub}(s)$ and each $K = \{k_1, \ldots, k_l\}$ with $s \equiv_K s'$, add the following generalized send clause (case a) or generalized push clause (case b) to $\Phi_P$:

\[
\begin{align*}
I(k_1), \ldots, I(k_l), q_1(t), \ldots, q_n(t), r(t, \kappa) &\rightarrow I(s') & \text{if } s' \notin X , \\
I(k_1), \ldots, I(k_l), q_1(t), \ldots, q_n(t), r(t, \kappa) &\rightarrow r_I(s', \kappa^{(1)}) & \text{if } s' \in X .
\end{align*}
\]  
(3.4 v a, b)

As we need to clarify in which steps of a protocol execution fresh tokens are generated, we define two subsets of the theory of a protocol: The only clauses that contain fresh variables are the clauses of the form (3.4 iv); the only prerequisites for applying these clauses may be pop clauses. Therefore, we write $\text{Fresh}(\Phi_P)$ for the set of all clauses in $\Phi_P$ which are of the form (3.4 iii) or (3.4 iv). In addition, we write $\text{Basic}(\Phi_P)$ for the set $\Phi_P$ without the push clauses of the form (3.4 iv).

We now formulate the following lemma, which characterizes attacks on protocols using the theory of a protocol:

**Lemma 3.5 (Theory of a Protocol).** Let $P$ be a protocol, let $\pi = [t_n \rightarrow r_n(s_n)]_{n=1}^N$ be an execution scheme, and use $t_{N+1} = \$$. Let $\sigma$ be a ground substitution.

We take the pairwise disjoint ground registers $\kappa^{(1)}, \ldots, \kappa^{(N)}$ defined in Section 2.2.5 and set $C_0$ to contain all anonymous constants in these $N$ registers. We further assume that there is a set $\{C_0, \ldots, C_N\}$ of pairwise disjoint subsets of $\Gamma$. 
Then $(\pi, \sigma)$ is an attack if and only if there are sets of facts $R_n$ for each $n \in \{1, \ldots, N\}$ with

\[
\begin{align*}
&\text{for all } n \in \{1, \ldots, N\} \quad (3.5\ a) \\
&\{I(c)\} \cup R_1 \cup \cdots \cup R_n \vdash_{\text{Basic}(\Phi_P)} I(\sigma(t_{n+1})) \quad \text{for all } n \in \{0, \ldots, N\}. \quad (3.5\ b)
\end{align*}
\]

### 3.2.1 Proof of Lemma 3.5, Part 1

**Proof.** Let $\mathcal{P} = (P, \Phi)$ be a protocol, let $\pi = [t_n \rightarrow r_n(s_n)]_{n=1}^N$ be an execution scheme, set $t_{N+1} = \mathcal{S}$. Let $\sigma$ be a ground substitution.

First, we assume $(\pi, \sigma)$ to be an attack. Then let $\{C_1, \ldots, C_N\}$ be the set of subsets of $I$ from the definition of an attack (2.21 a). We show that conditions (3.5 a) and (3.5 b) hold for every $n \in \{0, \ldots, N\}$ for a set $R_n$ we will define and for the $C_n$ as defined above.

First consider $n = 0$. We have nothing to show for condition (3.5 a). By (2.21 b) we know that $I(c) \vdash_{\Phi_P} I(\sigma(t_1))$, let $\Pi^0_n$ be a Horn proof of that. Using Lemma 3.1 and the fact that $\epsilon$ is a constant, we know that there is no decomposition in $\Pi^0_n$, hence, all clauses used in the proof are composition clauses, which are of the form (2.20 a) and (2.20 c). As the composition clauses are elements of Basic$(\Phi_P)$, called intruder pop clauses, we also have $I(c) \vdash_{\text{Basic}(\Phi_P)} I(\sigma(t_1))$ and $\Pi^0_n$ is a Horn proof of that, which is our claim for $n = 0$.

Now let $T_1, \ldots, T_N$ be sets of facts such that for all $n \in \{1, \ldots, N\}$ the sequence $\Pi^A_n$ is a proof of (2.21 a) and $\Pi^B_n$ is a proof of (2.21 b) with the restriction mentioned in Lemma 3.1.

We define $\bar{\Pi}^A_n$ as the Horn proof $\Pi^A_n$ without all the $c$-facts constructed by send clauses; then let $R_n$ be the set of facts in $\bar{\Pi}^A_n$ for each $n \in \{1, \ldots, N\}$.

We now take an $n \in \{1, \ldots, N\}$. Let $\Pi^A_n \setminus \bar{\Pi}^A_n = [a_1, \ldots, a_i]$ be the sequence of all $c$-facts constructed by send clauses, i.e., all terms sent to the intruder; and let $\Pi^B_n = [a_{i+1}, \ldots, a_N]$. We will construct a Horn proof $\bar{\Pi}^B_n = [b_1, \ldots, b_j]$ of (3.5 b) for that $n$ by the following algorithm, which uses variables $i$ and $j$ to iterate over $a_i$ and $b_j$:

1. Let $[b_1, \ldots, b_j]$ be the concatenation of $R_1, \ldots, R_n$. We set $j = j_1$, and $i = 1$.

2. Now $a_i$ is a $c$-fact, i.e., $a_i = I(t)$ for a term $t$, and we look at the send clause producing $a_i$:

   a) If it is of the form $[\ldots, r(t', \kappa) \rightarrow I(x)]$ with $x \in X$, we increment $j$ by two and set $b_{j-1} = r_1(t, \kappa^{(1)})$ as well as $b_j = a_i = I(t)$.

   b) If the send clause has the form $[\ldots \rightarrow I(s)]$ for $s \notin X$, we increment $j$ by one and choose $b_j = a_i$.

3. Now increment $i$ and repeat from step 2 on as long as the new $i$ is less than $i_1$ or equal, otherwise set $j_2 = j$ and continue.

4. a) If $a_1$ is not produced by a decomposition clause, we increment $j$ by one and set $b_j = a_i$. 
b) Otherwise, if \( a_i = I(t) \) is a c-fact produced by a decomposition clause \( \varphi \in \Phi_I \) with an anonymous ground substitution \( \sigma \), formally we have to differentiate for the different types of decomposition clauses. As example we will consider \( \varphi = [I((x_1,x_2)) \rightarrow I(x_1)] \), with \( t = \sigma(x_1) \).

i. If we find a fact of the form \( r_I(t,t',\kappa(1)) \) in \( \{b_1,\ldots,b_j\} \), we increment \( j \) by two and set \( b_{j-1} = r_I(t,\kappa(1)) \) and \( b_j = a_i \).

ii. If we do not find such a fact, there is a clause \( \psi \) in \( \Phi_P \) of the form

\[
\psi = \left[ I(k_1),\ldots,I(k_t),q_1(t_1),\ldots,q_m(t_1),r(t_1,\kappa) \rightarrow \begin{cases} I(t_2) & \text{or} \\ r_I(t_2,\kappa(1)) & \end{cases} \right]. \tag{3.6}
\]

and a substitution \( \sigma' \) with \( t = \sigma'(t_2) \). In the case \( r_I \), we increment \( j \) by two and set \( b_{j-1} = r_I(t,\kappa(1)) \) and \( b_j = a_i \). Otherwise for \( I \), we increment \( j \) by one and set \( b_j = a_i \).

We proceed analogously for all decomposition clauses, i.e., for all clauses of the form \( 2.20a \) and \( 2.20b \).

5. Now increment \( i \) and repeat from step 4 on as long as the new \( i \) is less than \( i_2 \) or equal, otherwise set \( j_3 = j \).

This algorithm produces a Horn proof of \( 3.5b \): For each \( a_i \) with \( a_i = I(t) \) we have a fact \( b_j \) with \( b_j = a_i \). We will show that the facts \( b_j \) are justified, i.e., the definition of a Horn proof holds for the sequence \( [b_1,\ldots,b_{j_3}] \).

Together with the fact that the algorithm terminates because he is iterating over a finite sequence this proves our claim.

Let \( j < j_3 \), and assume as an inductive hypothesis that the facts \( b_1,\ldots,b_j \) conform to the definition of a Horn proof. For \( j \leq j_1 \), the fact \( b_j \) is an a-fact, hence, we have nothing to show. For \( j > j_1 \) we will show that the next steps of the algorithm are correct: As \( j \) is incremented within this step, we have to show that the fact \( b_j \) (and possibly also \( b_{j-1} \)) conform to the definition of a Horn proof.

2. In this case, we have \( a_i \in \Pi^A_{\delta} \setminus \tilde{\Pi}^A_{\delta} \).

a) The fact \( b_{j-1} \) is justified as by construction of the theory of the protocol \( 3.4\ V\ b \) we have \([q_1(t'),\ldots,r(t'',\kappa) \rightarrow r_I(x,\kappa(1))] \in \text{Basic}(\Phi_P)\). The fact \( b_j \) is derived by the clause \([r_2(x,\kappa) \rightarrow I(x)] \) \( 3.4\ \text{ii a} \).

b) This step is also justified because for \( s \notin X \), the original send clause in \( \Phi \) is copied to the theory of the protocol \( \text{Fresh}(\Phi_P) \), see \( 3.4\ V\ a \).

4. In this case, we have \( a_i \in \Pi^B_{\delta} \).
a) We know that $a_i$ is either an a-fact from $T_n$ or $a_i$ is a c-fact constructed by a composition clause. The case $a_i \in T_n$ implies $b_j = a_i \in \{b_1, \ldots, b_k\}$, and hence, $b_j$ is justified. If on the other hand $a_i$ is produced by a composition clause, $b_j$ is justified by the same composition clause, as the composition clauses are elements of Basic($\Phi_B$), see (3.4.1a) and (3.4.1b), and for every fact $I(t) \in \{a_1, \ldots, a_{i-1}\}$ we have $I(t) \in \{b_1, \ldots, b_{j-1}\}$.

b) i. The justification for $b_{j-1}$ is the corresponding (generalized) intruder push clause, see (3.4.1b) and (3.4.2c); the fact $b_j$ is again derived by the clause $[r_1(x, \kappa_a) \rightarrow I(x)]$ (3.4.2a).

ii. This step is the most difficult one. We know that in the assumed Horn proof $\Pi_n^B$ the fact $a_i = I(t)$ was produced by a decomposition clause $\varphi$, we again assume that $\varphi = [I((x, y)) \rightarrow I(x)]$. We know $a_{\varphi} = I((t, t'))$ for an index $i' < i$. But as we are in case ii, there is no fact $r_1((t, t'), \kappa(I')) \in \{b_1, \ldots, b_{j-2}\}$. By algorithm, there has to be a fact $b_j' = I((t, t'))$ with $j' < j - 1$.

Now we take the justification for the fact $b_j'$. As in $\Pi_n^B$ the restrictions of Lemma 3.4.1 hold, $a_{\varphi}$ is not produced by a composition clause. By inductive hypothesis $b_j'$ has to be generated by a (generalized) send clause

$$\hat{\psi} = \left[ I(k_1), \ldots, I(k_l), q_1(t_1), \ldots, q_m(t_1), r(t_1, \kappa) \rightarrow \begin{cases} I(t_2) \text{ or } \\ r_1(t_2, \kappa(I)) \end{cases} \right]. \quad (3.7)$$

The case $r_2$ is not possible, hence, we have a clause with $I(t_2)$ on the right hand side, and a substitution $\sigma$ with $\sigma(t_2) = (t, t')$. By (3.4.3a) we know that $t_2 \notin X$, and because $t_2 \notin Y$ (in that case $t_2$ could not be mapped to a term containing a pairing symbol) we know that $t_2 = \langle t_{2a}, t_{2b} \rangle$ for terms $t_{2a}, t_{2b}$. Now by definition of the theory of the protocol, there is a clause

$$\psi = \left[ I(k_1), \ldots, I(k_l), q_1(t_1), \ldots, q_m(t_1), r(t_1, \kappa) \rightarrow \begin{cases} \text{or } I(t_{2a}) \\ r_1(t_{2a}, \kappa(I)) \end{cases} \right]. \quad (3.8)$$

For the decryption clauses amongst the decomposition clauses, we may have to add another key on the left hand side (i.e., $I(k_{l+1})$), but this key has already been used in $\Pi_n^A$ to construct $a_i$.

The clause $\psi$ can now be used in step 4.b) ii. of the algorithm.

We have proven that for every attack $(\pi, \sigma)$, the characterization in Lemma 3.5 holds.

### 3.2.2 Proof of Lemma 3.5, Part 2

We will now show the other direction of the lemma. Assume that for all $n \in \{0, \ldots, N\}$ the two conditions (3.5a) and (3.5b) hold. We now have to show that $(\pi, \sigma)$ is an attack,
i.e., we find sets \( T_n \) of facts of the form \( \Pi(I(t)) \) such that the following conditions hold:

\[
\begin{align*}
    r_n(\sigma(s_n), \kappa^{(n)}) & \vdash_{\Phi} \Pi(I(t)) \quad \text{for all } n \in \{1, \ldots, N\} \quad \text{(3.9a)} \\
    \{I(\varepsilon)\} \cup T_1 \cup \cdots \cup T_n & \vdash_{\Phi} I(\sigma(t_{n+1})) \quad \text{for all } n \in \{0, \ldots, N\}. \quad \text{(3.9b)}
\end{align*}
\]

For \( n \in \{1, \ldots, N\} \), let \( \Pi_n^A \) be a Horn proof of \( (3.5a) \), and let \( \Pi_n^B \) be a proof of \( (3.5b) \) for \( n \in \{0, \ldots, N\} \). We will construct Horn proofs \( \Pi_n^A \) and \( \Pi_n^B \) for \( (3.9a) \) and \( (3.9b) \) by transforming \( \Pi_n^A \) and \( \Pi_n^B \) by a method that preserves the semantics of each fact in these proofs.

Let \( \Pi_n^{\text{push}}, \Pi_n^{\text{send}}, \Pi_n^{\text{intruder}} \) be empty sequences for each \( n \in \{0, \ldots, N\} \), and let \( \Pi_n^{\text{pop}} \) also be an empty sequence. We will now iterate over the facts in \( \Pi_n^A \) and \( \Pi_n^B \); in every iteration we will append facts to the sets we just defined, e.g., \( \Pi_n^{\text{pop}} \).

For every \( n \in \{1, \ldots, N\} \) and each c-fact \( a \) in the concatenation \( \Pi_n^A \cdot \Pi_n^B \), we consider the following cases:

i. If \( a \) was produced by an intruder pop clause \( 3.4 \text{i.a} \) or \( 3.4 \text{i.b} \), we append \( a \) to \( \Pi_n^{\text{intruder}} \).

ii. a) If \( a \) was produced by the intruder push clause \( 3.4 \text{ii.a} \), we have nothing to do.

   b) If \( a \) was produced by a (generalized) intruder push clause \( 3.4 \text{ii.b} \) or \( 3.4 \text{ii.c} \), we know \( a = r_1(t, \kappa^{(1)}) \) for a ground term \( t \). Then we append \( \Pi(I(t)) \) to \( \Pi_n^{\text{intruder}} \).

iii. If \( a \) was produced by a pop clause \( 3.4 \text{iii} \), we append \( a \) to \( \Pi_n^{\text{pop}} \).

iv. If \( a \) was produced by a push clause from \( \Phi \) \( 3.4 \text{iv} \), we append \( a \) to \( \Pi_n^{\text{push}} \).

v. If \( a \) was produced by a (generalized) send or push clause (clauses \( 3.4 \text{v.a} \) and \( 3.4 \text{v.b} \)), we have \( a = I(\tilde{t}) \) or \( a = r_1(\tilde{t}, \kappa^{(1)}) \) for a term \( \tilde{t} \). We now take that clause \( \psi \in \Phi_{\psi} \) and the corresponding substitution \( \sigma \) with \( \sigma(t) = \tilde{t} \), and know by definition of the theory of the protocol that there is a clause \( \varphi \in \Phi \), such that we have:

\[
\begin{align*}
    \psi & = I(k_1), \ldots, I(k_l), q_1(t')_l, \ldots, q_n(t')_l, r(t', \kappa) \rightarrow \begin{cases} 
    I(\tilde{t}) \text{ or } \\
    r_1(\tilde{t}, \kappa^{(1)})
    \end{cases} , \quad \text{(3.10)} \\
    \varphi & = q_1(t'), \ldots, q_n(t'), r(t', \kappa) \rightarrow I(s) \\
    \text{with } s \models_{(k_1, \ldots, k_l)} t. \quad \text{(3.11)}
\end{align*}
\]

We append \( I(\sigma(s)) \) to \( \Pi_n^{\text{send}} \).

By \( s \models_{(k_1, \ldots, k_l)} t \) we know there is a sequence \( [s_1, \ldots, s_m] \) with \( s = s_1, s_m = t \) in which \( s_{j+1} \in \text{sub}(s_j) \) and \( \text{depth}(s_{j+1}) = \text{depth}(s_j) - 1 \) for all \( j \in \{1, \ldots, m - 1\} \). Let \( \sigma(s_j) = t_j \), we then append each \( I(t_j) \) to \( \Pi_n^{\text{intruder}} \).
This method terminates as we are iterating over finite sequences. Now we set $T_n = \Pi^\text{send}_{n}$ and define the following Horn proofs for all $n \in \{0, \ldots, N\}$:

$$
\hat{\Pi}^A_n = \Pi^\text{pop} \cdot \Pi^\text{push} \cdot \Pi^\text{send}_n \quad \text{and} \quad \hat{\Pi}^B_n = \Pi^\text{pop} \cdot \Pi^\text{send}_1 \cdot \ldots \cdot \Pi^\text{send}_n \cdot \Pi^\text{intruder}_n .
$$

Now we know that $\hat{\Pi}^A_n$ is a Horn proof of (3.9a) and $\hat{\Pi}^B_n$ is one of (3.9b). First take a look at the facts in $\hat{\Pi}^A_n$:

- The elements in $\Pi^\text{pop}$ are independent of all other facts, they are justified by the same pop clauses in $\Phi$ as in $\Phi_P$.
- All facts derived by a push clauses (3.4 iv), i.e., a clause copied from $\Phi$ to $\Phi_P$, can be derived by exactly this clause in $\Phi$ because we keep the order among the push facts, and because we put all facts derived by pop clauses in $\Pi^\text{pop}$.
- The applications of send clauses in $\Phi$ only depend on facts obtained by pop and push clauses of $\Phi$, and so the facts in $\Pi^\text{send}_i$ are justified.

Finally, as $T_n = \Pi^\text{send}_{n} \subseteq \hat{\Pi}^B_n$, this is a Horn proof for $T_n$. Now take a look at the facts in $\hat{\Pi}^B_n$:

- All facts in $\Pi^\text{pop}$ are justified as above; whereas all the facts in $\Pi^\text{send}_i$ for an index $i \leq n$ are a-facts.
- The facts appended to $\Pi^\text{intruder}_n$ in step ii. b) are justified as we transform all facts of the form $r_1(t, k^{(I)})$ to $I(t)$, such that for every application of an intruder push clause on push facts there is an equivalent application of decomposition clauses. More precisely, the prerequisites for an application of a clause $\varphi$ from (3.4 ii b) or (3.4 ii c) are transformed from $r_1(t, k^{(I)})$ to $I(t)$ and appear in $\Pi^\text{send}_n = T_n$ or earlier in $\Pi^\text{intruder}_n$, hence, in $\hat{\Pi}^B_n$ we can apply a decomposition clause corresponding to $\varphi$ from the intruder theory.
- In step v. we first have to take a look at the facts $I(k)$ for keys $k$. In the original Horn proof $\Pi^B_n$, each fact $I(k)$ has to be known prior to the application of this generalized push or send clause $\psi$. The algorithm puts this fact $I(k)$ in either $\Pi^\text{send}_n$ or $\Pi^\text{intruder}_n$, in the latter case $I(k)$ prior to the application of the clause $\psi$. Therefore, we can just assume that the necessary keys are known.

The facts added in step v. are justified as we added the result of a send clause to $\Pi^\text{send}_n$, hence, we have a copy in $T_n$ and can decompose this fact using the decomposition clauses of the intruder theory and the relevant keys.

We now have Horn proofs $\hat{\Pi}^A_n$ and $\hat{\Pi}^B_n$ for (3.9a) and (3.9b), which concludes our proof.
3.3 Stage Theory

The characterization of an attack presented in Lemma 3.5 will be transformed anew in this section. We start with an execution scheme of a protocol. First, each of the protocol steps resembles a major stage in the execution of the protocol. For each of these major stages it is now possible to state which keys are added to the intruder’s knowledge during this protocol step, allowing us to divide the major stage into multiple stages. This is illustrated in Figure 3.2.

As a consequence we can merge the \( (2 \cdot N + 1) \) Horn proofs necessary to show that the definition of an attack or the characterization in Lemma 3.5 holds, resulting in only one Horn proof. To do so annotate the predicate symbols with the stage they are used in – see Figure 3.3.

Formally we will annotate the predicate symbols in the Horn clauses with special indices: These indices contain the current protocol step. But during each step, the intruder may be able to gain knowledge of several keys – therefore, we will use indices of the form \((n, m)\) where \(n\) is the current protocol step and \(m\) is the number of keys obtained so far during that step. This gives us a partitioning of the protocol run into several stages as shown in the following example.

**Example 3.6.** A possible run of a protocol as illustrated in Figure 3.2 could be as follows:

1. \((0, 0)\) The intruder composes and sends his initial message \(m_1\) to start the protocol.
2. \((1, 0)\) The principal of the first protocol step receives \(m_1\), processes it and sends a message \(m_2\). The intruder decomposes \(m_2\) but does not learn any keys from it. He composes a message \(m_3\) and sends it.
3. \((2, 0)\) Another principal receives \(m_3\) and sends \(m_4\), containing a key \(k_1\) and the subterm \(\{k_3\}_{k_2}\). The intruder extracts \(k_1\).
(2,1) The intruder cannot extract \( k_3 \) as he does not know \( k_2 \) yet, but he composes and sends a message \( m_5 \).

(3,0) After receiving \( m_5 \), one principal sends \( m_6 \) which contains the key \( k_2 \). The intruder is able to decompose \( m_6 \) and gain knowledge of \( k_2 \).

(3,1) With the help of \( k_2 \) the intruder is able to decompose \( m_4 \) so that he can extract \( k_3 \).

(3,2) Now the intruder composes a message \( m_7 \) that may, for example, contain terms encrypted by \( k_1 \) or \( k_2 \).

(4,0) The next protocol step is executed – and so on.

Note that in the example and in the figure each of our principals is only sending one message – as stated earlier, each principal could just as well send a whole set of messages.

Remark 3.7. We must remark that up to this point, it does not seem necessary to divide a protocol step into multiple stages. But the second component of the indices is necessary to prevent cycles in Horn proofs. Even though this cannot happen in the Horn proofs we used so far because they are just sequences of facts, we will later see that in the ADAG there may be cyclic situations if we do not prevent this:

For example, consider a protocol step in which the intruder learns two keys \( k_1, k_2 \). We assume that there is a message \( m = \langle \{k_2\}_e, \{k_1\}_e \rangle \). Now in a Horn proof as defined in Definition 2.6 it is clear that the intruder can only learn \( k_1 \) and \( k_2 \) if he knew one of these keys before receiving \( m \).

But in the ADAG we loose that linearity and may get cycles, i.e., the intruder would be able to derive \( k_1 \) from the message \( m \) by using \( k_2 \), which he derived from that same message \( m \) by using \( k_1 \). By introducing stages we ensure that if \( k_1 \) is derived in stage \( e_1 \), this key can only be used in stages \( e_2 > e_1 \), avoiding the occurrence of cycles.

Now we are ready to give the formal definition of stages and stage mappings.

Definition 3.8. Let \( P \) be a protocol with an execution scheme \( \pi \) with \( N \) protocol steps, and let \( K \) be the set of keys in \( \Sigma \). Let \( E = \{0, \ldots , N\} \times \{0, \ldots , |K|\} \) be the set of stages of \( P \), let \( \prec \) be the lexicographical ordering on the set of stages \( E \), and let \( \ll \) be the ordering on the first component (and define \( \leq, \geq, >, \gg \) in the natural way):

\[
\begin{align*}
(n, m) &< (n', m') \quad \text{if } n < n' \text{ or } (n = n' \text{ and } m < m') , \\
(n, m) &\ll (n', m') \quad \text{if } n < n' , \\
n &\ll (n', m') \quad \text{if } n < n' .
\end{align*}
\]

We will denote \( \{ e' \in E | e' \leq e \} \) by \( E_{\leq e} \). For a set of predicate symbols \( R \), let \( R^{(E)} \) be the following set of predicate symbols (maintaining their arity):

\[
R^{(E)} = \{ r' | r \in R, e \in E \} .
\]
A partial function \( f: K \rightarrow E \) is called \textit{stage mapping} for \( P \) if

\[
\begin{align*}
  f(k) &\in \{ (n,m) \in E \mid n \neq 0, m \neq 0 \} \\
\end{align*}
\] (3.19)

for all \( k \). We will write \( K_f \) for the set of keys for which \( f \) is defined. Additionally, for each \( e \in E \) we will use \( K(e) = \{ k \in K \mid f(k) \leq e \} \).

The intended semantic of a stage mapping \( f \) is that for a key \( k \) with \( f(k) = e \), this key is derived in a stage \( e_k < e \) and thus can be used from stage \( e \) onwards to derive other messages or keys. This will be used during the definition of the stage theory.

From now on we will use \( Q \cup \tilde{I} \) with \( \tilde{I} = \{ 1 \}^{(E)} \) as pop symbols and \( \tilde{R} = (R \cup \{ r_\tilde{I} \})^{(E)} \) as the set of push symbols, resulting in unary predicate symbols like \( \tilde{I}(n,m) \) and binary predicate symbols like \( r^{(n,m)} \). By abuse of notation we will write \( I(n) \) if we have \( I^{(n,m)} \) for some \( m \).

\textbf{Definition 3.9.} Let \( P = (P, \Phi) \) be a protocol. Let \( E \) be the set of stages of \( P \), \( f: K \rightarrow E \) be a stage mapping for \( P \). The \textit{stage theory} \( \Phi_f \) of the protocol \( P \) and the stage mapping \( f \) is a selecting theory over \( (Q \cup \tilde{I}, \tilde{R}) \) consisting of the following clauses:

- intruder pop clauses for each \( e \in E \), each key \( k \in K(e) \), and each \( e', e'' \in E \leq e \)
  \[
  I^F(x), I^{\mu}(y) \rightarrow I^F((x,y)), \quad I^F(x) \rightarrow I^F(\text{hash}(x)), \quad (3.20 \text{ ia})
  \]
  \[
  I^F(x) \rightarrow I^F(\{x\}_k), \quad I^F(x) \rightarrow I^F(\|x\|_k), \quad (3.20 \text{ ib})
  \]

- (generalized) intruder push clauses for each \( e \in E \), each key \( k \in K(e) \), and each \( e' \in E \leq e \)
  \[
  r^F_1(x, \kappa_s) \rightarrow I^F(x), \quad (3.20 \text{ ii a})
  \]
  \[
  r^F_2((x,y), \kappa_s) \rightarrow r^F_2(x, \kappa^{(1)}), \quad r^F_2((x,y), \kappa_s) \rightarrow r^F_2(y, \kappa^{(1)}), \quad (3.20 \text{ ii b})
  \]
  \[
  r^F_1(\{x\}_k, \kappa_s) \rightarrow r^F_2(x, \kappa^{(1)}), \quad r^F_1(\|x\|_{k-1}, \kappa_s) \rightarrow r^F_2(x, \kappa^{(1)}), \quad (3.20 \text{ ii c})
  \]

- each pop clause in \( \Phi \), which is, by definition, also in \( \Phi_P \)
  \[
  q_1(x_1), \ldots, q_n(x_n) \rightarrow q(f(x_1, \ldots, x_n)), \quad (3.20 \text{ iii})
  \]

- for each push clause \( [q_1(t), \ldots, r(t, \kappa) \rightarrow r'(x, \kappa')] \in \Phi \), which, by definition, is also in \( \Phi_P \), and for each \( e = (i, 0) \in E \)
  \[
  q_1(t), \ldots, r^e(t, \kappa) \rightarrow r'^e(x, \kappa'), \quad (3.20 \text{ iv})
  \]

- for each (generalized) push clause \( [I(k_1), \ldots, I(k_l), q_1(t), \ldots, r(t, \kappa) \rightarrow r^I_2(x, \kappa^{(1)})] \in \Phi_P \), and accordingly, for each (generalized) send clause \( [\ldots \rightarrow I(s)] \in \Phi_P \), for each
$e \in E, e' = (n, 0) \in E_{\leq \ell},$ and $\{k_1, \ldots, k_l\} \subseteq K(e)$

\[ q_1(t), \ldots, r_{e'}(t, \kappa) \rightarrow \begin{cases} \Gamma^e(s) & \text{or} \\ \Gamma^e_k(x, \kappa^{(\ell)}) & . \end{cases} \tag{3.20v} \]

As we later need a unique stage $e_i$ for each fact $\Gamma^e_i(t)$ for a term $t$, we introduce the notion of normal Horn proofs. Afterwards we can formulate the lemma that links the description of an attack to the stage theory.

**Definition 3.10.** A Horn proof $\Pi$ is called normal if for each term $t$ there is at most one stage $e \in E$ with $\Gamma^e(t) \in \Pi$.

**Lemma 3.11 (Stage Theory).** Let $\mathcal{P} = (P, \Phi)$ be a protocol. Let $\pi = [t_n \rightarrow r_n(s_n)]_{n=1}^N$ be an execution scheme, set $t_{N+1} = \$. Take the pairwise disjoint ground registers $\kappa^{(1)}, \ldots, \kappa^{(N)}$ and again set $C_0$ to contain all anonymous constants in these $N$ registers. Let $\sigma$ be a ground substitution, $C \subseteq \Gamma$ a set of anonymous constants disjoint with $C_0$.

Then $(\pi, \sigma)$ is an attack if and only if there exists a stage mapping $f$ (which can be efficiently constructed) such that there is a normal Horn proof of the following statement for stages $e_n \ll n + 1$ and $e_k < f(k)$:

\[ \{ I^{(0)}(\varsigma) \} \cup \{ r^{(n,0)}(\sigma(s_n), \kappa^{(n)}) \mid n \in \{1, \ldots, N\} \} \vdash_{\Phi} C_f \]

\[ \{ \Gamma^e_k(\sigma(t_{n+1})) \mid n \in \{0, \ldots, N\} \} \cup \{ \Gamma^e_l(k) \mid k \in \text{dom}(f) \} \] \tag{3.21}

**3.3.1 Proof of Lemma 3.11, Part 1**

**Proof.** In the first part of the proof, we assume that there is an attack $(\pi, \sigma)$ on the protocol, and we construct a stage mapping $f$. To show that the condition holds we construct a Horn proof of statement (3.21), consisting of the concatenation of modified facts taken from Horn proofs used in Lemma 3.5, i.e., we annotate most of the facts with stages. Figure 3.3 illustrates this transformation.

Let $\mathcal{P} = (P, \Phi)$ be a protocol. Let $\pi = [t_n \rightarrow r_n(s_n)]_{n=1}^N$ be an execution scheme, set $t_{N+1} = \$. Take the pairwise disjoint ground registers $\kappa^{(1)}, \ldots, \kappa^{(N)}$ and again set $C_0$ to contain all anonymous constants in these $N$ registers. Let $\sigma$ be a ground substitution such that $(\pi, \sigma)$ is an attack on $\mathcal{P}$. By Lemma 3.5 we know there is a set $\{C_0, \ldots, C_N\}$ of pairwise disjoint subsets of $\Gamma$ such that:

\[ r_n(\sigma(s_n), \kappa^{(n)}) \vdash_{\text{Fresh}(\Phi_P)} C_n \]

\[ \{ I(\varsigma) \} \cup R_1 \cup \cdots \cup R_n \vdash_{\text{Basic}(\Phi_P)} I(\sigma(t_{n+1})) \] \tag{3.22a, b}

Let $\Pi^A_n$ be a Horn proof of (3.22a) for $n \in \{1, \ldots, N\}$ (and the empty sequence for $n = 0$, respectively). Let $\Pi^B_n$ be a proof of (3.22b) for $n \in \{0, \ldots, N\}$.
Let $K_n$ be the sequence of keys added to the intruder’s knowledge during step $n \in \{1, \ldots, N\}$, i.e., not known before. This means $K_n$ is a sequence such that for each $m \leq |K_n|$ we have $I(K_n[m]) \in \Pi^B\{m\}$, but also $I(K_n[m]) \not\in \Pi^B\{n\}$ for all $i < n$, with the same ordering of the keys in $K_n$ as in $\Pi^B\{n\}$. We define $M = \max \{|K_n| | n \in \{1, \ldots, N\}\}$. Now we can construct a stage mapping $f$ as

$$f: K \rightarrow \{0, \ldots, N\} \times \{0, \ldots, M\} \text{ with } f(k) = \begin{cases} (n, m) & \text{if } k = K_n[m], \\ \bot & \text{if } k \not\in \bigcup_{n=1}^{N} K_n. \end{cases} \quad (3.23)$$

We have efficiently constructed $f$ from $(\pi, \sigma)$, now we have to show that (3.21) holds by constructing a Horn proof $\hat{\Pi}$ thereof.

We take $n \in \{0, \ldots, N\}$. Let $\Pi^A_n = [a_1, \ldots, a_n]$ and $a \in \Pi^A_n$. We derive $a'$ from $a$ by replacing $r \in R \cup \{r_1, I\}$ with $r^{(n,0)}$ (which also means we know $a' = a$ if we have $a = q(t)$ for a pop symbol $q \in Q$).

Assume that $|K_n| = M'$, and let $\Pi^B_n = [b_1, \ldots, b_{M'+1}]$. We choose $j_m$ such that $b_{j_m} = I(k_m)$ for every $m \in \{1, \ldots, M'\}$, and we choose $j_0 = 0$. Now let $m \in \{0, \ldots, M'\}$ and $b \in [b_{j_m+1}, \ldots, b_{j_{m+1}}]$.

If $b$ is an $a$-fact, we derive $b'$ by replacing any $r \in R$ with $r^{(n,0)}$ and $I$ with $I^{(0,0)}$. If $b$ is a $c$-fact, we derive $b'$ from $b$ by replacing $r \in R \cup \{r_1, I\}$ with $r^{(n,m)}$. Hence, if $b = q(t)$ for a pop symbol $q \in Q$, we again have $b' = b$.

We then set $\hat{\Pi}_n = [a'_1, \ldots, a'_m, b'_1, \ldots, b'_{M'+1-1}]$. Now the concatenation $\hat{\Pi} = \hat{\Pi}_0 \cdots \hat{\Pi}_N$ is a Horn proof of (3.21).

First we check the conclusion of the Horn proof, i.e., if all facts in the conclusion occur in the proof: As $I(\sigma(t_{n+1}))$ is an element of $\Pi^B_n$ for all $n \in \{0, \ldots, N\}$, we have $I^{(n,m)}(\sigma(t_{n+1})) \in \Pi_n$ for some $m$ (and therefore, we have $I^{(n)}(\sigma(t_{n+1}))$). For each $k \in K$
with \( f(k) = (n, m) \), we have \( I(k) = b_{jn} \in \Pi_\delta^B \) by construction of \( f \) and so we have \( I^{(n,m-1)}(k) \in \hat{\Pi}_n \). We can now choose \( c_k = (n, m - 1) < f(k) \).

Now we check if each fact is justified: The facts of the form \( a' \) for some \( a \in \Pi_\delta^A \) are justified because the pushed facts are all indexed with stage \((n, 0)\), and we have \( r'(n, 0)(\sigma(s_n), k^{(n)}) \) in the assumption. The facts constructed by pop clauses are not modified.

Of the facts \( b' \) for some \( b \in \Pi_\delta^B \), the \( a \)-facts are clearly justified. For the facts constructed in \( \Pi_\delta^B \) by a clause \( \varphi \), we know that \( \varphi \) is a (generalized) push or send clause \((3.4 \text{ a})\) or \((3.4 \text{ b})\), an intruder pop or push clause, or a pop clause. In the latter case, we have \( b' = b \). In the other cases, we have a copy of \( \varphi \) in \( \Phi_f \) which has the appropriate stages as indices, as we use all \( \epsilon', \epsilon'' \leq \epsilon \) in \((3.20 \text{ a})\) to \((3.20 \text{ b})\), and we use all possible \( \epsilon' \leq \epsilon \) in \((3.20 \text{ v})\).

This shows that for each attack \((\pi, \sigma, f)\), we can efficiently construct a stage mapping \( f \), such that \((\pi, \sigma, f)\) fulfills statement \((3.21)\). But it remains to prove that we can also construct a normal Horn proof of that statement.

For each term \( t \) define \( \epsilon_t = \min \{ \epsilon \in E \mid I^\epsilon(t) \in \Pi \} \). We now obtain \( \hat{\Pi} \) from \( \Pi \) by replacing each \( I^\epsilon(t) \) for some \( \epsilon \in E \) with \( I^{\epsilon_t}(t) \). Now \( \hat{\Pi} \) is a Horn proof of \((3.21)\), and all facts in it are justified: We have \( I^{\epsilon_t}(t) \in \Pi \), and for each \( I^{\epsilon_t}(t) \) used as a prerequisite for the application of a clause \( \varphi \in \Phi_f \), we find a clause \( \varphi' \in \Phi_f \) which allows \( I^{\epsilon_t}(t) \) to be used instead.

This completes the proof of the first part of our claim.

### 3.3.2 Proof of Lemma 3.11, Part 2

We now assume that \((3.21)\) holds. Hence, we take an appropriate ground substitution \( \sigma \), a stage mapping \( f \) and an execution scheme \( \pi \) as above such that \((3.21)\) holds for \((\pi, \sigma, f)\); let \( \Pi \) be a normal Horn proof of that.

Due to Lemma \((3.5)\) we now only need to define appropriate sets \( R_n \) and \( C_n \) such that the following holds:

\[
\begin{align*}
&r_n(\sigma(s_n), k^{(n)}) \vdash_{\text{Fresh}(\Phi_f)} C_n \quad \text{ and } \quad R_n \quad \text{ for all } n \in \{1, \ldots, N\} \quad \text{(3.24 a)} \\
&\{I(\epsilon)\} \cup R_1 \cup \cdots \cup R_n \vdash_{\text{Basic}(\Phi_f)} I(\sigma(t_{n+1})) \quad \text{ for all } n \in \{0, \ldots, N\} . \quad \text{(3.24 b)}
\end{align*}
\]

We divide \( \Pi \) in several parts and modify them to get the Horn proofs for theses statements.

We first define a subsequence \( \Pi^{\text{pop}} \subseteq \Pi \) of all the facts of the form \( q(t) \) for a pop symbol \( q \in Q \) and a term \( t \), preserving the relative ordering amongst the elements of \( \Pi^{\text{pop}} \).

For all \( n \in \{0, \ldots, N\} \), let \( \Pi_n^A \subseteq \Pi \) be the subsequence of all facts of the form \( r'(n, 0)(t, \kappa) \) for a push symbol \( r \in R \) (note that \( r_1 \notin R \)). Accordingly, let \( \Pi_n^B \subseteq \Pi \setminus \Pi_n^A \) be the set of facts with a predicate symbol \( p^{(n,m)} \), ordered by their stage, but preserving the ordering amongst facts of the same stage.

Now \( \Pi_n^A \) is nearly a Horn proof of \((3.24 \text{ a})\) for each \( n \in \{1, \ldots, N\} \), we only need to remove the stage annotation. Let \( \xi \) be a substitution on terms (extended to sequences of terms)
which replaces each predicate symbol of the form \( r^e \) with the according symbol \( r \). We now use these substitutions to derive \( \hat{\Pi}^{\text{pop}} = \zeta(\Pi^{\text{pop}}) \) and \( \hat{\Pi}_n^A, \hat{\Pi}_n^B \) in the same manner.

Now with \( R_n = \hat{\Pi}_n^A \), the sequence \( \hat{\Pi}^{\text{pop}} \cdot \hat{\Pi}_n^A \) is a Horn proof of (3.24a) for all \( n \in \{1, \ldots, N\} \): This sequence contains only facts which can be derived by clauses in \( \text{Fresh}(\Phi_P) \), as the sequence \( \Pi^{\text{pop}} \) only contains facts obtained by pop clauses, and \( \Pi_n^A \) only uses push clauses of the form (3.20iv). For each fact we can use the same justification as in the Horn proof \( \Pi \). In addition, we can set \( C_n \) to the set of anonymous constants assigned to anonymous variables by the substitutions of the c-facts in \( \Pi_n^A \).

Similarly, \( \hat{\Pi}^{\text{pop}} \cdot \hat{\Pi}_1^A \cdot \hat{\Pi}_n^A - \hat{\Pi}_n^B \cdot \hat{\Pi}_n^B \) is a Horn proof of (3.24b) for all \( n \in \{0, \ldots, N\} \). The facts in \( \Pi^{\text{pop}} \) are clearly justified, and the choice for \( R_1, \ldots, R_n \) makes the facts in \( \hat{\Pi}_1^A \), \( \cdots \), \( \hat{\Pi}_n^A \) a-facts. Thus, we know that no fresh anonymous constants need to be assigned at all: In the Horn proofs \( \Pi_n^B \) no clause of the form (3.20iv) has been applied because no c-fact in \( \Pi_n^B \) is of the form \( r^{(n,0)}(t,k) \) (for any \( r, n, t, k \)).

The c-facts in \( \hat{\Pi}_n^B \) are derived by a clause \( \varphi \in \Phi_P \) which corresponds to a clause \( \varphi' \in \text{Basic}(\Phi_P) \), and so we can use this clause as justification. The only clauses weakened during the construction of the stage theory are the (generalized) push and send clauses of the form (3.20v), but we know by assumption that each key \( k \) occurring in \( \varphi' \) has to be known at the time \( \varphi \) is applied, so for the construction of a fact in stage \( e \), the fact \( \Pi^e(k) \) for a stage \( e_k < e \) has to occur in \( \Pi \). But then we know that \( \Pi(k) \) is an element of \( \hat{\Pi}_1^B \cdot \cdots \cdot \hat{\Pi}_n^B \) or an element of \( \Pi_n^B \) occurring prior to the c-fact constructed by \( \varphi' \).

This concludes the proof of the lemma. \( \square \)

**Remark 3.12.** From now on we will assume that there are only flat push clauses, so each generalized push clause that has a non-flat term on his left-hand-side is transformed according to the method from Section 2.4.3. By assumption all push and send clauses of a protocol have simple terms on the left-hand-side, so the transformation is possible because each non-flat generalized push clause has a linear left-hand-side. \( \triangleleft \)

### 3.4 The Graph of an Attack

We will now present a structure called *directed acyclic graph of the attack* (ADAG) introduced in [1true5a,1true5b], which represents an attack on a protocol: We show that such an ADAG for a protocol exists if and only if there is an attack on this protocol. As we later show a double exponential bound for ADAGs, this structure allows us to present the algorithm which decides security of recursive protocols. Again, we begin with preliminaries.

**Definition 3.13.** Let \( \Phi \) and \( \Psi \) be stage theories. The theory \( \Psi \) is an *instance* of \( \Phi \), if each clause in \( \Psi \) is an instance of a clause in \( \Phi \), more precisely if for each clause \( \psi \in \Psi \) there is a substitution \( \sigma: X \rightarrow T(\Sigma, X \cup Y) \) and a clause \( \varphi \in \Phi \) such that \( \hat{\sigma}(\varphi) = \psi \) where \( \hat{\sigma} \) is the extension of \( \sigma \) on clauses. \( \triangleleft \)
Term DAGs

**Definition 3.14.** Let $\Sigma$ be a signature. Then $D = (V, F, \mu)$ consisting of a finite set of nodes $V$, a set of edges $F \subseteq V \times V \times \mathbb{N}$ with an order in last component, and a labeling function $\mu : V \to \Sigma$ is a term DAG over $\Sigma$ if the following holds: For a node $v \in V$ with $\mu(v) = f$ for a function symbol $f \in \Sigma_m$, the node $v$ has $m$ ordered successors $v_1, \ldots, v_m$ in the DAG, i.e., $(v, v_i, j) \in F$ for all $i \in \{1, \ldots, m\}$.

We will write $v \equiv f(v_1, \ldots, v_m)$ if $D$ is clear from the context.

Embeddings

**Definition 3.15.** Let $T \subseteq T(\Sigma \cup \Gamma, X \cup Y)$ be a set of terms. Let $D = (V, F, \mu)$ be a term DAG over $\Sigma \cup \Gamma$. A function $\alpha : \text{sub}(T) \to V$ is a D-embedding for $T$ if for all $v \in V$ and all $y \in Y \cap \text{sub}(T)$ we have

\[
\begin{align*}
  v \equiv c & \in \Gamma & \text{if } v = \alpha(y), \quad (3.25 \text{a}) \\
  v \equiv f(v_1, \ldots, v_m) & \text{and } \alpha(t_j) = v_j \text{ for all } j \in \{1, \ldots, m\} & \text{if } v = \alpha(f(t_1, \ldots, t_m)). \quad (3.25 \text{b})
\end{align*}
\]

Two embeddings $\alpha_1$ and $\alpha_2$ are compatible if we have $\alpha_1(x) = \alpha_2(x)$ for each regular and anonymous variable $x \in \text{dom}(\alpha_1) \cap \text{dom}(\alpha_2) \cap (X \cup Y)$.

For $v \in V$ and $t \in T$ we further write $t \mapsto v$ if there exists a unique embedding $\alpha$ for $\{t\}$ which is determined by $\alpha(t) = v$. We write $(t, t') \mapsto (v, v')$ if both $t \mapsto v$ and $t' \mapsto v'$ hold true and both embeddings are compatible.

Registers

**Definition 3.16.** Let $\kappa$ be a variable register and $\kappa'$ be a ground register. We denote the substitution which maps each variable $y \in \kappa$ to the corresponding anonymous constant $c \in \kappa'$ by $\text{subst}(\kappa, \kappa')$ if such a substitution is well-defined; i.e., if for each $y \in \kappa$ we have exactly one $c \in \kappa'$:

\[
\text{subst}(\kappa, \kappa') = \{ y \mapsto c \mid \text{there exists } z \in \{1, \ldots, Z\} \text{ such that } \kappa[z] = y \in Y \text{ and } \kappa'[z] = c \in \Gamma \}. \quad (3.26)
\]

Again, we will write $(\kappa_1, \kappa_2) \mapsto (\kappa'_1, \kappa'_2)$ if both substitutions $\sigma_1 = \text{subst}(\kappa_1, \kappa'_1)$ and $\sigma_2 = \text{subst}(\kappa_2, \kappa'_2)$ exists and are compatible, i.e., if for each $y \in \text{dom}(\sigma_1) \cap \text{dom}(\sigma_2)$ we have $\sigma_1(y) = \sigma_2(y)$. \quad (3.27)

For example, the application of a push clause $[\ldots r_1(t_1, \kappa_1) \mapsto r_2(t_2, \kappa_2)]$ to a fact $r_1(t'_1, \kappa'_1)$ will result in a fact $r_2(t'_2, \kappa'_2)$ where we have $(\kappa_1, \kappa_2) \mapsto (\kappa'_1, \kappa'_2)$.

We will now define ADAGs and after the formal definition we will give some intuition of how ADAGs resemble terms, predicate symbols and so on.
ADAGs

**Definition 3.17.** Let $\mathcal{P} = (P, \Phi)$ be a protocol, let $\pi = [t_n \to r_n(s_n)]_{n=1}^N$ be an execution scheme and $f$ be a stage mapping for $\pi$ for the set $E$ of stages.

Let $Q_f$ be the set of predicate symbols occurring in $\Phi_f$ and let $K_P$ be the keys used in the protocol. By $T_P$ we denote the set of **static terms** occurring in $P$, more precisely:

$$T_P = \{c; s\} \cup \{t_n, s_n \mid n \in \{1, \ldots, N\}\} \cup K_P . \quad (3.27)$$

Let $\mathcal{D}$ be the tuple $(D, \Psi, \bar{\Gamma}, \alpha, \beta, \gamma, \delta, \lambda)$ with

- a finite term $\text{DAG} D = (V, F, \mu)$ over $\Sigma \cup \bar{\Gamma}$,
- a stage theory $\Psi$ which is an instance of the stage theory $\Phi_f$,
- a finite set of anonymous constants $\bar{\Gamma}$,
- an embedding function $\alpha: \text{sub}(T_P) \to V$, which embeds $T_P$ in $D$,
- a witness function $\beta: V \times Q_f \times \mathbb{N} \to V \times \mathbb{N} \times \Psi$,
- a register function $\gamma: V \times Q_f \times \mathbb{N} \to \bar{\Gamma}^Z$,
- a predicate function $\delta: V \to 2^{Q_f}$, and
- a freshness function $\lambda: \bar{\Gamma} \to V \times Q_f \times \mathbb{N} \times \{1, \ldots, Z\}$.

Then $\mathcal{D}$ is called $\text{DAG of the attack (ADAG)}$ for $\mathcal{P}$ and $(\pi, f)$ if the following three conditions hold (see the explanation below for additional notation we use):
(i) For all \( n \in \{1, \ldots, N\} \) and all \( k \in \text{dom}(f) \) we have
\[
\begin{align*}
\gamma^e(\alpha(t_n)) & \text{ for a stage } e \ll n, \\
\gamma^e(\alpha(k)) & \text{ for a stage } e < f(k), \text{ and} \\
\gamma^e(\alpha(\$)) & \text{ for a stage } e. 
\end{align*}
\]

(ii) For each node \( v \) we have
\[
\gamma^v(\delta(v)) \text{ for at most one } e \in E. 
\]

(iii) For each node \( v \) and each predicate symbol \( p \in \gamma^v(v) \) at least one of the following conditions holds, i.e., \( p \) is justified because of

(iii a) the application of a protocol step (or the intruder's initial knowledge)
\[
v = \alpha(c) \text{ with } p = \gamma^{(0,0)} \\
or \quad v = \alpha(s_n) \text{ with } p = \gamma^{(n,0)} \text{ for some } n \in \{1, \ldots, N\},
\]

\[
\begin{align*}
\gamma^*(v, p) & = \{0\}, \\
\gamma(v, p, 0, z) & \neq \gamma(v, p, 0, z') \text{ for } z \neq z', \text{ and} \\
\lambda(\gamma(v, p, 0, z)) & = (v, p, 0, z) \text{ for all } z \in \{1, \ldots, Z\};
\end{align*}
\]

(iii b) the application of an (intruder) pop clause with \( p \in Q \cup \tilde{I} \)
\[
v = f(v_1, \ldots, v_m) \quad \text{and} \\
\begin{array}{c}
\text{there is a clause } \psi = [p_1(x_1), \ldots, p_m(x_m) \rightarrow p(f(x_1, \ldots, x_m))] \in \Psi \\
\text{with } p_j \in \delta(v_j) \text{ for all } j \in \{1, \ldots, m\};
\end{array}
\]

(iii c) the application of an (intruder) push clause with \( p \in \tilde{R} \) witnessed by \( \beta \)
\[
\begin{align*}
\gamma^*(v, p) & \neq \emptyset, \text{ and for all } i \in \gamma^*(v, p) \\
\text{we have } v' & = f(v_1, \ldots, v_m) \\
\text{with } v & = v_j \text{ for some } j \in \{1, \ldots, m\},
\end{align*}
\]

such that \( \beta(v, p, i) = (v', p_m, i', \psi), \)
\[
\begin{align*}
\text{with } \psi & = [p_1(t_1'), \ldots, p_m(t_m', \kappa') \rightarrow p(x_j, \kappa)] \in \Psi, \\
t' & = f(x_1, \ldots, x_m), \\
p_j & \in \delta(v') \text{ for all } j \in \{1, \ldots, m\}, \text{ and} \\
(k, \kappa') & \mapsto (\gamma(v, p, i), \gamma(v', p_m, i')), \text{ as well as} \\
\lambda(\gamma(v, p, i, z)) & = (v, p, i, z') \text{ for some } z' \text{ with } \kappa[z] = \kappa[z'] \\
\text{for all } z \text{ with } \kappa[z] \in \gamma^*;
\end{align*}
\]
(iii d) or the application of a send clause with \( p \in \Gamma \) witnessed by \( \beta \)

\[
\beta(v, p, 0) = (v', p_m, l', \psi)
\]

(ADAG iii d)

with \( \psi = [p_1(t'), \ldots, p_m(t', k') \rightarrow p(t)] \in \Psi \), and

\[
p_j \in \delta(v') \quad \text{for all} \quad j \in \{1, \ldots, m\},
\]

\[
\delta_\chi(t', l') \mapsto (v, v') \quad \text{for} \quad \sigma_\chi = \text{subst}(k', \gamma(v', p_m, l')).
\]

Remark 3.18. For a node \( v \) and \( p \in \delta(v) \cap \overline{K} \) we define \( \gamma^+(v, p) \) to contain what we call the *indices* of the registers, i.e.,

\[
\gamma^+(v, p) = \{ i \in \mathbb{N} \mid \gamma(v, p, i) \text{ is defined} \}.
\]  
(3.28)

By abuse of notation, we will use \( \beta \) as a function \( \beta : V \times Q_f \times \mathbb{N} \rightarrow V \times Q_f \times \mathbb{N} \times \Psi \), i.e., mapping a node \( v \), a symbol \( p \) and a register \( i \) not only to \((v', l', \phi)\), but to \((v', p', l', \phi)\) if \( p' \) is in \( \phi \), i.e., if we have \( \phi = [\ldots, p'(t', k') \rightarrow p(t)] \) or \( \phi = [\ldots, p'(t', k') \rightarrow p(t, k)]. \)

Similarly, we will write \( \gamma(v, p, i, z) \) instead of \( \gamma(v, p, i)[z] \), so we are using \( \gamma \) as a function \( \gamma : V \times Q_f \times \mathbb{N} \times \{1, \ldots, Z\} \rightarrow \overline{\Gamma} \).

We will now give some intuition what was defined here; keep in mind that we will later show that an ADAG for a protocol represents an attack on that protocol, so we will explain how an ADAG represents a Horn proof of (3.21).
First of all, take a look at Figure 3.4. An ADAG consist mainly of a term DAG $D$ representing all terms that occur in the run of a protocol. The nodes of the DAG are labeled with function symbols and constants, so for each node $v$ there is a unique term $t$ defined by the labeling of $v$ and its descendants, we will say $v$ corresponds to $t$. The function $\alpha$ embeds the static terms of the protocol to $D$.

In addition the nodes are labeled with predicate symbols by $\delta$: A node $v$ corresponding to a term $t$ is labeled with all predicate symbols $p$ for which there is $p(t)$ or $p(t,\kappa)$ in the Horn proof. For each node and each symbol there may be an unbounded set of registers $\kappa$, in the ADAG they are represented by $\gamma$. Moreover, the freshness of a generated anonymous constant in a register of a node is witnessed by $\lambda$ through the definition in (ADAG iii c).

The set of conditions guarantee the resemblance between ADAGs and Horn proofs of (3.21): Condition (ADAG i) ensures that the conclusion of the Horn proof is fulfilled by the ADAG, while condition (ADAG ii) corresponds to the normality of the Horn proof.

Furthermore, each fact in a Horn proof is either an a-fact or constructed by a clause. Accordingly, each predicate symbol in the ADAG needs a justification. This is ensured by condition (ADAG iii) as illustrated in Figure 3.5. The fact is either an assumed fact (ADAG iii a), or there is a pop clause (ADAG iii b), a push clause (ADAG iii c), or a send clause (ADAG iii d) that justifies the fact. For the last two cases this justification is witnessed by $\beta$.

We can now express the following theorem, linking the existence of an ADAG to the insecurity of a protocol:

**Theorem 3.19 (Characterization of Attacks by ADAGs).** Let $P = (P, \Phi)$ be a protocol.

1. If $(\pi, \sigma)$ is an attack on the protocol then there exists a stage mapping $f$ and an ADAG $(D, \Phi, \hat{\Gamma}, \alpha, \beta, \gamma, \delta, \lambda)$ for $P$ and $(\pi, f)$.
2. If there is an ADAG for $P$ for an execution scheme and a stage mapping $(\pi, f)$ then there exists an attack $(\pi, \sigma)$ on the protocol for a substitution $\sigma$.

### 3.4.1 Proof of Theorem 3.19, Part 1

**Proof.** Let $P = (P, \Phi)$ be a protocol, and let $(\pi, \sigma)$ be an attack on that protocol. By Lemma 3.11 we know that there exists a stage mapping $f$ and sets $C_0, C$ such that statement (3.21) holds; let $\Pi$ be a Horn proof of that. We will now construct a corresponding ADAG.

- Let $T = \{ \sigma(t) \mid t \in T_P \}$ be a set of ground terms. We define a term DAG $D = (V, F, \mu)$ over $\Sigma \cup \hat{\Gamma}$ (with $\hat{\Gamma}$ as defined below) and the function $\hat{\alpha}$ to be a $D$-embedding for $T$. We also define $a$, which shall be a $D$-embedding for $T_P$ compatible with $\hat{\alpha}$, i.e., $a(t) = \hat{\alpha}(\sigma(t))$ for each $t \in T_P$. 

Let \( \hat{D} \) be the set of all anonymous constants occurring in the Horn proof and the assumption, i.e., including \( C \) and \( C_0 \).

For all \( c \in C_0 \) there is a register \( k^{(n)}_z \) and some \( z \in \{1, \ldots, Z\} \) with \( c = \kappa^{(n)}_z \) such that \( r^{(n,i)}_n(s_n, k^{(n)}_z) \) appears in the assumption of the Horn proof. Now set \( \gamma(a(s_n), r^{(n,i)}_n, 0) = \kappa^{(n)}_z \) and \( \lambda(c) = (a(s_n), r^{(n,i)}_n, 0, z) \).

Start with \( \delta(v) = \emptyset \) for all \( v \in D \). For each fact \( p(t) \in \Pi \) with \( \hat{a}(t) = v \) we put \( p \in \delta(v) \).

For each push symbol \( p \) and each term \( t \) for which there is a fact \( p(t, \kappa) \) in \( \Pi \), let \( \{\kappa_1, \kappa_2, \ldots, \kappa_l\} \) be the set of registers with \( p(t, \kappa_i) \in \Pi \) for each \( i \in \{1, \ldots, l\} \). Assume that \( \hat{a}(t) = v \). Then we put \( p \in \delta(v) \), and in addition we set \( \gamma(v, p, \kappa) = \kappa_i \) for each \( i \in \{1, \ldots, l\} \).

If \( p(\hat{t}) \) is a c-fact in \( \Pi \) constructed by a send clause \( \varphi = [\ldots, p'(t', \kappa') \rightarrow p(l)] \in \Phi_f \) and a substitution \( \sigma \), let \( v = \hat{a}(\hat{t}) \) and \( v' = \hat{a}(\sigma(t')) \). Let \( i' \) be the index such that \( \gamma(v', p, \kappa') = \sigma(\kappa') \). Then set \( \beta(v, p, 0) = (v', i', \varphi) \).

If \( p(\hat{t}, \hat{k}) \) is a c-fact in \( \Pi \) constructed by a push clause \( \varphi = [\ldots, p'(t', \kappa') \rightarrow p(t, \kappa)] \in \Phi_f \) and a substitution \( \sigma \), let \( v = \hat{a}(\hat{t}) \) and \( v' = \hat{a}(\sigma(t')) \). Then for let \( i \) be the index such that \( \gamma(v, p, i) = \kappa \), and let \( i' \) be the index with \( \gamma(v', p, i') = \kappa' \). Then set \( \beta(v, p, i) = (v', i', \varphi) \), and we also set \( \lambda(\sigma(y^*)) = (v, p, i, z) \) for each \( y^* \in \text{var}_\gamma(\kappa) \) and for one \( z \in \{1, \ldots, Z\} \) with \( y^* = \kappa_z \).

Now \( \mathcal{D} = (D, \Phi, \hat{D}, a, \beta, \gamma, \delta, \lambda) \) is an ADAG for \( \mathcal{P} \) and \( (\pi, f) \):

First we need to check if \( \mathcal{D} \) is well-defined, i.e., if all elements of \( \mathcal{D} \) are of the right type. Most of this directly holds by definition, the interesting thing is to check that \( \lambda \) actually is a functions, meaning there for each \( c \in \hat{D} \) there is at most one \( (v, p, i, z) \). This is ensured by the definition of Horn proofs: In \( \Pi \) each \( c \in C \subseteq \hat{D} \) is assigned only once by one substitution to a fresh variable \( y^* \in Y^* \), possibly multiple times in one register. In addition, the anonymous constants \( c \in C_0 \subseteq \hat{D} \) are used in the assumption of the Horn proof \( \Pi \), but they are not assigned anywhere else in the protocol.

We can now check the properties \([\text{ADAG}_{\mathcal{D}}]a\) to \([\text{ADAG}_{\mathcal{D}}]c\): Condition \([\text{ADAG}_{\mathcal{D}}]a\) is ensured by the Horn proof \( \Pi \), condition \([\text{ADAG}_{\mathcal{D}}]b\) holds because of the normality of the proof, for both conditions we refer to Lemma 3.1.1.

For the third condition we look at the reason for each node’s creation: For a node \( v \in D \) with a predicate symbol \( p \in \delta(v) \) we have a term \( \hat{t} \in T \) with \( v = \hat{a}(\hat{t}) \). We now take the fact \( a \in \Pi \) which contains \( \hat{t} \), i.e., which has the form \( p(\hat{t}) \) or \( p(\hat{t}, \hat{k}) \). If this is an a-fact, we know that \([\text{ADAG}_{\mathcal{D}}]b\) holds. If this is a c-fact, we have a clause \( \varphi \) constructing \( a \). If \( \varphi \) is an (intruder) pop clause, case \([\text{ADAG}_{\mathcal{D}}]c\) holds.

If \( a \) was constructed by a push clause \( \varphi = [p_1(t'), \ldots, p_m(t', \kappa') \rightarrow p(t, \kappa)] \in \Phi_f \) with the substitution \( \sigma \) (this means \( \sigma(t) = \hat{t} \)), we know \([\text{ADAG}_{\mathcal{D}}]c\) holds: We have an index \( i \) with \( \gamma(v, p, i) = \sigma(\kappa) \). There is a fact \( p_m(t', \kappa') \in \Pi \) with \( \hat{t}' = \sigma(t') \), which leads to a corresponding node \( v' = \hat{a}(\hat{t}') \). As the definition of a Horn proof holds for \( \Pi \), we put the
predicate symbols \( p_1, \ldots, p_m \) in \( \delta(v') \). Because of the application of \( \varphi \) we have \((\kappa, \kappa') \mapsto (\hat{\kappa}, \kappa')\), where we know by definition of \( \gamma \) that we have \((\hat{\kappa}, \kappa') = (\gamma(v, p, i), \gamma(v', p_m, i'))\).

Finally, for each \( y^* \in \text{var}^* (\kappa) \) we defined \( \lambda \) to point to \((v, p, i, z)\) for some \( z \) with \( \kappa[z] = y^* \).

If \( a \) was constructed by a send clause \( \varphi = [p_1(t'), \ldots, p_m(t', \kappa') \mapsto p(t)] \in \Phi \) with the substitution \( \sigma \) (this again means \( \sigma(t) = \hat{t} \)), we know that \( (\text{ADAG III}) \) holds with a similar argumentation: There is a fact \( p_m(t', \kappa') \in \Pi \) with \( \hat{t}' = \sigma(t') \), which again leads to the node \( v' = \hat{v}(t') \). The predicate symbols \( p_1, \ldots, p_m \) are elements of \( \delta(v') \). The interesting part here is the compatibility of the embeddings:

As \( t' \) and \( t \) are elements of \( T_p \), we know by the definition of the embedding \( \hat{a} \) that \((t, t') \mapsto (v, v')\), as for each variable in \( x \in \text{var}(t) \cap \text{var}(t') \) we know that \( \hat{a} (\sigma(x)) \) points to exactly one node. However, that is not enough – we have to show that each node corresponding to an anonymous variable \( y \in \text{sub}(t) \) is labeled with the correct anonymous constant.

In \( \varphi \), each anonymous variable \( y \in \text{sub}(t) \) has to occur in the register \( \kappa' \), so in the Horn proof \( \Pi \) the anonymous constant \( c = \hat{\sigma}(y) \) has to occur in \( \kappa' \). By construction of the DAG this gives us \( c \in \gamma(v', p_m, i') \) for an index \( i' \); so we know \( v \equiv \sigma_{v'}(t) \) for \( \sigma_{v'} = \text{sub}(\kappa', \gamma(v', p_m, i')) \).

As there are no anonymous variables in \( t' \), we can substitute each anonymous variable \( y \in \text{sub}(t) \) with the corresponding anonymous constant occurring in \( \kappa' \), without affecting the compatibility of the embeddings. Thus, we know \( (\sigma_{v'}(t), t') \mapsto (v, v') \).

This proves that \( \Pi \) is an ADAG for \( \mathbb{P} \) and \((\pi, f)\). We can now address the other direction of the proof.

### 3.4.2 Proof of Theorem 3.19, Part 2

We now assume that there is an ADAG \( \Pi = (D, \Phi, \hat{\Gamma}, \alpha, \beta, \gamma, \delta, \lambda) \) for a protocol \( \mathbb{P} \) and a pair \((\pi, f)\) of an execution scheme \( \pi = [t_n \rightarrow r_n(s_n)]_{n=1}^N \) and a stage mapping \( f \).

We have to construct a substitution \( \sigma \) such that \((\pi, \sigma)\) is an attack. We use Lemma 3.11 and can thus construct a Horn proof \( \Pi \) of statement 3.21. For each \( x \in T_p \cap X \) we set \( \sigma'(x) = t \) for the unique term \( t \) with \( a(x) \equiv t \).

1. Now start with \( \Pi' \) being the empty sequence.
2. Put all the facts of the form \( q(t) \in \Pi' \) where we have \( q \in Q \) in the set \( \delta(v) \) for a node \( v \equiv t \); putting \( q(t_1) \) before \( q(t_2) \) if \( t_1 \in \text{sub}(t_2) \).
3. Then put all the facts of the form \( p(t, \kappa) \in \Pi' \) where we have \( p \in \hat{R} \) in the set \( \delta(v) \) for a node \( v \equiv t \), and where \( \kappa = \gamma(v, r, i) \) for an index \( i \). We put \( p(t_1, \kappa_1) \) before \( p(t_2, \kappa_2) \) if we have \( t_2 \in \text{sub}(t_1) \).
4. Finally we put all the facts of the form \( I^e(t) \in \Pi' \) where we have \( I^e \in \delta(v) \) for a stage \( e \in E \) and a node \( v \equiv t \); putting \( I^e(t_1) \) before \( I^e(t_2) \) if \( t_1 \in \text{sub}(t_2) \).
Now \( \Pi \) is a Horn proof for statement (3.24), more precisely: \( \Pi \) is a normal Horn proof of the following statement with \( t_{N+1} = \$ \) for stages \( e_n \ll n + 1 \) and \( e_k < f(k) \), and for a set of pairwise disjoint registers \( \kappa(n) \) with \( C \) being disjoint with each \( \kappa(n) \):

\[
\{ \Gamma^{(0)}(c) \} \cup \{ \Gamma^{(n,0)}(\sigma(s_n), \kappa(n)) \mid n \in \{1, \ldots, N\} \} \vdash \Phi_f \\
\{ \Gamma^{n}(\sigma(t_{n+1})) \mid n \in \{0, \ldots, N\} \} \cup \{ \Gamma^{k}(k) \mid k \in \text{dom}(f) \} \quad (3.29)
\]

Let \( A \) be the assumption and \( B \) be the conclusion of this statement. We know that \( \Pi \) is a normal Horn proof because of (ADAG ii). The assumption \( A \) corresponds to (ADAG iii a). By (ADAG ia), (ADAG ib), and (ADAG ic), we also know \( B \subseteq \Pi \) for stages \( e_n \ll i + 1 \) and \( e_k < f(k) \).

Hence, we only have to show that the Horn proof \( \Pi \) is conclusive, i.e., that each fact in \( \Pi \) is a valid a- or c-fact.

Let \( a = p(\bar{i}) \) or \( a = p(\bar{i}, \bar{k}) \) be a fact in \( \Pi \). We know by construction that there is a node \( v \) with \( v \equiv t \) and \( p \in \delta(v) \). We can now use condition (ADAG iii) from the definition of an ADAG to distinguish the following cases as one of them has to apply for \( p, v \):

(iii a) In this case \( a \) is an a-fact, i.e., \( a \in A \).

(iii b) Now \( a \) is a c-fact constructed by an (intruder) pop clause. By construction we put \( p_j(\bar{i}_j) \in \Pi \) before \( a \) for all \( j \in \{1, \ldots, m\} \) and for \( v_j \equiv t_j \), because \( \bar{i}_j \in \text{sub}(\bar{i}) \) holds. By (ADAG iii b) we have a matching clause \( \psi \in \Psi \), which is an instance of a clause \( \varphi \in \Phi_f \). Hence, \( p(\bar{i}) \) can be derived in \( \Pi \) by that \( \varphi \).

(iii c) In this case \( a = p(\bar{i}, \bar{k}) \) is a c-fact constructed by a push clause \( \varphi \) of which \( \psi \) is an instance (with \( \beta(v, p, i) = (v', i', \psi) \) for indices \( i, i' \)); let \( \sigma_\psi \) be the substitution with \( \sigma_\psi(\varphi) = \psi \). We put the prerequisites for that construction \( p_j(\bar{i}'') \) and \( p_m(\bar{i}', \bar{k}') \) in \( \Pi \) before \( a \), because we put the pop facts to the beginning and because \( \bar{i} \in \text{sub}(\bar{i}') \) holds.

In addition, we find a substitution \( \bar{\delta}_1 \) with \( \bar{\delta}_1(\bar{i}') = \bar{i}' \), and \( \bar{\delta}_1(\bar{i}) = \bar{i} \), and another substitution, \( \bar{\delta}_2 \), which embeds \( (\kappa', \kappa') \) to \( (\bar{k}, \bar{k}') \) = \( (\gamma(v, p, i), \gamma(v', p_m, i')) \).

These two substitutions can be chosen disjoint as \( \text{var}(\bar{i}') \cap Y = \emptyset \), so we unify them to \( \bar{\delta}_3 = \bar{\delta}_1 \cup \bar{\delta}_2 \) and set \( \bar{\delta} = \bar{\delta}_3 \circ \sigma_\psi \).

Now \( a \) is constructed from \( p_j(\bar{i}'') \) and \( p_m(\bar{i}', \bar{k}') \) with the clause \( \varphi \) and the substitution \( \bar{\delta} \).

(iii d) In the last case, \( a \) is a c-fact constructed by a send clause, and the same argumentation as in case (ADAG iii c) applies, with \( \sigma_2 \) being subst(\( \kappa', \gamma(v', p_m, i') \)).

This concludes our proof; we have shown that ADAGs and attacks on protocols are equivalent notions.
4 Scaling Down ADAGs

Theorem 3.19 characterizes the existence of an attack by a special structure with a set of local properties. The ADAG for an attack can now be guessed and checked by an algorithm if we find a bound for the size of the ADAG.

4.1 Preliminaries

To do so we will first define some notation and show certain properties of ADAGs that are important to keep in mind.

Let $P = (P, \Phi)$ be a protocol and let $D = (D, \Psi, \tilde{\alpha}, \beta, \gamma, \delta, \lambda)$ be an ADAG for $P$ and a $(\pi, f)$, with $V$ being the set of nodes in $D$ and $\pi = [t_n \to r_n(s_n)]_{n=1}^{N}$.

Let $E$ be a set of stages for $P$. Let $\tilde{R} = (R \cup \{r_I\})^{(E)}$ be the set of push symbols which may be used in $D$, and let $\tilde{I} = \{I\}^{(E)}$.

4.1.1 Notation

**Definition 4.1.** We define the following relations, sets and notations:

- For two nodes $v, v' \in V$ we will use the relation $v \to_D v'$ if there is an edge $(v, v', j)$ in $D$ for a $j \in \mathbb{N}$. Let $\to_D^*$ be the reflexive transitive closure of $\to_D$.

  For a set of nodes $W$ we will also write $W \to_D v$ if $w \to_D v$ holds for a node $w \in W$.

- For two nodes $v, v' \in V$ we define a second kind of edges, $\beta$-edges, in the following way: $(v, p, i) \leadsto_D (v', p', i)$ if conditions (ADAG iii c) or (ADAG iii d) hold for $v'$ and $p'$ such that $\beta(v', p', i) = (v, p, i, \phi)$. Let $\leadsto_D^*$ be the reflexive transitive closure of $\leadsto_D$.

  A matching sequence of $\beta$-edges $(v_1, p_1, i_1) \leadsto (v_2, p_2, i_2) \leadsto \cdots \leadsto (v_{m+1}, p_{m+1}, i_{m+1})$ is called a $\beta$-path for some $m \in \mathbb{N}$ called the length of the path. We will omit the third component or both the second and third component and just write $v \leadsto_D v'$ if we have $(v, p, i) \leadsto_D (v', p', i')$ for some $p, p', i, i'$.

  In addition, we will further simplify the $\beta$-notation: As condition (ADAG ii) guarantees that for each node $u$ there is at most one stage $e$ with $1^e \in \delta(u)$, we will use $\beta(u, \tilde{I})$ instead of $\beta(u, 1^e, 0)$ if such a stage $e$ exists, and $(v, p, i) \leadsto_D (u, 1)$, respectively.
A node \( v \) of \( D \) is static if it belongs to the static terms of the protocol like \( t_n \) and \( s_n \), i.e., if \( v = \alpha(t) \) for a term \( t \in \text{sub}(T_P) \). We also need the nodes reachable from static nodes in a bounded number of steps:

\[
\text{Static}_D = \{ v \in V \mid \alpha(t) = v \text{ for a term } t \in \text{sub}(T_P) \},
\]

\[
\text{Static}_D^+ = \left\{ w \in V \mid v \leadsto_D^+ w \text{ for a term } v \in \text{Static}_D \text{ with } m \leq \|P\|^2 \right\}.
\]

A node \( v \) belongs to step \( n \) if we have \( \alpha(t_n) \rightarrow^*_D v \) or \( \alpha(s_n) \rightarrow^*_D v \).

\[
\text{Steps}_D(\leq n) = \{ v \in V \mid \alpha(t) \rightarrow^*_D v \text{ for a term } t \in \{c, t_1, s_1, \ldots, t_n, s_n\} \}.
\]

A node \( v \) can belong to multiple steps if, for example, \( \alpha(t_n) \rightarrow^*_D, \alpha(s_{n'}) \rightarrow^*_D v \) holds.

A node \( u \in V \) is a goal if \( u \) is the top node of the resulting term of a send action, i.e., if \( \Gamma^e \in \delta(u) \) because of a send clause for a stage \( e \):

\[
\text{Goals}_D(n) = \{ u \in V \mid \text{there is an } m \text{ such that } \Gamma^{(n,m)} \in \delta(u) \text{ and } P \text{ holds for } u \text{ and } \Gamma^{(n,m)} \},
\]

\[
\text{Goals}_D = \bigcup_{n=1}^{N} \text{Goals}_D(n),
\]

\[
\text{Goals}_D(v) = \{ u \in \text{Goals}_D \mid v \sim^* (u, \Gamma^e, 0) \}.
\]

In that case \((n, m)\) is the stage of the goal, and \( n \) is the protocol step of the goal; we will also call \( u \) an \( n \)-goal.

For a goal \( u \) with \( \Gamma^e \in \delta(u) \) and \( \beta(u, \Gamma^e, 0) = (v, p, i, \varphi) \) with \( \varphi = [\ldots, p(t', \kappa) \rightarrow \Gamma^e(t)] \), we find an embedding \( \alpha_u \) which embeds \( \sigma_k(t, t') \) to \( (u, v) \) for the substitution \( \sigma_k = \text{sub}(x, \gamma(v, p, i)) \).

We then define the set of fixed nodes, i.e., nodes belonging to \( t \) and \( t' \), and the set of nodes of variables, i.e., nodes representing the regular variables in \( t \):

\[
\text{Fix}_D(u) = \{ w \in V \mid \alpha_u(s) = w \text{ for a term } s \in \text{sub}(\sigma_k(t)) \cup \text{sub}(\sigma_k(t')) \},
\]

\[
\text{Var}_D(u) = \{ w \in V \mid \alpha_u(x) = w \text{ for a normal variable } x \in \text{var}(t) \subseteq \text{var}(t') \}.
\]

The send clause \( \psi \in \Psi \) may either be a send clause occurring in the stage theory of the protocol, i.e., \( \psi = \phi \) for a clause \( \phi \in \Phi_T \), or it may be an instance of such a send clause \( \phi \in \Phi_T \) with \( \psi \neq \phi \). In both cases, let \( \varphi = [\ldots, p(t', \kappa) \rightarrow \Gamma^e(t)] \). Let \( \alpha'_u \) and \( \alpha''_u \) be two embeddings with \( \alpha'_u(u) = \tilde{t} \) and \( \alpha''_u(v) = \tilde{t}' \). We define

\[
\text{Var}_D^+(u) = \{ w \in V \mid \alpha'_u(x) = w \text{ or } \alpha''_u(x) = w \text{ for } x \in \text{var}_X(\tilde{t}) \subseteq \text{var}(\tilde{t}') \}.
\]

Note that we may have two nodes \( \alpha'_u(x) \neq \alpha''_u(x) \) for a normal variable \( x \in \text{var}_X(\tilde{t}) \), while there is only one node \( \alpha_u(x) \) for each \( x \in \text{var}_X(\tilde{t}) \). Also note that we have
Var_D^* (u) = Var_D(u) if ψ = φ is not an instance.

For all sets and relations defined here with an index D, we will omit the index if it is clear from the context which ADAG we refer to.

4.1.2 Properties of ADAGs

To understand the proofs in the following sections it is important to point out some general properties of the structure of an ADAG D. Therefore, we will present and prove the following four lemmas.

Lemma 4.2 (Structure of β-Paths). Let v ∈ V be a node, let p ∈ δ(V) ∩ (\(\overline{R} \cup \overline{I}\)) be a predicate symbol and i ∈ γ(v, p) be the index of a register of v and p. Let

\[ \overline{v} = (v_1, p_1, i_1) \leadsto (v_2, p_2, i_2) \leadsto \cdots \leadsto (v_m, p_m, i_m) \]  \hspace{1cm} (4.10)

be a β-path of maximal length through (v, p, i), i.e., a path with (v, p, i) = (v_j, p_j, i_j) for a j ∈ \{1, \ldots, m\}, and such that there is no (v', p', i') with (v', p', i') \leadsto (v_1, p_1, i_1) or (v_m, p_m, i_m) \leadsto (v', p', i').

Then we can assume the following:

1. The subpath (v_1, p_1, i_1) \leadsto^* (v_j, p_j, i_j) is uniquely determined.
2. (v_1, p_1, i_1) = (s_n, r_n^{(n,0)}, 0) for an n ∈ \{1, \ldots, N\},
3. p_1, \ldots, p_{m-1} ∈ \overline{R}, i.e., besides the last step, the path consists of push clause applications,
4. v_1 \to v_2 \to \cdots \to v_{m-1}, i.e., besides the last step, the β-path moves along the normal edges of the ADAG, and
5. (v_m, p_m, i_m) = (u, I^e, 0) for a goal u and a stage e, and
6. if we have Goals_D(v) = ∅ for a node v, we also know δ(v) ∩ \(\overline{R}\) = ∅.

In other words, each β-path of maximal length begins at a push symbol r_n in a node corresponding to a term s_n, then it follows the normal edges of the DAG and the push symbols, moving from a node to one of its children, and finally it «jumps» to a goal by applying a send clause.

In terms of Horn proofs this only reflects the fact that each fact containing a push symbol was justified either by assumption (and the only assumption was r_n(s_n, r_n^{(n)}) or through construction by a flat push clause, and the fact is only necessary if it is a prerequisite for a send symbol.

Proof. Let v_j, p_j, i_j and \(\overline{v}\) be as described in the lemma. We will prove each assumption made in the lemma.
1. The subpath \((v_1, p_1, i_1) \sim^* (v_j, p_j, i_j)\) is uniquely determined because we can construct it directly: We distinguish between two cases, depending on which condition \(v\) and \(p\) fulfill in \(D\): If \(\text{ADAG i.a}\) holds for \(v_j\) and \(p_j\), we know \(j = 1\) by definition of the relation \(\sim\). Otherwise, either condition \(\text{ADAG ii.c}\) or condition \(\text{ADAG ii.d}\) holds for \(v\) and \(p\), so because \(\beta\) is a function there is a unique triple \((v_{j-1}, p_{j-1}, i_{j-1})\) with \((v_{j-1}, p_{j-1}, i_{j-1}) \sim (v_j, p_j, i_j)\). Then, by applying this construction recursively on \((v_{j-1}, p_{j-1}, i_{j-1})\), we get a path as assumed. This recursion terminates as we have a finite \(\text{DAG}\).

2. The proof of the first condition also shows that condition \(\text{ADAG i.a}\) holds for \(v_1\) and \(p_1\), so we know \(v_1 = \alpha(s_u), p_1 = r_n^{(n,0)}\) and \(i_n = 0\) for an \(n \in \{1, \ldots, N\}\).

3. If we have \((v, p, i) \sim (v', p', i') \sim (v'', p'', i'')\), we know that condition \(\text{ADAG iii.d}\) cannot hold for \(v'\) and \(p'\) because \(p' \in \overline{\Sigma}\) cannot be used as justification for any \(p''\), i.e., we cannot have \(\beta(v'', p'', i'') = (v', p', i', \varphi)\) for any \(\varphi\). By definition of \(\sim\), condition \(\text{ADAG iii.c}\) holds for \(v'\) and \(p'\), and so we have \(p' \in \overline{R}\). Applying this argumentation on all nodes \(v \in \{v_1, \ldots, v_m\}\) and \(v'' \in \{v_3, \ldots, v_m\}\) proves the third condition.

4. Here we can use the same argumentation as for the previous condition – knowing that \(\text{ADAG iii.c}\) holds for \(v'\) and \(p'\) leads to \(v \rightarrow v'\).

5. The proof for this condition is the most technical one: There may be \(\text{ADAGs}\) where this condition does not hold, but then we can construct a very similar \(\text{DAG}\) which only has less registers and perhaps a smaller predicate symbol labeling, but for which this condition holds.

Take a look at \(v = v_j\), and assume that there is no goal \(u\) with \((v, p, i) \sim^* (u, l)\). We either have \(j \neq m\), then condition 3 we have just proven leads to \(p_j \in \overline{R}\), or we have \(j = m\), in which case we also have \(p_j \in \overline{R}\) because otherwise \(v_j\) would be a goal and we could select \(u = v_j\).

We will now remove the register which is indexed by \(i_j\) and all of its descendants on all \(\beta\)-paths by restricting \(\gamma\), \(\beta\), and \(\lambda\) in the following way:

For all nodes \(v' \in V\), predicate symbols \(p' \in \delta(v')\) and all indices \(i' \in \gamma^*(v', p')\) such that we do not have \((v, p, i) \sim^* (v', p', i')\), we set

\[
\gamma_1(v', p', i') = \gamma(v', p', i') \quad (4.11)
\]

\[
\beta_1(v', p', i') = \beta(v', p', i') \quad (4.12)
\]

\[
\lambda_1(c) = \lambda(c) \text{ if } \lambda(c) = (v', p', i', z) \text{ for } a z \in \{1, \ldots, Z\} \quad (4.13)
\]

Otherwise \(\gamma_1, \beta_1,\) and \(\lambda_1\) shall be undefined, i.e., \(i' \notin \gamma^*_1(v', p')\).

We may have created a structure where a predicate symbol \(\beta\) for a node \(v\) does not have any registers left, \(\gamma^*_1(v, \beta) = \emptyset\), which would contradict the definition of an \(\text{ADAG}\) in condition \(\text{ADAG iii.d}\). But then, this predicate symbol \(\beta\) can clearly not lead to any goals, so we can also remove it.
As \( \beta \) may lead to other push symbols, we do not only remove \( \beta \), but also all of its descendants on \( \beta \)-paths: We set

\[
\delta_2(v') = \{ p' \in \delta(v') \mid (\tilde{v}, \tilde{p}) \not\sim^* (v', p') \} .
\] (4.14)

Now for all nodes \( v' \in V \), predicate symbols \( p' \in \delta_2(v') \) and all indices \( i' \in \gamma_1(v', p') \) we set

\[
\begin{align*}
\gamma_2(v', p', i') &= \gamma_1(v', p', i') , \\
\beta_2(v', p', i') &= \beta_1(v', p', i') , \\
\lambda_2(c) &= \lambda_1(c) \text{ if } \lambda_1(c) = (v', p', i', z) \text{ for a } z \in \{1, \ldots, Z\} .
\end{align*}
\] (4.15) (4.16) (4.17)

We will show that the result \( \mathcal{D}' = (D, \Psi, \hat{\tilde{R}}, \alpha, \beta_2, \gamma_2, \delta_2, \lambda_2) \) is an ADAG. Then, by repeating this process for all indices \( i \) and all predicate symbols \( p \) we get an ADAG \( \mathcal{D}'' \) which fulfills the fifth condition of the lemma.

We will show that we actually constructed an ADAG: As we only restricted the domain of the functions, all the components of the ADAG are still well-defined. The ADAG conditions \( \text{(ADAG i a)} \) to \( \text{(ADAG ii)} \) do clearly still hold, as we modified no \( I \)-symbols. For each node \( v \) and each symbol \( p \in \delta(v) \), we know that one of the four conditions \( \text{(ADAG iii a)} \) to \( \text{(ADAG iii d)} \) holds in \( \mathcal{D} \). Then this condition will still hold in \( \mathcal{D}' \): The cases \( \text{(ADAG iii a)} \) and \( \text{(ADAG iii b)} \) were not modified.

In case \( \text{(ADAG iii c)} \) we know because of \( p \in \delta(v) \) that \( (v, p, i) \sim^* (u, I) \) for an index \( i \) and a goal \( u \); otherwise, we would have removed \( p \) from \( \delta(v) \). But then, we also know that \( (v', p', i') \sim^* (u, I) \) for \( \beta(v, p, i) = (\tilde{v}, \tilde{p}, \hat{\tilde{i}}, \phi) \), i.e., we did not remove \( \beta \) or \( \hat{\tilde{i}} \) from \( \tilde{v} \). Hence, case \( \text{(ADAG iii c)} \) still holds.

The last case, \( \text{(ADAG iii d)} \), holds analogously – if \( v \) is a goal with \( \beta(v, I) = (\tilde{v}, \tilde{p}, \hat{\tilde{i}}, \phi) \), we know that \( (\tilde{v}, \tilde{p}, \hat{\tilde{i}}) \sim^* (u, I) \), so we did not remove \( \hat{\tilde{p}} \) from \( \beta'(\tilde{v}) \) or \( \hat{\tilde{i}} \) from \( \gamma'(\tilde{v}, \tilde{p}) \). This proves that by this transformation we get an ADAG.

6. This is a direct consequence of the fifth condition easily seen by contraposition: If \( p \in \gamma(v) \cap \hat{\tilde{R}} \), then we have at least one index \( i \in \gamma^*(v, p) \) and can apply the lemma’s fifth condition on \( (v, p, i) \) and can apply Goals\( \mathcal{D}(v) \neq \emptyset \).

This concludes the proof of our lemma, which will be used in the following proofs. \( \square \)

**Lemma 4.3 (Restriction of Nodes).** For a protocol \( \mathcal{P} \) we can assume that an ADAG for \( \mathcal{P} \) contains only nodes \( w \) for which \( a(t) \rightarrow^* w \) holds for a term \( t \in T_\mathcal{P} \).

**Proof.** Let \( W \) be the set of nodes of an ADAG \( \mathcal{D} \) such that \( w \in W \) is no descendant of \( a(t) \) for any \( t \in T_\mathcal{P} \). Furthermore, let \( w \in W \) be a node without a parent – we can select such a node as any parent of \( w \) would also be in \( W \).

We can remove this node without damaging the ADAG: First, \( w \) is not necessary to fulfill the ADAG conditions \( \text{(ADAG i a)} \) to \( \text{(ADAG i c)} \). For \( \text{(ADAG ii)} \) and \( \text{(ADAG iii a)} \) we have nothing to
show. Then, it cannot be any \( v_j \) in condition (ADG iii b) as \( w \) has no parents. Furthermore, by the previous lemma we know that \( \delta(v) \) cannot contain any push symbols, as it would have to be descendant of a node \( \alpha(t_n) \) by the previous lemma. Thus, it cannot be used as a justification in the witness function of (ADG iii c) and (ADG iii d).

Hence, if we define \( \mathcal{I}' \) to be the same as \( \mathcal{I} \) just with removing \( w \) from the set of nodes (and possibly from the functions), all ADAG conditions do still hold for \( \mathcal{I}' \). We can repeat this procedure until we reach an ADAG \( \mathcal{I}'' \) for which we have \( W = \emptyset \).

**Lemma 4.4** (Nodes of Variables). Let \( u \) be an \( n \)-goal of an ADAG \( \mathcal{I} \). Then we have \( \text{Var}_\mathcal{I}(u) \subseteq \text{Steps}_\mathcal{I}(\leq n) \).

From the protocol point of view, this is plausible as it only states that the terms sent in a step \( n \) have to be known to the principals in that step or an earlier step than \( n \).

This lemma also states that if \( d \) is the maximal depth of a term occurring in a send clause in the stage theory \( \Psi \) of \( \mathcal{I} \), than each path of length greater than \( d \) from \( u \) or some of its descendants in the DAG will lead to a node in \( \text{Steps}_\mathcal{I}(\leq n) \).

**Proof.** Let \( u \) be an \( n \)-goal of an ADAG \( \mathcal{I} \), and let \( v \in \text{Var}_\mathcal{I}(u) \). Then by the embedding condition in (ADG iii d) we know that \( v \) is not only a descendant of \( u \), but also of a node \( w \) with \( \beta(u, l) = (w, r', i, \varphi) \). We also know \( r' \in \delta(w) \) for a push symbol \( r \in R \). By looking at all possible send clauses we know \( e \leq n \). Then Lemma 4.2 states there is a node \( \tilde{w} = \alpha(s_m) \) for some \( m \) with \( \tilde{w} \rightarrow^* w \). Again, we know \( m \leq e \). But then, we have \( m \leq n \), leading to \( v \in \text{Steps}_\mathcal{I}(\leq m) \subseteq \text{Steps}_\mathcal{I}(\leq n) \).

**Lemma 4.5.** Let \( v \) be a node of an ADAG \( \mathcal{I} \) with \( v \in \text{Steps}_\mathcal{I}(\leq n) \), but \( v \notin \text{Static}_\mathcal{I}^+ \). Then there is an index \( i \leq n \) with \( \alpha(t_i) \rightarrow^*_x v \).

**Proof.** Let \( v \) be a node of an ADAG \( \mathcal{I} \). Then because of \( v \in \text{Steps}_\mathcal{I}(\leq n) \) there is a term \( t \in \{c, t_1, s_1, \ldots, t_n, s_n\} \) with \( \alpha(t) \rightarrow^*_x v \). If \( t = c \), we would have \( \alpha(c) \rightarrow^0 v \), so \( v \in \text{Static}_\mathcal{I}^+ \), this case is not possible. If \( t = t_i \) for an \( i \leq n \), there is nothing to show.

The only case left to consider is \( t = s_i \) for an \( i \leq n \). Then there is a path in the DAG of the form \( \alpha(s_i) \rightarrow w_1 \rightarrow \cdots \rightarrow w_m \rightarrow v \). As \( v \notin \text{Static}_\mathcal{I}^+ \), this path is longer than the depth of \( s_i \), which means there is a node \( w_j \) on the path with \( w_j = \alpha(x) \) for a variable \( x \in \text{sub}(s_i) \). But then by definition of a protocol step we know that \( x \in \text{sub}(t_i) \) for an \( i' \leq i \), and thus, there is a path \( \alpha(t_{i'}) \rightarrow u_1 \rightarrow \cdots \rightarrow u_{i'} \rightarrow w_j \rightarrow w_{j+1} \rightarrow \cdots \rightarrow w_m \rightarrow v \).

This shows \( \alpha(t_{i'}) \rightarrow^*_x v \).
4.2 Constructing Simple ADAGs

We now define simple ADAGs. Simple means that a goal with predicate symbol $I^e$ has no descendants in Static$_D^*$ with a predicate symbol $I^{e'}$ for any $e \gg e'$. The main idea is that these predicates are not necessary:

If a goal $u$ with stage $e$ is descendent of a goal $u'$ with stage $e' \ll e$, then we cannot use the predicate symbol $I^e$ to show that $I^{e'}$ holds, as the intruder’s knowledge in stage $e$ cannot be used to show his knowledge in stage $e' \ll e$.

But there may be another ancestor of $u$ in the DAG which needs the predicate symbol $I^e$. Therefore, we just copy the node $u$ to the node $\hat{u}$, leaving the original node as a descendent of $v$, but deleting it’s symbol $I^e$, and making the new copy $\hat{u}$ (which is annotated with the symbol $I^e$) a descendent of all ancestors of $u$ that need the symbol $I^e$.

**Definition 4.6.** The ADAG $D$ is called simple if for every $v \notin \text{Static}_D^*$ that is a descendent of $u \in \text{Goals}_D(n)$, the node $v$ is not in $\text{Goals}_D(m)$ for any $m > n$. 

As we copy some nodes, there may be a situation where the definition of an ADAG no longer holds: Assume that we duplicate a node $v$ which is in the set $\text{Var}_D(u)$ of a goal $u$ (with $\beta(u, I) = (u', i', \varphi)$). Further assume that the copy, call it $\hat{v}$, will only be a descendant of $u$ (not $u'$), while $v$ is only a descendant of $u'$ (and not of $u$). Now the embedding condition in (ADAG iii) is violated, i.e., a variable in $I$ and $i'$ is no longer mapped to the same node $v$. We will repair this by using instances of send clauses, as later shown in the definition of $\beta_u$. For this we use the following definition:

**Definition 4.7.** Let $\Phi_f$ be a stage theory for a protocol with $N$ steps. Then $\hat{\Phi}_f$ is the same stage theory unified with all instances $q'$ of send clauses $q \in \Phi_f$, where the depth of $q'$ is not greater than $|\Phi| \cdot (N + 1)$. 

It is easy to see that for a protocol $P = (P, \Phi)$ the size of such an instantiated stage theory $\hat{\Phi}_f$ is exponential in terms of the size of $P$, as the maximal depth equals $|\Phi| \cdot (N + 1) < |P|^2$, i.e., is polynomial in terms of the protocol size.

**Theorem 4.8 (Simple ADAGs).** Let $P$ be a protocol. If $D = (D, \Phi_f, \hat{\Gamma}, \alpha, \beta, \gamma, \delta, \lambda)$ is an ADAG for a protocol $P = (P, \Phi)$ and $(\pi, f)$, then there exists a simple ADAG $D'$ for $P$ and $(\pi, f)$ with $D' = (D', \hat{\Phi}_f, \hat{\Gamma}, \alpha', \beta', \gamma', \delta', \lambda')$. 

### 4.2.1 Proof of Theorem 4.8

**Proof.** In order to prove this theorem, we will construct a simple ADAG $D_N$ from an ADAG $D_0$ step-by-step: We will construct a sequence $D_0, \ldots, D_N$ of ADAGs where the following condition will hold for each ADAGs $D_n$ in the sequence: For every $v \notin \text{Static}_D^n$ that is a
descendent of \( u \in \text{Goals}_{D_n}(m) \) for an \( m \leq n \), the node \( v \) is not in \( \text{Goals}_{D_n}(m') \) for any \( m' > m \). In short:

\[
\text{For all } v \not\in \text{Static}_{D_n}\
\text{with } \text{Goals}_{D_n}(m) \rightarrow^* v \text{ for an } m \leq n
\]

we have \( v \not\in \text{Goals}_{D_n}(>m) \).

We will later show that this condition holds for \( n = N \), so we know that \( D_N \) is simple. We add another two conditions that will ease the argumentation:

Let \( u \in \text{Goals}_{D_n}(m) \) for an \( m > n \). For all \( v \in \text{Var}_{D_n}(u) \) we either have \( v \in \text{Var}_{D_n}(u) \), i.e., the variable that \( v \) corresponds to was not instantiated, or we have \( v \in \text{Steps}_{D_n}(\leq n) \). In short:

\[
\text{For all } u \in \text{Goals}_{D_n}(>n)\
\text{for all } v \in \text{Var}_{D_n}(u) \setminus \text{Var}_{D_n}(u),
\]

we have \( v \in \text{Steps}_{D_n}(\leq n) \).

And finally we need to state that the depth of send clauses used in the witness function \( \beta_n \) is bounded:

\[
\text{For all } \varphi = [\ldots, p'(t', \kappa) \rightarrow p(t) ] \in \tilde{\Phi}_f \\
\text{with } \beta_n(v, p, i) = (v', i', \varphi) \text{ for some } v, v', p, i, i'
\]

we have depth\((t), \text{depth}(t') \leq |P| \cdot (n + 1)\).

For \( D_0 \) all three conditions hold; the first one because there is no \( m \)-goal for \( m \leq 0 \); the second one because \( D_0 \) uses no instances, so \( \text{Var}_{D_n}(u) = \text{Var}_{D_n}(u) \) for all goals \( u \); and the last one, too, because \( D_0 \) uses no instances, so the depth of terms in send clauses is smaller than \( |P| \).

Now let \( n \in \{1, \ldots, N-1\} \). Let \( D_{n-1} = (D_{n-1}, \tilde{\Phi}_f, \tilde{\Gamma}_n, \beta_{n-1}, \gamma_{n-1}, \delta_{n-1}, \lambda_{n-1}) \) be an ADAG for \( P \) and \( (\pi, f) \), with \( V_{n-1} \) being the set of nodes in \( D_{n-1} \). Assume that the inductive hypothesis - i.e., conditions (4.19a) to (4.19c) - holds for \( D_{n-1} \). We will now define \( D_n = (D_n, \tilde{\Phi}_f, \tilde{\Gamma}_n, \beta_n, \gamma_n, \delta_n, \lambda_n) \), with \( V_n \) being the set of nodes in \( D_n \). We will then show that all three conditions also hold for \( D_n \).

As explained above, the main step in this construction is to duplicate some nodes in \( V_{n-1} \). We define the following set \( W \subseteq V_{n-1} \) of nodes that will be duplicated. These are all nodes that are only belonging to protocol steps greater than \( n \), and that are a descendent of an \( n \)-goal, but are no \( n \)-goal itself:

\[
W = \{ w \in V_{n-1} | \text{Goals}_{D_{n-1}}(n) \rightarrow^* w, \text{ for all } w \not\in \text{Goals}_{D_{n-1}}(n), \text{ and } w \not\in \text{Steps}_{D_{n-1}}(\leq n) \}.
\]
There may be some static nodes belonging to \( W \), and we do not want to manipulate them, hence, we define the set \( W_S \subseteq W \) of static nodes and their descendants. Formally, let \( W_S \) be the smallest set for which the following fixed point equation holds:

\[
W_S = \{ w \in W \mid w \in \text{Static}_{D_{n-1}} \text{ or } W_S \rightarrow w \}.
\] 

(4.21)

We know \( W_S \subseteq \text{Static}^+_{D_{n-1}} \): Each node \( w \in W_S \) is either in \( \text{Static}_{D_{n-1}} \), or it is a descendant of a node \( \hat{w} \in \text{Static}_{D_{n-1}} \). By \( W_S \subseteq W \) we know that in the latter case, \( \hat{w} \) and \( w \) are the descendants of an \( n \)-goal \( u \). But on the other hand both \( \hat{w} \) and \( w \) are not in \( \text{Steps}_{D_{n-1}}(\leq n) \) (as they are in \( W \)). By Lemma [4.4] we know that the length of the path \( \hat{w} \rightarrow^* w \) is bounded by \( |\Phi|^2 \) as that is the maximal depth of a send clause in \( \Phi_f \). Hence, we know that \( w \) is in \( \text{Static}^+_{D_{n-1}} \). In fact, the bound \( |\Phi|^2 \) in the definition of \( \text{Static}^+_{D_{n-1}} \) originates in this step.

Now we are able to construct the ADA DAG \( D_n = (D_n, \hat{\Phi}_f, \bar{F}, \alpha, \beta, \gamma, \delta, \lambda_n) \), i.e., we will define the components of \( D_n \).

The Term DAG \( D_n \)

We assume that we have \( D_{n-1} = (V_{n-1}, F_{n-1}, \mu_{n-1}) \), and we will construct the term DAG \( D_n = (V_n, F_n, \mu_n) \).

Let \( W^> \) be a set of new nodes, i.e., \( W^> \cap V_{n-1} = \emptyset \) with a bijection \( f : W \rightarrow W^> \); then set \( V_n = V_{n-1} \cup W^> \). Now each node \( w \in W \subseteq V_{n-1} \) is split into two nodes \( w \) and \( f(w) \in V_n \).

As we need the corresponding nodes, we define two functions:

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

\[
\triangleright : V_{n-1} \rightarrow V_n \\
\triangleright : V_n \rightarrow V_{n-1}
\]

Now we define the set of edges \( F_n \) by the following conditions for each \((v, w, l) \in F_{n-1}\):

\[
(v, w^>, l) \in F_n \text{ if } v \in W, \text{ if } v \notin W, \text{ otherwise. (4.24 a)}
\]

and either \((v, w^>, l) \in F_n \text{ if } v \notin W, w \in W, v \notin \text{Goals}_{D_{n-1}}(\leq n) \text{ (4.24 b)}
\]

or \((v, w, l) \in F_n \text{ otherwise. (4.24 c)}
\]

The labeling function \( \mu_n \) is defined by \( \mu_n(v) = \mu_{n-1}(v^>) \). Note that we did not modify the nodes, edges or labeling in \( \text{Steps}_{D_{n-1}}(\leq n) \), so we have \( \text{Steps}_{D_{n-1}}(\leq n) = \text{Steps}_{D_n}(\leq n) \).

The Register Function \( \gamma_n \) and the Freshness Function \( \lambda_n \)

The registers assigned to a node and a push symbol will just follow the paths on which they were constructed. Hence, take a node \( v \in V_{n-1} \) and a push symbol \( r \in \delta_{n-1}(v) \cap \bar{R} \). Now for each \( i \in \gamma_{n-1}^*(v, r) \) we set define the node \((v, r, i)^\triangleright \in V_n \) to be either \( v \) or \( v^> \).
We will set \((v, r, i)^\circ = v^\circ\) if there are nodes \(\varrho, w \in V_{n-1}\) with \(\varrho \sim_{\mathcal{D}_{n-1}} w \sim_{\mathcal{D}_{n-1}} (v, r, i)\) and with \(\varrho \not\in W\), but \(w \in W\) such that (4.24b) holds for \(\varrho\) and \(w\), i.e., we have \(\varrho \to_{\mathcal{D}_n} w^\circ\). Otherwise we set \((v, r, i)^\circ = v\).

Now for each \(v \in V_{n-1}\), each \(r \in \delta_{n-1}(v) \cap \tilde{R}\) and each \(i \in \gamma_{n-1}^*(v, r)\) we define:

\[
\gamma_n((v, r, i)^\circ, r, i) = \gamma_{n-1}(v, r, i) .
\]

(4.25)

Accordingly, for each \(e \in \tilde{V}\) we set:

\[
\lambda_n(e) = ((v, r, i)^\circ, r, i, z) \quad \text{for } \lambda_{n-1}(e) = (v, r, i, z) .
\]

(4.26)

**The Predicate Symbol Labeling Function \(\delta_n\)**

Let \(v \in V_{n-1}\) be a node. Let \(\delta_Q = \delta_{n-1}(v) \cap Q\), as well as \(\delta_R = \delta_{n-1}(v) \cap \tilde{R}\), and \(\delta_I = \delta_{n-1}(v) \cap \tilde{I}\). Then we have \(\delta_{n-1}(v) = \delta_Q \cup \delta_R \cup \delta_I\). Now we have to assign these symbols to \(v\) and \(v^\circ\) if the latter exists; so we will have \(\delta_{n-1}(v) = \delta_n(v) \cup \delta_n(v^\circ)\).

Hence, for now assume that \(v \in W\). The pop symbols can just be copied to both \(v\) and \(v^\circ\).

The I-symbols we want to remove from \(v\) are all the symbols \(I^e\) with a stage that is greater than \(n\), so we define \(\delta_I^{3^e} = \{ I^e \in \delta_I \mid e \gg n \}\). Finally we define the sets \(\delta_R(v), \delta_R(v^\circ) \subseteq \delta_R\) in the following way:

\[
\delta_R(v) = \{ r \in \delta_R \mid (v, r, i)^\circ = v \quad \text{for an index } i \in \gamma_{n-1}^*(v, r) \} ,
\]

(4.27)

\[
\delta_R(v^\circ) = \{ r \in \delta_R \mid (v, r, i)^\circ = v^\circ \quad \text{for an index } i \in \gamma_{n-1}^*(v, r) \} .
\]

(4.28)

Now we can define \(\delta_n\), so for each \(v \in V_{n-1}\) we set:

\[
\delta_n(v) = \delta_{n-1}(v) \quad \text{if } v \not\in W ,
\]

(4.29a)

\[
\delta_n(v) = \delta_Q \cup \delta_R(v) \cup \begin{cases} 
\delta_I \setminus \delta_I^{3^e}, & \text{if } v \not\in W_S \\
\delta_I, & \text{if } v \in W_S
\end{cases} \quad \text{if } v \in W ,
\]

(4.29b)

\[
\delta_n(v^\circ) = \delta_Q \cup \delta_R(v^\circ) \cup \delta_I \quad \text{if } v \in W .
\]

(4.29c)

**The Witness Function \(\beta_n\)**

This is the most technical part of the construction. In most cases, we will use a definition similar to \(\beta_n((v, p, i)^\circ, p, i) = \beta_{n-1}(v, p, i)\). But in some cases we have to modify the witness for the application of a send clause.

Hence, we distinguish between the following cases for \(v \in V_n\) and \(p \in \delta_n(v)\) if neither (ADAG iii a) nor (ADAG iii b) applies for \(v^\circ\) and \(p\) in \(\mathcal{D}_{n-1}\):
1. For $p \in \delta_n(v) \cap \tilde{R}$ we define
\[
\beta_n(v, p, i) = ((v', p', i', \varphi') \cap p', i', \varphi) \quad \text{for } \beta_{n-1}(v', p, i) = (v', p', i', \varphi) \quad (4.30a)
\]

2. For $p \in \delta_n(v) \cap \tilde{I}$ we have the following cases:
   a) If we have $\text{Fix}_{D_{n-1}}(v^\circ) \cap W = \emptyset$, or
   b) if we have $\text{Var}_{D_{n-1}}(v^\circ) \subseteq \text{Steps}_{D_n}(\leq n)$, we do nearly the same as in $4.30a$:
   \[
   \beta_n(v, p, 0) = ((v', p', i', \varphi') \cap p', i', \varphi) \quad \text{for } \beta_{n-1}(v', p, 0) = (v', p', i', \varphi) \quad (4.30b)
   \]
   c) Otherwise, we have $\text{Var}_{D_{n-1}}(v^\circ) \nsubseteq \text{Steps}_{D_n}(\leq n)$, but we know that $v^\circ$ is a goal (let $e$ be the stage with $1^e \in \delta_{n-1}(v^\circ)$), so by Lemma $4.4$ we have $e \gg n$. But then by $(4.19b)$ we know $\beta_{n-1}(v^\circ, 1) = (v', p', i', \varphi)$ for a clause $\varphi \in \Phi_f$ with $\varphi = [\ldots, p'(t', k) \rightarrow t'(t)]$.
   As case a) did not apply, we know $\text{Fix}_{D_{n-1}}(v^\circ) \cap W \neq \emptyset$, so we know that a node $w \in \text{Fix}_{D_{n-1}}(v^\circ)$ is in $W$, i.e., a descendant of an $n$-goal $u$. We will define a substitution $\sigma$ which maps each regular variable $x$ to a term if and only if $x$ is corresponding to such a node $w \in W$.
   Hence, we need the embedding $\theta_v$ which embeds $(\sigma_x(t), i')$ to $(v, v')$ as defined in $3.3$ for $D_{n-1}$. Now we define $\sigma$ to be a substitution, the domain of $\sigma$ will be the set $X_\sigma = \{ x \in \text{var}(t) \mid \theta_v(x) \in W \}$. Hence, for each $x \in X_\sigma$ we do the following steps, as illustrated in Figure $4.1$.
   Let $u$ be an $n$-goal such that $w = \theta_v(x)$ is a descendant of $u$ in $D_{n-1}$. Now let $\psi$ be a send clause such that $\beta_{n-1}(u, l) = (u', r', i', \psi)$. We assume that
\[ \psi = [\ldots r'(s', k') \rightarrow I'(s)] \], so let \( \theta_n \) be the embedding which embeds \((\sigma(v)(s), s')\) to \((u, u')\) as defined in \(\text{ADAG iii d} \) for \(D_{n-1}\).

Because of Lemma 4.4 there is a term \(s_x \in \text{sub}(s)\) with \(\theta_n(s_x) = w\): We know that \(u\) is an \(n\)-goal, \(w\) is a descendant of \(u\), and \(w \notin \text{Steps}_{D_{n-1}}(\leq t)\) because of the definition of \(W\).

Hence, we now set \(\sigma(x) = s_x\). We assume that \(\text{var}(s_x) \cap (\text{var}(t) \cup \text{var}(\kappa)) = \emptyset\), else we have to rename the variables in the intersection.

After repeating this for each \(x \in X_v\), we set

\[
\beta_n(v, p, 0) = ((v', p', i', \varphi) \in I'(v, p, \varphi)) \quad \text{for } \beta_n(v, p, 0) = (v', p', i', \varphi). 
\]

(4.30 c)

Note that we may have an \textit{anonymous} variable in the term \(s_x\), i.e., we took \(s_x\) from \(\text{sub}(s)\) and not, for example, from \(\text{sub}(\sigma(v)(s))\), but – as shown later – this does not raise any problems.

This completes our \textit{ADAG} \(D_n = (\text{ADAG } D_n, \tilde{\Phi}, \tilde{\beta}, \alpha, \beta_n, \gamma_n, \delta_n, \lambda_n)\), we now have to show that this actually is an \textit{ADAG} and that the three conditions (4.19 a), (4.19 b), and (4.19 c) hold.

\textbf{Proof of the ADAG Conditions}

We will show that \(D_n\) actually is an \textit{ADAG}. To do so, we first have to check the type of each component.

The graph \(D_n\) is acyclic: If there is a cycle \(v_1, \ldots, v_m \in D_n\), we have a cycle \(v_1' \ldots, v_m'\) in \(D_{n-1}\). The type of the other components is easy to check from the definition of each component; and when later proving condition (4.19 c) we will show that in the definition of \(\beta_n\) we used instances of send clauses that actually are in \(\tilde{\Phi}\).

Now we show that \(\text{ADAG i a} \) to \(\text{ADAG iii d} \) do still hold.

(i) For \(\text{ADAG i a} \) to \(\text{ADAG iii c} \) note that we did not alter \(\alpha\). Hence, for each node \(v = a(t)\) for a term \(t \in \{t_n \mid n \in \{1, \ldots, N\}\} \cup \{k \mid k \in \text{dom}(f)\} \cup \{\delta\}\), we know \(\Gamma^t \in \delta_{n-1}(v)\) for an appropriate stage \(e_t\). We show that \(\Gamma^t \in \delta_n(v)\), i.e., that \(I\) was not removed: We know that if \(v \in W\), then \(v \) is also in \(W_5\). Thus, in the definition of \(\delta_n\) we did not remove \(\Gamma^0\), as only (4.29 a) or the second case of (4.29 b) are possible.

(ii) For \(\text{ADAG ii} \) just assume that \(\Gamma^1, \Gamma^2 \in \delta_n(v)\) for \(e_1 \neq e_2\), then we would have \(\Gamma^1, \Gamma^2 \in \delta_{n-1}(v')\), which contradicts the inductive hypothesis that \(\text{ADAG ii} \) holds for \(D_{n-1}\).

(iii) Let \(v \in V_n\) be a node, \(p \in \delta_n(v)\) be a predicate symbol. By construction and inductive hypothesis, one of the for cases \(\text{ADAG iii a} \) to \(\text{ADAG iii d} \) holds for \(v'\) and \(p \in \delta_{n-1}(v')\). Hence, we can distinguish between this four cases:
(iii a) We know \( \text{ADAG iii a} \) holds for \( v^< \) and \( p \) in \( D_{n-1} \). First assume that \( v^< = a(c) \).
As \( a(c) \in \text{Steps}_{D_0}^{(1 \leq 1)} \) for each \( \text{ADAG} \ D \) (the first message of the intruder can only contain terms constructed from \( c \)), we know \( v^< \notin W \), so \( v^< = v \). This means \( v = a(c) \) and \( p = I^{(0,0)} \in \delta_n(v) \), so \( \text{ADAG iii a} \) holds for \( v \) and \( p \).
Now assume that \( v^< = a(s_m) \) for an \( m \in \{1, \ldots, N \} \). Then we have \( p = r_m^{(m,0)} \in \delta_{n-1}(v^<) \). We defined \( (v^<, r_m^{(m,0)}, 0) = v^< \), so we did neither change \( \gamma(v, p, 0) \) nor \( \lambda(v, p, 0, z) \) for any \( z \), so \( \text{ADAG iii a} \) holds for \( v \) and \( p \).

(iii b) If \( \text{ADAG iii b} \) holds for \( v^< \) and \( p \) in \( D_{n-1} \), the same pop clause \( \psi \) will also be sufficient as a justification for \( v \) and \( p \) in \( D_n \). We will show this for multiple cases:

For the case \( p \in (Q \cup \{ I^c \mid c \leq n \}) \), we know that the necessary pop symbols \( p_1, \ldots, p_m \) in the labeling of the descendants of \( v^< \) were copied from \( \delta_{n-1}(w) \) to both \( \delta_n(w) \) and \( \delta_n(w^\gamma) \) for each descendant \( w \) of \( v^< \).

Take a look at \( p \in \{ I^c \mid c \geq n \} \). Let \( v \in V_n, w \in V_{n-1} \) be nodes with \( p \in \delta_{n-1}(v^<) \) and \( p' \in \delta_{n-1}(w) \), such that \( p \) was produced by an intruder pop clause using \( p' \). We know \( v^< \to D_{n-1} w \), and to show that \( \text{ADAG iii b} \) still holds, we have to show that \( v \to D_n w' \) for a node \( w' \in \{w, w^\gamma\} \) with \( p' \in \delta_n(w') \).

- If we have \( v \in W^\gamma \), we constructed an edge \( v \to D_n w' \) by \ref{e:adag3a} distinguish between two cases:
  - For \( w \in W \) we have \( w' = w^\gamma \) and \( p' \in \delta_n(w^\gamma) \) by \ref{e:adag3c}.
  - For \( w \notin W \) we have \( w' = w \) and \( p' \in \delta_n(w) \) because of \ref{e:adag2a}.

- Otherwise we have \( v \notin W^\gamma \), i.e., \( v \in V_{n-1} \). We constructed an edge \( v \to D_n w' \) as we applied either condition \ref{e:adag3b} or condition \ref{e:adag3c}.
  - If we used \ref{e:adag3b}, we know \( w \in W \), then again we set \( w' = w^\gamma \) and have \( p' \in \delta_n(w^\gamma) \) by \ref{e:adag3c}.
  - If \ref{e:adag3c} was applied, we have \( v \in W, w \in W \) or \( v \in \text{Goals}_{D_{n-1}} \).

If the latter one applies, we have nothing to show, as we will handle the case \( \text{ADAG iii c} \) later.

For \( w \notin W \) we again set \( w' = w \) and know \( p' \in \delta_n(w) \) because of \ref{e:adag2a}. Assume that \( w \in W \).

For \( v \in W \) we know \( v \in W_S \), else, we would have removed \( p \) as defined in \ref{e:adag2b}. But then by \ref{e:adag2} we would also have \( w \in W_S \) because \( w \in W \) and \( v \to D_{n-1} w \). Then for \( w' = w \) we know \( v \to D_n w \) as well as \( p' \in \delta_n(w) \) because of \ref{e:adag2b}.
Now assume that $v = v^\triangleright$, i.e., $(v, p, i)\triangleright = v$. Then by the definition of $(v, p, i)\triangleright$ we know that there are no nodes $\vartheta, w \in V_{n-1}$ with $\vartheta \sim_{D_{n-1}} w \sim_{D_{n-1}} (v, r, i)$, with $\vartheta \not\in W$, and $w \in W$ such that (4.24b) holds for $\vartheta$ and $w$. But then the same applies for $(v', p', i')$, leading to $(v', p', i')\triangleright = v'$. This means $p' \in \delta_n(v')$ and $i' \in \gamma_n^*(v', p')$, and this also means $v' \rightarrow_{D_n} v$ because of condition (4.24b) (remember that (4.24b) did not apply).

- First, assume that $v \neq v^\triangleright$, this means $v \in W^\triangleright$. We again use the definition of $(v, p, i)\triangleright$, but this time we know that there are nodes $\vartheta, w \in V_{n-1}$ with $\vartheta \sim_{D_{n-1}} w \sim_{D_{n-1}} (v, p, i)$ and with $\vartheta \not\in W$, but $w \in W$ such that (4.24b) holds for $\vartheta$ and $w$.

We then can either have $w = v^\triangleright$, then we know $\vartheta = v'$ and we have $v' \rightarrow_{D_n} v$ because of (4.24b). Then $v' = v \not\in W$ also implies $p' \in \delta_n(v')$ and $i' \in \gamma_n^*(v', p')$.

Or we have $w \neq v^\triangleright$, but then we know $v' \neq v^\triangleright$: There are nodes $\vartheta, w \in V_{n-1}$ with $\vartheta \sim_{D_{n-1}} w \sim_{D_{n-1}} (v', p', i') \sim_{D_{n-1}} (v^\triangleright, p, i)$ with $\vartheta \not\in W$, but $w \in W$ such that (4.24b) holds for $\vartheta$ and $w$. Then we know $v' \rightarrow_{D_n} v$ by condition (4.24a), and we also know $p' \in \delta_n(v')$ and $i' \in \gamma_n^*(v', p')$ because $(v', p', i')\triangleright = v'$.

(iii c) In this case, too, we will show that $\text{ADAG III}$ still holds. Hence, let $i \in \gamma_n^*(v, p)$, and let $\beta_n(v, p, i) = (v', p', i', \varphi)$. We know by construction that we have $\beta_n_{i-1}(v^\triangleright, p, i) = (v'^{\triangleright}, p', i')$, so we have to show that $v'$ is a father of $v$, that $p' \in \delta_n(v')$ and that $i' \in \gamma_n^*(v', p')$. Then we know that the registers are still compatible and the freshness function still points to the right registers because we did modify these functions accordingly.

Distinguish between two cases:

- First, assume that $v = v^\triangleright$, i.e., $(v, p, i)\triangleright = v$. Then by the definition of $(v, p, i)\triangleright$ we know that there are no nodes $\vartheta, w \in V_{n-1}$ with $\vartheta \sim_{D_{n-1}} w \sim_{D_{n-1}} (v, r, i)$, with $\vartheta \not\in W$, and $w \in W$ such that (4.24b) holds for $\vartheta$ and $w$. But then the same applies for $(v', p', i')$, leading to $(v', p', i')\triangleright = v'$. This means $p' \in \delta_n(v')$ and $i' \in \gamma_n^*(v', p')$, and this also means $v' \rightarrow_{D_n} v$ because of condition (4.24b) (remember that (4.24b) did not apply).

- Now assume that $v \neq v^\triangleright$, this means $v \in W^\triangleright$. We again use the definition of $(v, p, i)\triangleright$, but this time we know that there are nodes $\vartheta, w \in V_{n-1}$ with $\vartheta \sim_{D_{n-1}} w \sim_{D_{n-1}} (v, p, i)$ and with $\vartheta \not\in W$, but $w \in W$ such that (4.24b) holds for $\vartheta$ and $w$.

We then can either have $w = v^\triangleright$, then we know $\vartheta = v'$ and we have $v' \rightarrow_{D_n} v$ because of (4.24b). Then $v' = v \not\in W$ also implies $p' \in \delta_n(v')$ and $i' \in \gamma_n^*(v', p')$.

Or we have $w \neq v^\triangleright$, but then we know $v' \neq v^\triangleright$: There are nodes $\vartheta, w \in V_{n-1}$ with $\vartheta \sim_{D_{n-1}} w \sim_{D_{n-1}} (v', p', i') \sim_{D_{n-1}} (v^\triangleright, p, i)$ with $\vartheta \not\in W$, but $w \in W$ such that (4.24b) holds for $\vartheta$ and $w$. Then we know $v' \rightarrow_{D_n} v$ by condition (4.24a), and we also know $p' \in \delta_n(v')$ and $i' \in \gamma_n^*(v', p')$ because $(v', p', i')\triangleright = v'$.

(iii d) In this case we know $v^\triangleright$ was a goal, and we will show that $v$ is also a goal with $p = I^e$. Let $\beta_n(v, I) = (v', p', i', \varphi)$ with $\varphi = [\ldots, p'(i', k') \rightarrow p(t)]$. Then we know by definition of $\beta_n$ that $p' \in \delta_n(v')$, and that $i' \in \gamma_n^*(v', p')$. We only have to show that the embeddings are compatible, i.e., that $\sigma_k(t, i') \rightarrow (v, v')$ for the substitution $\sigma_k = \text{subst}(k', \gamma_n(v', p', i'))$. As we did neither modify $k'$ nor $\gamma_n(v', p', i')$, we will still use the same substitution $\sigma_k$.

We can now distinguish between the cases we had when defining $\beta_n(v, I)$ by conditions (4.30b) and (4.30c).

In the first case, we have $\text{Fix}_{D_{n-1}}(v^\triangleright) \cap W = \emptyset$. Then we know that there are no nodes in $\text{Fix}_{D_{n-1}}(v^\triangleright)$ that are duplicated, and as we know $\text{Var}_{D_{n-1}}(v^\triangleright) \subseteq \text{Fix}_{D_{n-1}}(v^\triangleright)$, all nodes of variables of $v$ are left unchanged.

The same argumentations applies for the case $\text{Var}_{D_{n-1}}(v^\triangleright) \subseteq \text{Steps}_{D_{n}}(\leq n)$, because this also leads to $\text{Var}_{D_{n-1}}(v^\triangleright) \cap W = \emptyset$, so the nodes of variables of $v$ were left unchanged.
But in the last case, we may have copied some nodes of variables of \( v \) because we cannot ensure that \( \text{Var}_{D_{n-1}}(v^c) \cap W \) is empty. Again, see Figure 4.1 for illustration. Take a node \( w \in \text{Var}_{D_{n-1}}(v^c) \cap W \) corresponding to the variable \( x \). Then there is a node \( \bar{w} \neq w \), which could contradict the embedding condition: We may have

\[
\begin{align*}
  v & \rightarrow \bar{w}^c, & v & \not\rightarrow w, \text{ but } (4.31) \\
  v' & \not\rightarrow \bar{w}^c, & v' & \rightarrow w. \quad (4.32)
\end{align*}
\]

But we replaced \( x \) by a term \( s_x \), where the term \( s_x \) was taken from the set of subterms of a term \( s \) which embeds to an \( n \)-goal \( u \). Thus, let \( x' \) be a variable in \( \text{sub}(s_x) \). Then there is a node \( \bar{w}' \) corresponding to \( x' \) in the embedding of \( s \) to \( u \), and by applying Lemma 4.4 to \( D_{n-1} \) we know that \( \bar{w}' \) is in \( \text{Steps}_{D_{n-1}}(<n) \), so we know \( \bar{w}' \in \text{Steps}_{D_n}(\leq n) \). But then \( \bar{w}' \) was not split, and there are two paths \( w \rightarrow^* \bar{w}' \) and \( \bar{w}^c \rightarrow^* \bar{w}' \). This leads to \( v \rightarrow^* \bar{w}' \) and \( v' \rightarrow^* \bar{w}' \); thus for this \( \bar{w}' \) we have repaired the embedding such that \( x' \in \text{var}(I) \) and \( x' \in \text{var}(I') \) both embed to \( \bar{w}' \).

**Proof of Condition (4.19 a)**

Take a node \( v \in V_n \) that is a descendant of an \( m \)-goal \( u \in \text{Goals}_{D_n}(m) \) with \( m \leq n \). Let \( v \notin \text{Static}_D^+ \) (and thus, \( v^c \notin \text{Static}_{D_n}^+ \)). We have to show that \( v \notin \text{Goals}_{D_n}(>m) \), so we assume that \( v \in \text{Goals}_{D_n}(>m) \) and show that each of the following cases leads to a contradiction:

1. If \( m \) is smaller than \( n \), we get a contradiction with the inductive hypothesis \( 4.19 \text{a} \) because of \( v^c \in \text{Goals}_{D_{n-1}}(>m) \) and \( u^c \in \text{Goals}_{D_{n-1}}(m) \).

2. If \( m \) equals \( n \), we have three cases:

   a) First assume that \( v \in W^c \): In this case there would have to be an \( n \)-goal \( \bar{u} \in \text{Goals}_{D_n}(n) = \text{Goals}_{D_{n-1}}(n) \) and a path of the following kind with \( w_1, \ldots, w_l \in W \) (and thus, \( w_l \neq \bar{w}^c \)):

   \[
   \bar{u} \rightarrow_{D_n} \bar{w}_1^c \rightarrow_{D_n} \cdots \rightarrow_{D_n} \bar{w}_l^c \rightarrow_{D_n} v. \quad (4.33)
   \]

   But by (4.24b) there is no edge \( \bar{u} \rightarrow_{D_n} \bar{w}^c \), there is only an edge \( \bar{u} \rightarrow_{D_n} \bar{w} \).

   b) Now assume that \( v \notin W \). As \( v = v^c \notin \text{Static}_{D_{n-1}}^+ \), we have \( v \notin W_S \). Then by the definition of \( \delta_n \) (4.29b) there is no \( I^* \in \delta_n(v) \) for any \( e \gg n \).

   c) Finally, consider \( v \notin W \cup W^c \), i.e., \( v^c \notin W \). Take a look at the definition of \( W \) in (4.20) and distinguish between the different possibilities why we have \( v \notin W \): We know that \( v^c \) cannot be an \( n \)-goal (then we would have nothing to show, see (ADAG ii)), and by assumption about \( u \) we have \( u^c \in \text{Goals}_{D_{n-1}}(n) \).
and \( u^< \to^* v^< \), so by definition of \( W \) we know \( v = v^< \in \text{Steps}_{D_{n-1}}(\leq n) = \text{Steps}_{D_n}(\leq n) \).

Lemma 4.5 states there is an \( m \leq n \) with \( a(t_m) \to_{D_n}^* v \). Let \([v_1, \ldots, v_l]\) be a path in the DAG \( D_n \) from \( v_1 = a(t_m) \) to a leaf \( v_l \) with \( v = v_k \) for a \( k \leq l \), as shown in Figure 4.2.

By the definition of an ADAG (ADAG1) there is a stage \( e_1 \ll m \) with \( \Gamma_1 \in \delta_n(a(t_m)) \). And there is a minimal index \( i \) such that \( v_i \) is a goal for a stage \( e_i \leq e_1 \). Now assume that \( v \) is a goal for a stage \( e_\nu \gg n \). Then we know:

\[
e_\nu \gg n \geq m \gg e_1 \geq e_i.
\] (4.34)

Finally, we again consider three cases:

i. In the case \( k < i \) note that by choice of \( i \) we have \( \Gamma_j \in \delta_n(v_j) \) for a stage \( e_j \) with \( e_1 \geq e_j \geq e_i \) for all \( j \in \{1, \ldots, i\} \). But then we would have \( e_k \in \delta_n(v) \) with \( e_k \leq e_1 \ll e_\nu \), which contradicts (ADAG2) for \( v \).

ii. The case \( k = i \) directly contradicts (ADAG2), as this would give \( \Gamma^0 \in \delta_n(v_k) \) and \( \Gamma^i \in \delta_n(v_k) \).

iii. The case \( k > i \) contradicts the inductive hypothesis (4.19a): \( v \) would be descendant of a goal \( v_i \) with stage \( e_i \ll n \ll e_0 \).

**Proof of Conditions (4.19 b) and (4.19 c)**

Now we have to show that for every \( u \in \text{Goals}_{D_n}(m) \) for an \( m > n \) and for every \( v \in \text{Var}_{D_n}(u) \setminus \text{Var}_{D_n}(u) \) we have \( v \in \text{Steps}_{D_n}(< n) \). Therefore, let \( u \) be an \( m \)-goal, and let \( v \in \text{Var}_{D_n}(u) \setminus \text{Var}_{D_n}(u) \).

We know that \( v \) is a node that corresponds to a variable that was introduced by an instantiation during the definition of \( \beta_n \) for an \( n' \). We have to show that \( v \in \text{Steps}_{D_n}(< n) \).

If \( n' < n \), by inductive hypothesis (4.19b) we already have \( v \in \text{Steps}_{D_n}(< n) \). Now assume that \( n' = n \), i.e., during the definition of \( D_n \) we replaced a variable \( x \) corresponding to a node \( w \) with a subterm \( s_x \), such that there is a variable \( x' \in \text{var}_X(s_x) \) which \( v \) corresponds to. The term \( s_x \) was taken from the set of subterms of a term \( s \) which embeds to an \( n \)-goal.
scaling down ADAGs

\[ u' \]

But then by Lemma 4.4, we know that the node corresponding to \( x' \) is in \( \text{Steps}_{D_n}(<n) \), so we know \( v \in \text{Steps}_{D_n}(<n) \subseteq \text{Steps}_{D_n}(\leq n) \).

As mentioned earlier, we still have to show that when defining \( \beta_n \), we actually used instances that are in \( \hat{\Phi}_f \), i.e., that are not too deep; this is also necessary to show the third condition.

Note that we do only instantiate each variable once, so if we replaced \( x \) by \( s_x \), we know that the node \( w \) that corresponds to \( x \) is in \( \text{Var}_{D_n}(u) \). The resulting nodes \( w'_1, w'_2, \ldots \) corresponding to the variables in \( s_x \) will then be in \( \text{Steps}_{D_n}( < n) \). Thus, for \( D_{n+1}, \ldots, D_N \), the nodes \( w_i' \) will not be in \( W \), and hence, the variables occurring in \( s_x \) will not be substituted again.

Also note that each such \( x \) corresponding to a node \( w \in \text{Var}_{D_n}(u) \) is reachable from \( u \) with a path that is shorter than \( |P| \). We extend the length of that path by the depth of \( s_x \), so to prove both conditions we only have to know that the depth of \( s_x \) is bounded by \( |P| \cdot n \), but this is the third condition \( 4.19c \) of the inductive hypothesis.

Now we have shown that our construction creates an ADAG \( D_n \) for which all three of our conditions hold. Thus, after constructing \( D_N \) from \( D_{N-1} \) in this way we get an ADAG which is simple.

### 4.3 Bounding the Number of Goals

Now every ADAG can be transformed into a simple one. The next theorem states that a simple ADAG can be transformed into an ADAG with an exponentially bounded number of goals.

This transformation uses the observation that one part of the relevant information of a node \( v \) is the predicate symbols its labeled with, i.e., the value of \( \delta(v) \). Hence, from the set of all goals \( u \) which have the same labeling \( \delta(u) \), we can pick a specific one as a representative.

During this process we will not delete any nodes, but we will delete the predicate symbol \( I^c \) from some nodes, transforming them from goals to non-goals. The other components of the ADAG like the witness and the labeling function are adjusted accordingly.

In fact there are a goals we do not want to alter at all, for example, the goals in the static part of the protocol. Nevertheless, after this transformation we have an exponentially bounded number of goals without deleting any nodes.

**Theorem 4.9** (Bounded Number of Goals). Let \( \Pi \) be a protocol. If \( D = (D, \Psi, \bar{\Gamma}, \alpha, \beta, \gamma, \delta, \lambda) \) is a simple ADAG for \( \Pi \) and \((\pi, f)\) where each term occurring in \( \Psi \) has at most polynomial depth in terms of the size of the protocol, then there exists an ADAG \( D' = (D', \Psi, \bar{\Gamma}, \alpha, \beta', \gamma', \delta', \lambda) \) for \( \Pi \) and \((\pi, f)\), where \( \text{Goals}_{\Sigma'Y} \) is exponentially bounded in terms of the size of \( \Pi \).
4.3.1 Proof of Theorem 4.9

**Proof.** This proof, too, will be done by constructing a sequence \([D_{N+1}, \ldots, D_1]\) of ADAGs, this time starting with \(D = D_{N+1}\) and ending with an ADAG \(D_1\) with a bounded number of goals. The inductive hypothesis is straightforward: For each \(n \in \{1, \ldots, N\}\) we will construct \(D_n\) from \(D_{n+1}\) such that the following condition holds for \(D_n\):

\[
\text{Goals}_{D_n}(\geq n) \text{ is exponentially bounded}.
\]

Again, this condition holds for any ADAG \(D_{N+1}\) we start with; and after we constructed an ADAG \(D_1\) for which this condition holds, we have proven the theorem.

Let \(n \in \{1, \ldots, N\}\). Let \(D_{n+1} = (D_{n+1}, \Psi, \tilde{\Gamma}, \alpha, \beta_{n+1}, \gamma_{n+1}, \delta_{n+1}, \lambda)\) be a simple ADAG with \(D_{n+1} = (V, F_{n+1}, \mu)\) being the DAG. Assume as inductive hypothesis that the condition holds for \(D_{n+1}\). We will construct \(D_n = (D_n, \Psi, \tilde{\Gamma}, \alpha, \beta_n, \gamma_n, \delta_n, \lambda)\) with \(D_n = (V, F_n, \mu)\), i.e., use the same stage theory, the same set of anonymous variables, the same embedding and freshness functions and even the same set of nodes. Hence, we will only define the new set of edges \(F_n\) and the new predicate labeling, register and witness functions \(\delta_n, \gamma_n\), and \(\beta_n\).

**Selecting the Protected Goals**

Let \(G = \text{Goals}_{D_{n+1}}(n)\) be the set of \(n\)-goals, we will define a subset \(G^\triangleright \subseteq G\) of protected goals, i.e., nodes we keep as goals, whereas for all nodes \(v\) in \(G^\triangleleft = G \setminus G^\triangleright\) we will later remove \(\triangleleft v\) from \(\delta_n(v)\).

First, we will define an equivalence relation on \(G\) according to the predicate labeling of the goals: For \(u_1, u_2 \in G\) let \(u_1 \equiv u_2\) if \(\delta_{n+1}(u_1) = \delta_{n+1}(u_2)\). We will need a representative from each equivalence class of \(\equiv\), so for each class \(G_u\) for a goal \(u\), let \(\equiv^u\) denote a unique node such that no goal \(w \in G_u\) is a descendant of \(\equiv^u\).

We will put \(\equiv^u\) in \(G^\triangleright\), but some other goals need to be protected, too; so we define \(G^\triangleright\) to be the least subset of \(G\) such that

\[
\begin{align*}
\{ u \equiv^u | u \in G \} & \subseteq G^\triangleright, \quad (4.36a) \\
\text{Fix}_{D_{n+1}}(v) \cap G & \subseteq G^\triangleright \quad \text{for all } v \in \text{Goals}_{D_{n+1}}(\geq n), \quad (4.36b) \\
\text{Static}_{D_{n+1}}^{\triangleright} \cap G & \subseteq G^\triangleright, \quad (4.36c) \\
\{ u \in G | u \sim_{D_{n+1}}^v \} & \subseteq G^\triangleright \quad \text{for all } v \in \text{Goals}_{D_{n+1}}(\geq n), \quad (4.36d) \\
\{ u \in G | G^\triangleright \nrightarrow_{D_{n+1}} u \} & \subseteq G^\triangleright. 
\end{align*}
\]

The reasons why we need to protect all these goals are hidden in the details of the proof. The first line states that for each occurring value of \(\delta\), we need to keep one representative. We need the second line to protect all the fixed parts of \(m\)-goals for \(m > n\) so the embeddings in \(D_{n+1}\) are still compatible – we will later show that this protection is
also guaranteed for \( m = n \). The third line protects nodes in the static part of the DAG, which eases our argumentation in the following steps because, e.g., the property of »simple« only holds for non-static goals. Condition (4.36 d) is necessary to prevent damage to \( \beta \)-paths from a node to a goal: The \( \beta \)-paths move along the normal edges of the DAG, and we will change some edges, so by this condition we guarantee that no \( \beta \)-path is damaged. Finally, (4.36 e) and the definition of \( u^\Xi \), or more precisely the fact that \( u^\Xi \) has no descendants in its equivalence class, are necessary to guarantee that we will get a DAG. All this parts will be referred to in the proof later.

We will now start with defining our ADAG \( D_n \), i.e., the functions \( \delta_n, \beta_n, \gamma_n \) and the term DAG \( D_n \).

### The Predicate Symbol Labeling Function \( \delta_n \)

Let \( v \in V \) be a node. Again, let \( \delta_Q = \delta_{n+1}(v) \cap Q \), as well as \( \delta_I = \delta_{n+1}(v) \cap I \). We will set \( \delta_R = \delta_{n+1}(v) \cap R \), but also define \( \delta_R' \subseteq \delta_R \) to only contain the necessary push symbols, i.e., let \( \delta_R' \) be the set of all \( r \in \delta_R \) with \((v, r, i) \sim_{D_{n+1}}^* u \) for an index \( i \) and a goal \( u \not\in G^\equiv \).

Now we can define \( \delta_n \) for each \( v \in V \):

\[
\begin{align*}
\delta_n(v) &= \delta_Q \cup \delta_R' \quad \text{if } v \in G^\equiv, \\
\delta_n(v) &= \delta_Q \cup \delta_R' \cup \delta_I \quad \text{if } v \not\in G^\equiv.
\end{align*}
\]

### The Witness Function \( \beta_n \) and the Register Function \( \gamma_n \)

For each node \( v \in V \) and \( p \in \delta_n(v) \cap (R \cup I) \) we will reduce \( \beta_n \) and \( \gamma_n \) to the necessary parts.

Let \( i \in \gamma_{n+1}^*(v, p) \). If \((v, p, i) \sim_{D_{n+1}}^* u \) for a goal \( u \not\in G^\equiv \), we will set \( \beta_n(v, p, i) = \beta_{n+1}(v, p, i) \) and \( \gamma_n(v, p, i) = \gamma_{n+1}(v, p, i) \); otherwise let \( \beta_n(v, p, i) \) and \( \gamma_n(v, p, i) \) be undefined.

### The Term DAG \( D_n \)

As stated above, we do neither remove nor add any nodes. But we will modify some edges: All I-symbols that are created by intruder pop clauses which are based on a goal \( u \in G^\equiv \) will now be based on \( u^\Xi \). But as the pop clauses follow the edges of the DAG, we have to modify the DAG:

Let \( W \) be the set of all nodes \( w \) with \( I^\epsilon \in \delta_{n+1}(w) \), but \( w \not\in \text{Goals}_{D_{n+1}} \), i.e., we know that (ADAG III.a) or (ADAG III.b) holds for \( w \) and \( I^\epsilon \) in \( D_{n+1} \), so the I-symbol was derived by an intruder pop clause (or \( w = a(c) \)). Now define the set of edges \( F_n \) such that for all \((v, u, l) \in F_{n+1} \) we have

\[
\begin{align*}
(v, u^\Xi, l) &\in F_n \quad \text{if } v \in W \text{ and } u \in G^\equiv, \\
(v, u, l) &\in F_n \quad \text{if } v \not\in W \text{ or } u \not\in G^\equiv.
\end{align*}
\]
This completes our ADAG $D_n = (D_n, \Psi, \tilde{T}, \alpha, \beta_n, \gamma_n, \delta_n, \lambda)$, we now show that we constructed a simple ADAG and condition 4.35 holds.

**Proof of the ADAG Conditions**

We will show that $D_n$ actually is an ADAG. As we kept most of the components, there is not much to show for the types – the only interesting part here is to see that $D_n$ is acyclic. To do so, assume that there is a cycle $v_1, \ldots, v_j$ in $D_n$.

As $D_{n+1}$ is acyclic, there must be an edge $v_j \rightarrow_{D_n} v_{j+1}$, that is modified, i.e., in $F_n$ but not in $F_{n+1}$. By (4.36c) we know that $G^\triangleright$ is closed under $\rightarrow^*$, so no descendant of $v_{j+1}$ is in $G^\triangleright$, which means that (4.38a) cannot be applied for any descendant of $v_{j+1}$.

Hence, $v_j \rightarrow_{D_n} v_{j+1}$ is the only edge in the cycle that was modified, i.e., $v_{j+1} \rightarrow_{D_{n+1}}^* v_j$.

By (4.38a) there is a node $u$ with $u^\equiv = v_{j+1}$ and $(v_j, u, l) \in F_{n+1}$. But then we have

$$u^\equiv = v_{j+1} \rightarrow_{D_{n+1}}^* v_j \rightarrow_{D_{n+1}} u.$$  \hspace{1cm} (4.39)

But $u^\equiv \rightarrow^* u$ contradicts the selection of the representative $u^\equiv$; so our assumption that there is a cycle must be wrong.

Now we have to check if (ADAG iii a) to (ADAG iii d) do still hold.

(i) Because of (4.36c) all nodes in the range of $\alpha$ are elements of $G^\triangleright$, so we did not modify their predicates.

(ii) We did at most remove predicates $\Gamma^i$, so condition (ADAG ii) still holds.

(iii) Let $v \in V$ and $p \in \delta_n(v)$. As $D_{n+1}$ is an ADAG and $\delta_n(v) \subseteq \delta_{n+1}(v)$, we know one of the four conditions (ADAG iii a) to (ADAG iii d) holds in $D_{n+1}$. We will distinguish between these four cases and show that each case still holds for $v$ and $p$ in $D_n$:

(iii a) For this case there is nothing to show.

(iii b) In this case we may have modified an edges $v \rightarrow_{D_{n+1}} v_j$ to $v \rightarrow_{D_n} v_j^\equiv$. But as $\delta_{n+1}(v_j) = \delta_{n+1}(v_j^\equiv)$ and $\delta_{n+1}(v_j^\equiv) \cap (Q \cup \tilde{T}) = \delta_n(v_j^\equiv) \cap (Q \cup \tilde{T})$, the condition (ADAG iii b) still holds for $v$ and $p$.

(iii c) In this case, too, we show that (ADAG iii c) still holds, mainly because we did not change most of the components. We have to show this for each $i \in \eta^*_n(v, p)$, i.e., each $i$ for which $\beta_n(v, p, i)$ is defined. Therefore, let $\beta_n(v, p, i) = \beta_{n+1}(v, p, i) = (v', p_m, i', \psi)$. During the definition of $\delta^*_R$ we might have removed the predicate $p_m$ from $\delta^*_R$ for $v'$, or we might have removed the edge between $v'$ and $v$, both would be a problem. The predicates $p_1, \ldots, p_{m-1}$ are pop symbols so we did not modify them.

But because $\beta_n(v, p, i)$ is defined we know that there is a goal $u \notin G^\triangleright$ with $(v, p, i) \leadsto_{D_{n+1}}^* u$. As we also know $(v', p_m, i') \leadsto_{D_{n+1}} (v, p, i)$, we know that $(v', p_m, i') \leadsto_{D_{n+1}}^* u$, and thus, we have $p_m \in \delta^*_R$ for $v'$.
Now we might still have damaged the DAG such that \( v' \) no longer is a parent of \( v \), but of another node \( v'' \). For this to be the case, \( v \) would have to be an \( n \)-goal in \( G^<_n \). Again look at the goal \( u \) with \( (v,p,i) \sim_{D_{n+1}}^* u \), and let \( e_u \) be the stage of \( u \). In Lemma 4.10 we will later show that \( e_u \gg n \). Then we have \( v \in G^>^n \) by (4.36d), so we have not altered the edge from \( v' \) to \( v \).

(iii d) This case is the most technical one. Again, we will show that (ADAG iii d) still holds. Note that for each goal \( u \notin G^<_n \) we have \( (u,p,0) \sim_{D_{n+1}}^0 (u,p,0) \) for \( p \in \bar{I} \), so the witness function for \( v, p \) and 0 is defined.

Let the witness be \( \beta_n(v,p,0) = \beta_{n+1}(v,p,0) = (v',p_m,t',\psi) \) with the send clause \( \psi = [p_1(t'),\ldots,p_m(t',\kappa') \to p(t)] \). Again, the definition of \( \delta'_R \) might have removed the predicate \( p_m \) from \( \delta'_R \) for \( v' \), or we might have modified edges in Fix\(_{D_{n+1}}(v) \) (or between \( v' \) and \( v \)).

The predicates \( p_1,\ldots,p_{m-1} \) are pop symbols which we did not modify. And as \( v \) was a goal in \( D_{n-1} \), we set \( p_m \in \delta'_R \), so we have \( p_m \in \delta_m(v') \).

In \( D_{n-1} \), the pair \( (\sigma_k(t),t') \) embeds to \( (v,v') \), hence, it also embeds in \( D_n \) as long as we did not modify any edge in Fix\(_{D_{n+1}}(v) \), i.e., as long as we do not have any \( u \in \text{Fix}_{D_{n+1}}(v) \) with \( u \in G^<_n \). Assume that there is such an \( u \), we will show that this leads to a contradiction. First, we know that \( u \) is an \( n \)-goal as it is in \( G^<_n \) and it is either a descendant of \( v \) or of \( v' \), or both.

- If \( v' \to^* u \) holds, note that \( u \in G^<_n \) means \( u \notin \text{Static}^+_D_{n+1} \) by (4.36c). We then either have \( u \in \text{Steps}_{D_{n+1}}(\leq n) \) or \( u \notin \text{Steps}_{D_{n+1}}(\leq n) \). In the first case, we use Lemma 4.11 so we have \( u \notin \text{Goals}_{D_{n+1}}(\geq n) \), so \( u \notin G^<_n \).

In the latter case, \( u \notin \text{Steps}_{D_{n+1}}(\leq n) \) implies \( v' \notin \text{Steps}_{D_{n+1}}(\leq n) \), so \( v \notin \text{Goals}_{D_{n+1}}(\leq n) \). But then we have \( v \in \text{Goals}_{D_{n+1}}(\geq n) \), which means \( u \in \text{Fix}_{D_{n+1}}(v) \cap G \subseteq G^>^n \) by (4.36d). This contradicts \( u \in G^<_n \).

- If \( v \to^* u \) holds, again note \( u \in G^<_n \) means \( u \notin \text{Static}^+_D_{n+1} \) by (4.36c). As \( D_{n+1} \) is simple, we know \( v \notin \text{Goals}_{D_{n+1}}(< n) \). But as above, we have \( v \notin \text{Goals}_{D_{n+1}}(\geq n) \), otherwise, we would have \( u \notin G^<_n \). Hence, we have \( v \in \text{Goals}_{D_{n+1}}(n) \) = \( \Gamma \). But the I-symbol \( p \) is still in \( \delta_m(v) \), so we have \( v \in G^>^n \). Then by (4.36c) we also have \( u \in G^>^n \), i.e., \( u \notin G^<_n \).

Now we know there is no node in \( \text{Fix}_{D_{n+1}}(v) \cap G^<_n \), so the structure of \( \text{Fix}_{D_{n+1}}(v) \) has not changed in \( D_n \). Then we also know \( (\sigma_k(t),t') \to (v,v') \).

This shows that \( D_n \) is still an ADAG. Note that it is also a simple ADAG: Take some nodes \( v,w \in V \) with \( v \to^* w \), but \( v \notin \text{Goals}_D(j) \) for a \( j \) and \( w \notin \text{Static}^+_D \). We have to show that \( w \notin \text{Goals}_D(> j) \).

During the definition of \( \delta_n \) we did only remove symbols, so because of the assumption that \( D_{n-1} \) was simple this could only happen if we changed an edge on the path from \( v \) to \( w \). This means there is an \( n \)-goal \( u \) such that \( v \to^*_D u \) and \( u \equiv \to^*_D w \), and we modified the edge pointing to \( u \) to an edge pointing to \( u'' \).
But then because of the modification and condition (4.36c) we know that \( u \notin \text{Static}_{D_n} \). Then we have \( j \geq n \), otherwise the nodes \( v \) and \( u \) would contradict the simplicity of \( D_{n+1} \). But we also know \( u \equiv \rightarrow_{D_{n+1}} w \) (remember that on each path we modified only one edge), and the simplicity of \( D_{n+1} \) tells us that \( w \notin \text{Goals}_{D}(>n) \), so \( w \notin \text{Goals}_{D}(>j) \subseteq \text{Goals}_{D}(>n) \).

Thus, we obtained a simple ADAG, and we will now have to show that condition (4.35) holds.

**Proof of Condition (4.35)**

We now show that \( \text{Goals}_{D_n}(\geq n) \) is exponentially bounded in terms of the size of the protocol. By inductive hypothesis, \( \text{Goals}_{D_{n+1}}(>n) \) is exponentially bounded, and as we did only modify \( n \)-goals, we also know \( \text{Goals}_{D_n}(>n) \) is exponentially bounded. Because we removed the \( I \)-symbol from all goals in \( G^< \), we only need to find a bound for \( G^> \).

Therefore, we will take a look at the five conditions (4.36a) to (4.36e) defining \( G^> \):

a) The number of goals in \( \{ u \equiv u \mid u \in G \} \) is bounded by the number of equivalence classes of \( \equiv \). As \( \equiv \) is defined by different values of \( \delta_{n+1} \) with \( \delta_{n+1} : V \rightarrow 2^{Q_1} \), we can have at least \( 2^{Q_1} \) different possible values, i.e., equivalence classes. This is exponential in terms of the size of the protocol.

b) As stated above, we know by inductive hypothesis that there are at most exponentially many goals in \( \text{Goals}_{D_n}(>n) \). As the depth of each send clause is polynomial, there can be an exponential number of goals in the fixed part of each goal. But exponentially many goals and exponentially many nodes for each of them still gives a total number of nodes that is exponentially bounded in terms of the size of the protocol.

c) The set \( \text{Static}_{D_{n+1}}^+ \) is defined by all nodes which are reachable from a node \( v = a(l) \) for a term \( t \in \text{sub}(T_P) \) in less than \( |P|^2 \) steps. This equals all nodes which are reachable from a node \( v' = a(l') \) for a term \( t' \in T_P \) (note the missing »sub«) in less than \( |P|^3 \) steps. As the maximum degree of the nodes is two, this leads to exponentially many nodes in \( \text{Static}_{D_{n+1}}^+ \).

For the following steps we can use Lemma (4.11) which shows that there are no additional goals in \( S = \text{Steps}_{D_{n+1}}(\leq n) \), so all goals that we will add in (4.36d) and (4.36e) are in \( G \setminus S \). Also note that Lemma (4.4) states that for each node \( u \in G \setminus S \), the nodes of variables of \( u \) are in \( S \).

d) By inductive hypothesis, there is only an exponentially bounded number of \( m \)-goals \( u \) for \( m > n \). We will show that on each \( \beta \)-path to such a goal, there can be only a polynomial number of \( n \)-goals \( u \) not in \( S \). Therefore, we will add a polynomially bounded number of goals \( v \) for each goal \( u \) greater than \( n \), leading to a total number of \( n \)-goals added in this step that is exponentially bounded.

So let \( u \) be an \( m \)-goal for an \( m > n \). Then by Lemma (4.2) there is a unique \( \beta \)-path \( v_1, \ldots, v_l \) from \( v_1 = a(l_m') \) to \( v_l = u \) for an \( m' \leq m \). By the lemma, we also know \( v_i \rightarrow_{D_{n+1}} v_{i+1} \) for all \( i \in \{1, \ldots, l-2\} \).
If no $n$-goal exists on the path, we have nothing to show. Therefore, assume that $i \in \{1, \ldots, l - 1\}$ is the minimal index of an $n$-goal on that path, i.e., $v_i$ is an $n$-goal. Lemma 4.4 shows that $v_i$ is in $S$ for each $j > n + |P|^2$. Hence, for $u$ we will only add $\{v_{i+1}, \ldots, v_{i+|P|^2}\} \cap G$ to $G^\triangleright$, which is polynomially bounded.

e) We have already shown that the set $G^\triangleright$ is exponentially bounded if we only use conditions 4.36a to 4.36d.

We will now show that for each $u$ added so far, there is only an exponentially bounded number of goals added by condition 4.36e. This shows that the whole set $G^\triangleright$ is exponentially bounded.

As we have already shown, the goals added by condition 4.36e cannot be in $S$. They have to be descendant of an $n$-goal $u \in G^\triangleright$, but the nodes of variables of $u$ are in $S$ (Lemma 4.4). Hence, the goals added by this condition can only be in the fixed part $\text{Fix}_{D_{n+1}}(u)$.

Now as the number of fixed nodes of a goal is exponentially bounded, we add only an exponentially bounded number of goals for each goal already in $G^\triangleright$ prior to condition 4.36e.

Thus, $G^\triangleright$ is exponentially bounded, which is the desired conclusion.

\[
\square
\]

4.3.2 Lemmas concerning Goals

We will show two lemmas we referred to in the proof above. The first states that an $n$-goal can only be used to send goals whose step is greater than $n$, i.e., that a $\beta$-path through an $n$-goal leads to a goal greater than $n$:

**Lemma 4.10.** Let $v$ be an $n$-goal in a simple ADAG $D$ with $v \not\in \text{Static}^*_D$. Assume that $v \leadsto^* (u, I^\gamma, 0)$. Then $n \ll \gamma$ holds.

**Proof.** For the proof let $v$ be an $n_1$-goal in a simple ADAG $D$ with $v \not\in \text{Static}^*_D$, and assume that we have $I^\alpha \in \delta(v)$ for $e_1 = (n_1, m_1)$ and $v \leadsto^* (u, I^\gamma, 0)$. We will show that $e_1 \ll e_7$ holds.

If we have $v \leadsto^* (u, I^\gamma, 0)$, we know that $r^e \in \delta(v)$ for a stage $e_6 = (n_6, m_6)$ with $e_6 \leq e_7$. But then we also have $(a_{\Delta SM}, r_n(\Delta SM), 0) \leadsto^* (v, r^e)$ for an $n_5 \leq n_6$. As $v \not\in \text{Static}^*_D$, we know by Lemma 4.5 that $v$ is a descendant of $w = a(t_{u_i})$ for an $n_4 \leq n_5$. By condition (ADAG1a) we have $I^\delta \in a(w)$ for a stage $e_3 \ll n_4$.

Now take the first goal on the path from $w$ to $v$. Either $v$ is that goal, then by the structure of the intruder pop clauses we know $e_1 \leq e_3$, or $v$ is a descendant of another goal $v'$ with $I^{v'} \in \delta(v')$ for a stage $e_2 \leq e_3$. In the latter case we have $e_1 \leq e_2$ as the ADAG is simple and $v \not\in \text{Static}^*_D$. 


Hence, altogether we know $e_1 \ll e_7$ by

$$
\left\{ \begin{array}{c}
e_1 \\
e_1 \leq e_2
\end{array} \right. \leq e_3 \ll n_4 \leq n_5 \leq n_6 \leq e_7.
$$

(4.40)

We now show another lemma used above to prove the ADAG conditions. This lemma states that a node belonging to a step is no goal of that step or of a later step. Intuitively this should hold for a normal ADAG, as any term processed by a principal in a step has to be known to the intruder earlier than that step, so is makes no sense to send that information again.

**Lemma 4.11.** Let $v$ be a node in a simple ADAG $D$ with $v \not\in \text{Static}_D^+$, but $v \in \text{Steps}_D(\leq n)$. Then $v \not\in \text{Goals}_D(\geq n)$ holds.

**Proof.** Let $v$ be a node in a simple ADAG $D$ such that $v \not\in \text{Static}_D^+$, but $v \in \text{Steps}_D(\leq n)$ for an $n \leq N$. If $v$ is no goal at all, we have nothing to show. Now assume that $v$ is a goal with $I^e_v \in \delta(v)$. Then we refer to Lemma 4.5 to show that there is an $m \leq n$ with $w \rightarrow^* v$ for $w = \alpha(t_m)$. Let $I^e_w \in \delta(w)$, we know $e_w \ll m$. We will show that $e_v \leq e_w$ holds, which will lead to $e_v \ll e_w \ll m \leq n$, i.e., $v \not\in \text{Goals}_D(\geq n)$.

We now use the same argumentation as in the previous lemma: Take the first goal on the path from $w$ to $v$. Either $v$ itself is that goal, then by the structure of the intruder pop clauses we know $e_v \leq e_w$. Or $v$ could be a descendant of another goal $u$ with $I^e_u \in \delta(v)$ for a stage $e_u \leq e_w$. In the latter case we have $e_v \leq e_u \leq e_w$ as the ADAG is simple and $v \not\in \text{Static}_D^+$.

### 4.4 Bounding the Size of an ADAG

Finally, the last theorem states that if we have such an ADAG with a bounded number of goals, we can reduce the total number of nodes in the ADAG (and the other relevant components) so that the whole ADAG is exponentially bounded.

**Theorem 4.12 (Bounded ADAGs).** Let $P$ be a protocol. If $D = (D, \Psi, \hat{\Gamma}, \alpha, \beta, \gamma, \delta, \lambda)$ is an ADAG for $P$ and $(\pi, f)$ with an exponentially bounded set of goals in terms of the size of $P$ and where each term occurring in $\Psi$ has at most polynomial depth in terms of the size of the protocol. Then there is an ADAG $D' = (D', \Psi, \hat{\Gamma}', \alpha', \beta', \gamma', \delta', \lambda')$ for $P$ and $(\pi, f)$ which is exponentially bounded.

As above, this is done by merging nodes with the same labeling – but now we respect the goals and the registers, i.e., we merge all nodes which have the same labeling, lead to the same goals and which have undistinguishable registers from a certain point of view. After this merge the ADAG has only an exponential number of nodes, while the additional structures like the labeling function are also exponentially bounded in terms of the protocol size.
For the ease of understanding we will present the main proof with the help of four lemmas that will be proven afterwards Sections 4.4.1 to 4.4.4.

Proof. Let \( \mathcal{P} \) be a protocol. Let \( \mathcal{D} = (D, \Psi, \hat{\Gamma}, \alpha, \beta, \gamma, \delta, \lambda) \) be an ADAG for \( \mathcal{P} \) and \( (\pi, f) \) with the restrictions mentioned in the theorem, and let \( D = (V, F, \mu) \).

We will first define a set \( \text{Fixed}_D \) of nodes that we will protect during the rest of the proof, and the complement \( \text{Free}_D \):

\[
\text{Fixed}_D = \text{Static}_D^+ \cup \bigcup_{u \in \text{Goals}_D} \text{Fix}_D(u), \quad (4.41)
\]

\[
\text{Free}_D = V \setminus \text{Fixed}_D. \quad (4.42)
\]

For the next part of the proof, we need to know that each node in \( \text{Free}_D \) has exactly one parent. By Lemma 4.13 this assumption is possible whilst still meeting the restrictions mentioned in the theorem. Hence, we will assume that \( \mathcal{D} \) is an ADAG where each \( w \in \text{Free}_D \) has exactly one parent.

To reduce the set of nodes \( V \), we will split \( \text{Free}_D \) into two sets of paths: push paths, which are labeled with push symbols originating in \( \text{Fixed}_D \) and leading to a goal, and pop paths, which lead to a pop symbol at the top of that path. We will first define the set of push paths:

\[
\text{Paths}_D^\triangledown = \left\{ [v_1, \ldots, v_m] \mid (v_1, r_{n,0}^{(n,0)}, 0) \sim v_2 \sim \cdots \sim v_m \sim (v, I_e, 0) \right\} \quad \text{for an } n \in \{1, \ldots, N\} \text{ with } v_1 = a(s_n) \quad (4.43)
\]

Now the set of pop paths is defined – and according to the fifth condition of Lemma 4.2 we can assume that the nodes on these paths are not labeled with any push symbols:

\[
\text{Paths}_D^\triangle = \left\{ [v_1, \ldots, v_m] \mid w \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_m \right\} \quad \text{for a node } w \in \text{Fixed}_D \cup \text{Paths}_D^\triangledown \quad (4.44)
\]

and \( v_1, \ldots, v_m \notin \text{Fixed}_D \cup \text{Paths}_D^\triangledown \)

such that for all \( v_m \rightarrow w' \) we know \( w' \in \text{Fixed}_D \cup \text{Paths}_D^\triangle \).

By abuse of notation we will write \( \text{Paths}_D^\triangledown \) or \( \text{Paths}_D^\triangle \) for the set of all nodes occurring in a path in \( \text{Paths}_D^\triangledown \) or \( \text{Paths}_D^\triangle \), respectively – as we already did in (4.44).

Note that we can assume that each node in \( \text{Free}_D \) is an element of either \( \text{Paths}_D^\triangle \) or \( \text{Paths}_D^\triangledown \). Let \( v \in V \) be a node. If \( \text{Goals}_D(v) \) is not empty, \( v \) is element of a \( \beta \)-path \( \vec{v} \in \text{Paths}_D^\triangledown \) by Lemma 4.2. Otherwise, we note that \( v \) leads to a leaf \( v' \) (possibly \( v = v' \)). By Lemma 4.3 we can assume that there is an ancestor \( w \in \text{Fixed}_D \), hence, take a path from \( w \) to \( v' \) through \( v \), then there is a subpath \( \vec{v} \) through \( v \) without nodes in \( \text{Fixed}_D \) or in \( \text{Paths}_D^\triangledown \), such that \( \vec{v} \in \text{Paths}_D^\triangle \).
Now Lemma 4.14 shows that by modifying the paths in Paths$_D$, the number of nodes on these paths may be bounded to a double exponential number. Afterwards we can apply Lemma 4.17 which proves the same for pop paths. This yields that the total set of nodes $V$ can be double exponentially bounded because $V = \text{Fixed}_D \cup \text{Paths}_D \cup \text{Paths'}_D$.

Finally we can transform the ADAG such that not only the set of nodes, but the whole structure is double exponentially bounded as shown in Lemma 4.18. This concludes the proof of our theorem. □

### 4.4.1 Splitting Nodes with Multiple Parents

**Lemma 4.13** (Single Parents). Let $D$ be an ADAG for a protocol $P$ with the same assumptions and restrictions as in Theorem 4.12. Then there exists an ADAG $D'$ still fulfilling these restrictions, but where each node in $\text{Free}_D$ has exactly one parent node.

**Proof.** Let $D = (D, \Psi, \tilde{\Gamma}, \alpha, \beta, \gamma, \delta, \lambda)$. We will split the nodes in $\text{Free}_D$ which have more than one parent – this can be done without harm to the ADAG because the nodes that must not be split are in $\text{Fixed}_D$.

Let $w \in \text{Free}_D$ be a node with the set of parents $v_1, \ldots, v_m$ for an $m > 1$. Then we will define the ADAG $D' = (D', \Psi, \tilde{\Gamma}, \alpha, \beta', \gamma', \delta', \lambda')$.

We begin with the DAG $D'$. Take new nodes $w_1, \ldots, w_m$. Let $V' = \{w_1, \ldots, w_m\} \cup V \setminus \{w\}$. Let $F'$ be the same as $F$ but where each edge $(v_j, w, l)$ is replaced by $(v_j, w, l)$. And finally let $\mu'$ be the same as $\mu$ but with $\mu'(w_j) = \mu(w)$ for all $j \in \{1, \ldots, m\}$. Then we set $D' = (V', F', \mu')$.

For each predicate symbol $r \in \tilde{R}$ and each $i \in \mathbb{N}$ we again define $(w, r, i)^p = w_j$ if $\beta(w, r, i) = (v_j, r', i', \varphi)$. Similar to the previous constructions we define $\delta_Q = \delta(w) \cap Q$, $\delta_R = \delta(w) \cap \tilde{R}$, as well as $\delta_{\tilde{\Gamma}} = \delta(w) \cap \tilde{\Gamma}$ and

$$\delta'^i_R = \{ r \in \delta_R \mid (w, r, i)^p = w_j \text{ for an index } i \in \mathbb{N} \} \, .$$

Note that the new nodes will not be goals as there are no goals in $\text{Free}_D$, so we do not have to care for the witnesses of $\Gamma$-symbols. Now we can define $\delta'$, $\beta'$, $\gamma'$ and $\lambda'$ for the new nodes:

$$\delta'(w_j) = \delta_Q \cup \delta_R \cup \delta_{\tilde{\Gamma}} \quad \text{for all } j \in \{1, \ldots, m\} \, ,$$

$$\beta'((w, p, i)^p) = \beta(w, p, i) \quad \text{for all } p \in \delta(w), i \in \gamma^*(w, p) \, ,$$

$$\gamma'((w, p, i)^p) = \gamma(w, p, i) \quad \text{for all } p \in \delta(w), i \in \gamma^*(w, p) \, ,$$

$$\lambda'(c) = ((w, p, i)^p, p, i, j) \quad \text{if } \lambda(c) = (w, p, i, j) \, .$$

For all other nodes in $V \cap V'$, for all related registers, indices and constants, let $\delta' = \delta$, $\beta' = \beta$, $\gamma' = \gamma$ and $\lambda' = \lambda$. Then $D' = (D', \Psi, \tilde{\Gamma}, \alpha', \beta', \gamma', \delta', \lambda')$ is an ADAG, and $D'$ has an exponential number of goals, as we did not duplicate any goals.
If there is a node $w \in \text{Free}_\mathcal{D}$, we will apply this transformation on $\mathcal{D}'$, until we get an ADAG $\mathcal{D}''$ that still fulfills the restrictions mentioned in Theorem 4.12 but in which each node $w \in \text{Free}_\mathcal{D}''$ has at most one parent. Now by Lemma 4.3 we can assume that each node in $w \in \text{Free}_\mathcal{D}''$ has exactly one parent.

4.4.2 Shortening the Push Paths

Lemma 4.14 (Bounding Push Paths). Let $\mathcal{D}$ be an ADAG for a protocol $\mathcal{P}$ with the same assumptions and restrictions as in Theorem 4.12 as well as the restriction that each node in $\text{Free}_\mathcal{D}$ has only one parent. Then there exists an ADAG $\mathcal{D}'$ still fulfilling these restrictions, but where the number of nodes in $\text{Paths}_{\mathcal{D}'}$ is bounded double exponentially in terms of the size of the protocol.

Proof. Let $\mathcal{D} = (D, \Psi, \bar{\Gamma}, \alpha, \beta, \gamma, \delta, \lambda)$ be an ADAG as mentioned in the lemma. To prove this lemma, we define sections of push paths which will be special subpaths of push paths:

For each $[v_1, \ldots, v_j, \ldots, v_m] \in \text{Paths}_\mathcal{D}$ the set $\text{Sections}_\mathcal{D}$ shall contain each non-empty sequence $[v_i, \ldots, v_j]$ of maximal length, such that $\text{Goals}_\mathcal{D}(v_i) = \text{Goals}_\mathcal{D}(v_j)$ and that $v_j, \ldots, v_i \notin \text{Fixed}_\mathcal{D}$. Maximal length means that neither $[v_{j-1}, \ldots, v_j]$ nor $[v_j, \ldots, v_{j+1}]$ could be in $\text{Sections}_\mathcal{D}$.

Lemma 4.15 restricts the number of sections, i.e., the size of the set $\text{Sections}_\mathcal{D}$ is exponentially bounded. Then Lemma 4.16 shows that $\mathcal{D}$ can be modified such that each section has a length which is double exponentially bounded, without losing the restrictions in the lemma.

4.4.2.1 Bounding the Number of Sections

Lemma 4.15 (Bounding the Number of Sections). Let $\mathcal{D}$ be an ADAG fulfilling the restrictions mentioned in Lemma 4.14. Then the number of paths in $\text{Sections}_\mathcal{D}$ is exponentially bounded in terms of the size of the protocol.

Proof. Let $\mathcal{D}$ be an ADAG with the necessary restrictions. Examine the size of $\text{Sections}_\mathcal{D}$: First, Lemma 4.2 states that the size of $\text{Paths}_\mathcal{D}$ equals the number of goals in $\mathcal{D}$. We look at each push path $\vec{\sigma} = [v_1, \ldots, v_m] \in \text{Paths}_\mathcal{D}$, and we show that the number of sections that can be constructed from this path is bounded.

First, let $i_1, \ldots, i_l$ be the set of indices with $v_{i_j} \in \text{Fixed}_\mathcal{D}$ (and let $i_0 = 0$ as well as $i_{l+1} = m + 1$). This leads to the following set of subsequences of $\vec{\sigma}$, where the size of the set is equal to or less than $l + 1 \leq |\text{Fixed}_\mathcal{D}| + 1$, i.e., exponential:

$$\text{Paths}_\mathcal{D}(\vec{\sigma}) = \left\{ [v_{j+1}, \ldots, v_{j+l}] \mid j \in \{0, \ldots, l\} \right\} . \tag{4.50}$$

Now let $\vec{\omega} = [w_1, \ldots, w_m] \in \text{Paths}_\mathcal{D}(\vec{\sigma})$. Let $i_1, \ldots, i_l$ be all indices with $\text{Goals}_\mathcal{D}(w_j) \neq \text{Goals}_\mathcal{D}(w_{j+1})$, and so $\vec{\omega} \notin \text{Sections}_\mathcal{D}$. But we know $\text{Goals}_\mathcal{D}(w_{j+1}) \subseteq \text{Goals}_\mathcal{D}(w_j)$ because
of Lemma 4.14 and the fact that \( w_{j+1} \in \text{Free}_D \) has only one parent, i.e., each goal reachable from \( w_{j+1} \) is also reachable from \( w_j \). Therefore, \( l \) is bounded by \( |\text{Goals}_D(w_1)| \), which is bounded by the total number of goals. Now we can define the following set (with \( i_0 = 0 \) and \( i_{l+1} = m \)):

\[
\text{Sections}_D(\vec{w}) = \left\{ [w_{i+1}, \ldots, w_{j+1}] \mid j \in \{0, \ldots, l\} \right\} .
\] (4.51)

Then each sequence \( \vec{u} \in \text{Sections}_D(\vec{w}) \) is a section, as \( \vec{u} \) contains no nodes from \( \text{Fixed}_D \) and the set of reachable goals equals for all nodes in \( \vec{u} \). On the other hand, for each section \( \vec{u} \) we find a sequence \( \vec{u} \) which is a subsequence of a \( \beta \)-path \( \vec{v} \in \text{Paths}_D(w) \), so altogether we have:

\[
\text{Sections}_D = \left\{ \vec{u} \mid \vec{v} \in \text{Paths}_D(w), \vec{w} \in \text{Paths}_D(\vec{v}), \vec{u} \in \text{Sections}_D(\vec{w}) \right\} .
\] (4.52)

As for each \( \vec{v} \in \text{Paths}_D(w) \) we have a set \( \text{Paths}_D(\vec{v}) \) of exponential size, and because for each \( \vec{w} \in \text{Paths}_D(\vec{v}) \) we have a set \( \text{Sections}_D(\vec{w}) \) of exponential size, we altogether have an exponentially bounded set \( \text{Sections}_D \).

4.4.2.2 Shortening Sections

**Lemma 4.16 (Bounding the Length of Sections).** Let \( D \) be an ADAG fulfilling the restrictions mentioned in Lemma 4.14. Then we can construct an ADAG \( D' \) still meeting the restrictions, but where the length of paths in \( \text{Sections}_D \) is exponentially bounded in terms of the size of the protocol.

**Proof.** Let \( D = (D, \Psi, \bar{\Psi}, \alpha, \beta, \gamma, \delta, \lambda) \) be an ADAG as mentioned in the lemma. We will reduce the number of nodes in a section by a pumping method.

**Comparing Registers and Nodes**

Let \( \vec{v} = [v_0, v_1, \ldots, v_m] \in \text{Sections}_D \). We will compare each \( v_j \) with \( v_0 \) by a function \( \Delta_{v_0} \) which notes the differences in the labeling \( \delta \) and the registers \( \gamma \) between \( v_0 \) and \( v_j \). We will search for nodes \( v_1, v_2 \) with \( \Delta_{v_0}(v_1) = \Delta_{v_0}(v_2) \) and \( j_2 > j_1 \), and then we will replace the edge \( v_{j_1} \rightarrow v_{j_1} \) by an edge \( v_{j_1} \rightarrow v_{j_2} \).

As each node on the path may have a register with fresh anonymous constants, we cannot simply compare the values of the registers – we need to abstract from the concrete constants as explained below. Thus, we first define a function \( \Delta^* \) for comparing registers.

We need a fixed set of \( Z \) anonymous variables \( y_1, \ldots, y_Z \in Y \) and \( Z \) fresh variables \( y_1', \ldots, y_Z' \in Y' \). Then we define the following function on two ground registers:

\[
\Delta^*((c_1, \ldots, c_Z), (c_1', \ldots, c_Z')) = (\chi(c_1'), \ldots, \chi(c_Z'))
\] (4.53)

with a function \( \chi(c') = \begin{cases} y_j & \text{if } c' = c_j \text{ and } j = \min \left\{ j' \mid c' = c_{j'} \right\}, \\ y_j^* & \text{if } c' \notin \{ c_1, \ldots, c_Z \}. \end{cases} \)
To clarify this definition we will give some examples. The function compares each $c'_j$ with $c_j$: It uses $y_j$ as a variable for $c_j$ (and for all $j' > j$ with $c_{j'} = c_j$), and the fresh anonymous constants not occurring in $c_1, \ldots, c_2$ are mapped to the fresh variables $y_{j'}^\star$.

Note that the results may be ambiguous as we did not care for the ordering of the fresh variables $y_{j'}^\star$; in fact, we do not need these details as we only show a rough complexity bound. Therefore, the following results would be possible

\[
\Delta^*(\langle a, b, c, d \rangle, \langle b, c, c, d \rangle) = \langle y_2, y_3, y_4, y_4^\star \rangle, \quad (4.54)
\]
\[
\Delta^*(\langle a, b, c, c \rangle, \langle c, c, c, d \rangle) = \langle y_2, y_3, y_4^\star, y_4^\star \rangle, \quad (4.55)
\]
\[
\Delta^*(\langle a, b, c, d \rangle, \langle c, f, e, a \rangle) = \langle y_2^\star, y_3^\star, y_4 \rangle, \quad (4.56)
\]
\[
\Delta^*(\langle a, b, c, d \rangle, \langle e, f, c, a \rangle) = \langle y_4^\star, y_5^\star, y_4, y_4^\star \rangle. \quad (4.57)
\]

This enables us to compare the nodes $v' \in \mathcal{V}$ with the node $v_0$ which is the first node of $\bar{v}$, but first, just for one push symbol and one of the registers of $v_0$. Therefore, let $p \in \delta(v_0)$ and $i \in \gamma^*(v_0, p)$. Then we set:

\[
\Delta_{v_0}(v', p, i) = \left\{ (p', \Delta^* (\gamma(v_0, p, i), \gamma(v', p', i'))) \mid (v_0, p, i) \sim^* (v', p', i') \right\}. \quad (4.58)
\]

Now let $[(p_1, i_1), \ldots, (p_l, i_l)]$ be a sequence of all $p \in \delta(v_0)$ and $i \in \gamma^*(v_0, p)$. Then we define:

\[
\Delta_{v_0}(v') = (\delta(v'), \Delta_{v_0}(v', p_1, i_1), \ldots, \Delta_{v_0}(v', p_l, i_l)). \quad (4.59)
\]

We will count the possible values for $\Delta_{v_0}$: In the first component, we have a subset of $Q_f$, so there are $2^{|Q_f|}$ possible values. In each of the next $l$ components, we have a set of tuples as defined in (4.58). Each of these tuples consists of a push symbol and a register containing variables from two sets of size $Z$, so the size of each tuple is bounded by $|Q_f| \cdot (2Z)^2$.

Now analyze how many tuples can occur in $\Delta_{v_0}(v', p_1, i_1), \ldots, \Delta_{v_0}(v', p_l, i_l)$: Each node can contain no more registers than the number of goals reachable from that node, which is a consequence of Lemma 4.4. Hence, the total number of tuples in all $l$ sets is bounded by the total number of goals. This leads to the following bound for the possible values of $\Delta_{v_0}$:

\[
2^{|Q_f|} \cdot (|Q_f| \cdot (2Z)^2)^{|\text{Goals}_D|} \quad (4.60)
\]
\[
< 2^n \cdot (n \cdot (2n)^n)^{|\text{Goals}_D|} \quad \text{with } n = c_0 \cdot |P|^{c_0} \text{ for some } c_0, c_1 \in \mathbb{N} \quad (4.61)
\]
\[
\leq (2^n + 2^{n^2})^{|\text{Goals}_D|} \quad \text{for some } c_2 \in \mathbb{N} \quad (4.62)
\]
\[
< 2^n \cdot 2^{n^2} \cdot 2^{n^3} \quad \text{with } |\text{Goals}_D| \leq 2^n c_4 \text{ for some } c_4 \in \mathbb{N} \quad (4.63)
\]
\[
\leq 2^{2n^4} \quad \text{for some } c_4 \in \mathbb{N} \quad (4.64)
\]

Now we can define the transformation on the ADAG that bounds the length of a section $\bar{v} = [v_0, v_1, \ldots, v_m]$ to $2^{2n^4}$. If the length of a section is already less than $2^{2n^4}$, we have
nothing to do. Else, there have to be two nodes \(v_{j_1}, v_{j_2}\) with \(\Delta_{v_0}(v_{j_1}) = \Delta_{v_0}(v_{j_2})\) and \(j_1 < j_2\).

We will now replace the edge \(v_{j_1-1} \rightarrow v_{j_1}\) by \(v_{j_1-1} \rightarrow v_{j_2}\) and further modify the ADAG. Note that by repeating this procedure we are able to bound the length of the section \(\vec{v}\) to be less than \(2^{\alpha'\cdot 4}\). Then, by iterating over all sections, we reduce the number of nodes in each section.

**Transforming the ADAG to Shorten a Section**

We will define the term \(\text{DAG } D'\) based on \(D\), with the modified edge as explained above and without the nodes \(v_{j_1}, \ldots, v_{j_2-1}\) (and without their relevant descendants):

\[
V' = V \setminus \{ w \in V \mid v_{j_1} \rightarrow^* w, \text{ but not } v_{j_2} \rightarrow^* w \} ,
\]

\[
F' = (F \cap (V' \times V' \times \mathbb{N})) \cup (v_{j_1-1}, v_{j_2}, 1) .
\]

We now have to change the witness function for \(v_{j_2}\) to point to \(v_{j_1-1}\), and we have to modify some registers as the constants in the registers of \(v_{j_1}\) and \(v_{j_2}\) may differ. Each fresh anonymous constant \(c\) that is generated between \(v_0\) and \(v_{j_1-1}\) will be replaced by \(\hat{\chi}(c)\). This may mean that it is not changed at all if \(\hat{\chi}(c) = c\), but in this way we will construct \(\beta\)-paths which are valid for the registers of \(v_{j_2}\).

Note that in this step we may have removed an unbounded number of anonymous constants: All the constants that had been generated along the path from \(v_{j_1}\) to \(v_{j_2}\) are deleted if they do not occur in any register of \(v_{j_2}\).

From \(\Delta_{v_0}(v_{j_1}) = \Delta_{v_0}(v_{j_2})\) we know \(\delta(v_{j_1}) = \delta(v_{j_2})\). We also know that for each \(p \in \delta(v_{j_1})\) and each \(i \in \gamma^*(v_{j_1}, p)\) we have a matching index \(i_2 \in \gamma^*(v_{j_2}, p)\). For simplification we just assume that \(i_1 = i_2\); this is possible because the actual values of the indices \(i_1, i_2\) do not matter, so we could modify the values of \(\gamma^*(v_{j_2}, p)\) to match \(\gamma^*(v_{j_1}, p)\) without damaging the ADAG.

Now we know that for each register \(\kappa_{v_0}\) of \(v_0\), for each \(p \in \delta(v_{j_1})\) and each \(i \in \gamma^*(v_{j_1}, p)\) we have a register \(\kappa_{\Delta}\) with

\[
\kappa_{\Delta} = \Delta^*(\kappa_{v_0}, \gamma(v_{j_1}, p, i)) = \Delta^*(\kappa_{v_0}, \gamma(v_{j_2}, p, i)) .
\]

Let \(\chi_1^{-1}\) be a mapping from constants to variables which replaces each anonymous constant with the corresponding variable in the variable register \(\kappa_{\Delta}\), i.e., this mapping – extended on registers – would give \(\chi_1^{-1}(\gamma(v_{j_1}, p, i)) = \kappa_{\Delta}\). Let \(\chi_2\) be the mapping from variables to anonymous constants such that – extended on registers – the mapping \(\chi_2\) would give \(\chi_2(\kappa_{\Delta}) = \gamma(v_{j_2}, p, i)\). By (4.53) we know both \(\chi_1^{-1}\) and \(\chi_2\) exist and that when combining them to \(\chi_{(p,i)} = \chi_1^{-1} \circ \chi_2\) we have a function with \(\chi_{(p,i)}(\gamma(v_{j_1}, p, i)) = \gamma(v_{j_2}, p, i)\).

Let \(\hat{\chi}\) be the unification of all functions \(\chi_{(p,i)}\) for each \(p \in \delta(v_{j_1})\) and each \(i \in \gamma^*(v_{j_1}, p)\). This unification is a well-defined function as each constant in its domain is either fresh or in a register of \(v_0\), and \(\hat{\chi}\) is the identity on the latter ones. We will define the functions of the ADAG \(\mathcal{D}' = (D', \Psi, \hat{\Gamma}, \kappa, \beta', \gamma', \delta', \lambda')\), beginning with \(\delta'\), which is just the restriction of
4 Scaling Down ADAGs

Δ to V'. Now define the register and the witness function for all nodes v ∈ V', symbols p ∈ δ'(v), and indices i ∈ γ*(v, p) by

\[
\beta'(v, p, i) = \begin{cases} 
\beta(v, p, i), & \text{if } v \neq v_{j_2}, \\
\beta(v_{j_1}, p, i), & \text{if } v = v_{j_2}, 
\end{cases}
\]

\[
(4.68) \]

\[
\gamma'(v, p, i) = \begin{cases} 
\hat{\chi}(\gamma(v, p, i)), & \text{if } v \in \{v_0, \ldots, v_{j_2-1}\}, \\
\gamma(v, p, i), & \text{otherwise}.
\end{cases}
\]

\[
(4.69) \]

We can now define the freshness function – this looks a bit technical because we may have removed nodes and their registers, and because other registers may have been affected by \(\hat{\chi}\). Note that \(\lambda\) is not defined for constants \(c \in \text{dom}(\hat{\chi}) \setminus \text{range}(\hat{\chi})\), as they are replaced and removed from the ADAG.

\[
\lambda'(c) = \begin{cases} 
\lambda(c), & \text{if } c \notin \text{dom}(\hat{\chi}) \text{ and } \lambda(c) = (v, p, i, z) \text{ for a node } v \in V', \\
\lambda(\hat{\chi}^{-1}(c)), & \text{if } c \in \text{range} \hat{\chi}.
\end{cases}
\]

\[
(4.70) \]

Now \(D'\) is an ADAG still fulfilling all restrictions mentioned in Theorem 4.12, but with a shortened section. As explained above, we can repeat this procedure until each section has a bounded length.

**Proving the ADAG Conditions and Restrictions**

We will now prove that \(D'\) is still an ADAG. First, we modified the DAG, but we only connected two nodes with an edge that were connected before by a sequence of edges, so \(D'\) is still acyclic. We will now check conditions \(\text{ADAG i a)}\) to \(\text{ADAG iii d)\). As we did not manipulate \(I\)-symbols, there is nothing to show for \(\text{ADAG i a)}\) to \(\text{ADAG ii)\). Now let \(v \in V'\) and \(p \in \delta'(v, p)\). Then we have \(v \in V\) and \(p \in \delta(v, p)\) and one of the conditions \(\text{ADAG iii a)}\) to \(\text{ADAG iii d)\) holds for \(v\) and \(p\) in \(D\). Again, distinguish between these four cases:

(iii a) In this case we have nothing to show because \(v \in \text{Fixed}_D\).

(iii b) For the application of pop clauses note that when manipulation edges, we did replace the edge \(v_{j_1-1} \rightarrow v_{j_1}\) by \(v_{j_1-1} \rightarrow v_{j_2}\), but we know \(\delta(v_{j_1}) = \delta(v_{j_2})\), so \(\text{ADAG iii b)\) still holds.

(iii c) This case is the interesting one. Distinguish three cases:

- First, let \(v \notin \{v_0, v_1, \ldots, v_{j_2}\}\). Then nothing was changed for \(v\), so condition \(\text{ADAG iii c)\) still holds.

- Now let \(v = v_{j_2}\), so we connected \(v\) to a new parent node during the modification. We did not change the registers of \(v\), so the freshness function is still correct, but we have to show that the modified witness function is correct, i.e., for each register of \(v\) there is a compatible register of the new parent.


Let $i \in \gamma^*(v, p)$. Then we know that not only $p \in \delta(v)$, but also $p \in \delta(v_j)$ by equation (4.50). Hence, there was a witness $\beta(v_j, p, i)$ in $D$, and we can use it as a witness $\beta'(v, p, i)$.

Now for the embedding of the registers let $\beta(v_j, p, i) = (v', p', i', \varphi)$ (with $v' = v_{j-1}$), and let $\kappa, \kappa'$ be the registers in $\varphi$. Then we can start with the compatibility of the embeddings in $D$ and show:

\[
\begin{align*}
(\kappa, \kappa') & \mapsto (\gamma(v', p', i'), \gamma(v_j, p, i)), \tag{4.71} \\
\text{it follows} (\kappa, \kappa') & \mapsto (\hat{\chi}(v', p', i'), \hat{\chi}(\gamma(v_j, p, i))) \tag{4.72} \\
& = (\hat{\chi}(v', p', i'), \gamma(v_{j}, p, i)) \tag{4.73} \\
& = (\gamma'(v', p', i'), \gamma'(v_{j}, p, i)). \tag{4.74}
\end{align*}
\]

The first transformation is valid because $\hat{\chi}$ is a function that changes both single embeddings in the same way. The second step is valid by definition of $\hat{\chi}$ and the functions $\chi_{(p, i)}$, whereas the final step is valid by definition of $\gamma'$ (4.69).

Now we know that condition (ADAG iii) still holds for this case.

- Finally, we assume that $v \in \{v_0, v_1, \ldots, v_{j-1}\}$, so $v$’s registers may have been changed by $\chi$, and so may the freshness function.

As in case (ADAG iii c) we know the embeddings may have changed, but are still compatible, as we did change the register of the parent node of $v$ in the same way. This does even hold for $v = v_0$ as we do not change its registers (or that of its parent node) because by definition, $\text{dom}(\hat{\chi})$ only contains constants that are in none of the registers of $v_0$, i.e., fresh.

The freshness function was modified accordingly, i.e., when $c$ was replaced by $c'$, we have $\lambda'(c') = \lambda(c)$. Thus, condition (ADAG iii) also holds for this case.

(iii d) In this case we know that $v \in \text{Fixed}_D$, and also $v' \in \text{Fixed}_D$ for $\beta(v, I) = (v', p', i', \varphi)$, so we have nothing to show.

This proves that our construction yields an ADAG. But this ADAG also fulfills all restrictions we mentioned before: It is still simple; we only shortened a path, so each set of nodes contradicting the simplicity would also show that the ADAG we started with would not have been simple. It still has only one parent for each node in $\text{Free}_D$; we only changed the edge $v_{j-1} \rightarrow v_j$ to $v_{j-1} \rightarrow v_{j-1}$ and removed the node $v_{j-1}$. The set of goals was not manipulated, this also holds for the stage theory, so both are still bounded.

### 4.4.3 Shortening the Pop Paths

**Lemma 4.17 (Bounding Pop Paths).** Let $D$ is an ADAG for a protocol $P$, which fulfills the restrictions of Theorem 4.12 as well as the restrictions that each node in $\text{Free}_D$ has only one parent and that the number of nodes in the push paths is double exponentially bounded in terms of the size of
the protocol. Then there exists an ADAG $\mathcal{D}'$ fulfilling these restrictions, while the number of nodes in $\text{Paths}^\Delta_\mathcal{D}$ is double exponentially bounded as well.

Proof. We will prove this lemma by first shortening each single path and then showing that the sum over all nodes in all paths is double exponentially bounded.

Shortening a Pop Path

Let $\mathcal{D} = (\mathcal{D}, \Psi, \hat{\lambda}, \alpha, \beta, \gamma, \delta, \lambda)$ be such an ADAG with the $\text{DAG} D = (V, F, \mu)$, and let $\mathcal{D} = [v_1, \ldots, v_m] \in \text{Paths}^\Delta_\mathcal{D}$. By Lemma 4.2 we can assume that there are no push symbols in the labeling of the nodes of $\mathcal{D}$ and that for all nodes $v_j \in \mathcal{D}$ we have $\text{Goals}^\Delta_\mathcal{D}(v_j) = \emptyset$ (otherwise, $v_j$ would be in $\text{Paths}^\nabla_\mathcal{D}$).

Let $M = 2^{\Omega(|)}$. Then we know that if $m > M$, there are two nodes $v_{j_1}, v_{j_2}$ with $\beta(v_{j_1}) = \beta(v_{j_2})$ and $j_1 < j_2$. Like in Section 4.4.2.2, we will replace the edge $v_{j_1 - 1} \rightarrow v_{j_1}$ by $v_{j_1 - 1} \rightarrow v_{j_2}$. However, we do not have to remove $v_{j_1}, \ldots, v_{j_2 - 1}$ (and the relevant descendants) explicitly, as shown below.

Formally we define $\mathcal{D}'$ based on $\mathcal{D}$ just by replacing the set of edges $F$ with the following set $F'$:

$$F' = (F \setminus (v_{j_1 - 1}, v_{j_1}, 1)) \cup (v_{j_1 - 1}, v_{j_2}, 1).$$

(4.75)

This clearly leads to an ADAG: The only condition affected may be $(\text{ADAG iii} \mathcal{D})$, but it still holds because of $\beta(v_{j_1}) = \beta(v_{j_2})$.

Now we can use Lemma 4.3 which shows that each node $w$ which was on a pop path $\mathcal{D} = [v_1, \ldots, v_j, \ldots, w, \ldots] \in \text{Paths}^\Delta_\mathcal{D}$ and that is no descendant of $v_j$ can be removed, as it is no longer a descendant of any $\alpha(t)$ for any $t \in T_p$.

Then we get an ADAG with the necessary restrictions: The only thing we have to check is that each node in Free$\mathcal{D}$ has only one parent. In Free$\mathcal{D}'$, there was exactly one node with more than one parent: The node $v_{j_2}$ was a descendant of $v_{j_2 - 1}$ and $v_{j_1 - 1}$. But by applying Lemma 4.3 we removed $v_{j_2 - 1}$, so this restriction holds again. We can repeat this procedure until the length of each pop paths is bounded by $M$.

Bounding the Number of Pop Paths

We will show that in an ADAG $\mathcal{D}$ with a bounded length of the pop paths as described above, there is at most a double exponential number of pop paths, which proves our statement that there are at most double exponentially many nodes in the set $\text{Paths}^\Delta_\mathcal{D}$.

Let $w \in \text{Fixed}_\mathcal{D} \cup \text{Paths}^\nabla_\mathcal{D}$ be a node. Let $t_w$ be the tree rooted in $w$ consisting of all paths $[v, \ldots] \in \text{Paths}^\Delta_\mathcal{D}$ with $w \rightarrow v$; we know this is a tree because the nodes in $\text{Paths}^\Delta_\mathcal{D}$ have exactly one parent. The grade of each node in this tree is bounded by the maximal degree of the signature, and by assumption about the length of the pop paths the depth of the tree is at most exponential. Hence, we know that there are at most double exponentially many nodes in $t_w$. 

As each path in $\text{Paths}_{D}$ belongs to a tree $t_{w}$, we can bound the number of nodes in $\text{Paths}_{D}$ by the following expression which is double exponential in terms of the size of the protocol:

$$
\sum_{w \in \text{Fixed}_{D}} | \{ v \in V \mid v \in t_{w} \} |.
$$  \hspace{1cm} (4.76)

This concludes the proof of this lemma and shows that the total number of nodes in an ADAG can be bounded to be double exponential.

### 4.4.4 Reducing the ADAG Components

**Lemma 4.18 (Bounding the ADAG).** Let $D$ be an ADAG for a protocol $P$ with a double exponentially bounded number of nodes in terms of the size of $P$, and with the bounds assumed in Theorem 4.12. Then we can assume that the whole ADAG with all of its components has a double exponentially bounded size.

**Proof.** Let $D = (D, \Psi, \tilde{\Gamma}, \alpha, \beta, \gamma, \delta, \lambda)$ be an ADAG with a double exponentially bounded number of nodes. We will now show bounds for all components or reduce them, leading to an ADAG that can be described in a double exponentially bounded amount of space.

We will use the assumption that the indices of registers $i \in \gamma^*(v, p)$ for each node $v$ and each symbol $p \in \delta(v)$ do not exceed an exponential size: By Lemma 4.2, the number of registers in a node $v$, i.e., $\sum_{p \in \delta(v)} | \gamma^*(v, p) |$, is bounded by the number of goals that $v$ leads to. As an upper bound, each node could lead to exponentially many goals and thus have exponentially many registers. As the values of the indices do not matter, we can rename each register such that these indices are smaller than a certain bound $M \in O(2^{|P|})$ where $O$ is the Landau notation.

Now we will turn to the components of the ADAG. As the set of nodes is bounded, $D$ is bounded as well. By the assumption transferred from Theorem 4.12, the stage theory $\Psi$ is exponentially bounded.

Now look at the set of anonymous constants $\tilde{\Gamma}$: We know we have at most $M$ registers at one node. Each of the registers could contain $Z$ fresh anonymous constants. Thus, the total number of constants occurring in the ADAG is also double exponentially bounded, and we can set $\tilde{\Gamma}' \subseteq \tilde{\Gamma}$ to contain all anonymous constants in all registers as well as all constants used in the labeling of the DAG. Then, $\tilde{\Gamma}'$ is double exponentially bounded.

Finally, look the functions of $D$:

\[
\begin{align*}
\alpha &: \text{sub}(T_{P}) \rightarrow V, \\
\beta &: V \times Q_{f} \times \mathbb{N} \rightarrow V \times \mathbb{N} \times \Psi, \\
\gamma &: V \times Q_{f} \times \mathbb{N} \rightarrow \tilde{\Gamma}'^Z, \\
\delta &: V \rightarrow 2^{Q_{f}}, \\
\lambda &: \tilde{\Gamma}' \rightarrow V \times Q_{f} \times \mathbb{N} \times \{1, \ldots, Z\}.
\end{align*}
\]  \hspace{1cm} (4.77) \hspace{1cm} (4.78) \hspace{1cm} (4.79) \hspace{1cm} (4.80) \hspace{1cm} (4.81)
We will analyze the size of each set occurring in the list of functions above:

- The sets \( T_P, Q_f \), and \( Z \) have polynomial size in terms of the size of the protocol.
- The sets \( \text{sub}(T_P), 2^{|Q_f|} \), and \( \Psi \) are of exponential size.
- The cardinality of \( V, \hat{\Gamma}', \) and \( \hat{\Gamma}^Z \) is double exponential.
- Finally, each occurrence of \( N \) can be substituted by \( M \), which is double exponential.

Now we will use the fact that for finite sets \( A \) and \( B \), the description of a function \( f: A \rightarrow B \) can be done in space \( O(|A| \cdot \log |B|) \) (with Landau notation \( O \)) by taking functions \( \mu_A: \{1, \ldots, |A|\} \rightarrow A \) and \( \mu_B: B \rightarrow \{1, \ldots, |B|\} \) and storing the \( |A| \)-tuple
\[
\left( \mu_B(f(\mu_A(1))), \ldots, \mu_B(f(\mu_A(|A|))) \right). \tag{4.82}
\]

Thus, all functions and hence the whole ADAG can be described in space that is at most double exponential.

4.5 Proof of the Main Result

**Proof of the Main Theorem** Let \( P = (P, \Phi) \) be a protocol. Theorems allow us to develop the following algorithm:

**Algorithm 4.19.** Let \( P = (P, \Phi) \) be a protocol. Guess an execution scheme \( \pi \) and a stage mapping \( f \) for \( P \). Construct the instance \( \hat{\Phi}_f \) of \( \Phi \) of exponential size as shown in Definition 4.7. Take a double exponential set \( \hat{\Gamma} \) of anonymous constants. Then guess a \( \text{DAG} \) \( D = (V, F, \mu) \) of double exponential size which uses only constants from \( \hat{\Gamma} \). Guess functions \( \alpha, \beta, \gamma, \delta, \lambda \) of appropriate type. Then verify if the \( \text{ADAG} \) conditions (\( \text{ADAG i} \)) to (\( \text{ADAG iii d} \)) hold for \( D = (D, \Psi, \hat{\Gamma}, \alpha, \beta, \gamma, \delta, \lambda) \), the protocol \( P \) and (\( \pi, f \)). If that is the case, return »insecure«, otherwise return »secure«.

Our algorithm decides in nondeterministic double exponential time if there is an attack by constructing and guessing all necessary components in nondeterministic double exponential time and then checking the properties of an ADAG. The latter can be done by checking (\( \text{ADAG i} \)) – this is possible in less than \( O(|\beta|) \) steps – and then iterating over each node, checking conditions (\( \text{ADAG ii} \)) and (\( \text{ADAG iii} \)) for each node – this can mostly be done locally in that node and its direct ancestors and descendants, the most complex thing to check is the embedding in the case (\( \text{ADAG iii d} \)), but this only affects an exponential number of nodes, i.e., the goals.

This concludes the proof of our main theorem: We can check nondeterministically in double exponential time if there is an ADAG for a given protocol, i.e., if there is an attack on the protocol.
5 Conclusion

The goal of this thesis was to extend the model of selecting theories used to analyze recursive cryptographic protocols; the extension now allows us to generate fresh tokens like nonces or keys. Our main theorem shows that security of protocols in this model is decidable with respect to a bounded number of sessions.

To obtain this result we extended the formalism and protocol model introduced in [Tru05a, Tru05b]: First, the signature of the protocol model was joined with an infinite set of constants like in [KW04], then we modified the notion of ADAGs to cope with the generation of fresh tokens and unbounded finite subsets of infinite signatures.

We have shown that in this model, security is equivalent to the absence of an ADAG, and so we were able to prove decidability by bounding the size of ADAGS. To do so, we described a method to scaled down any ADAG until its size is at most double exponential in terms of the size of the protocol.

Extensions and Open Problems

A major disadvantage of this model is the necessity for simple push clauses and simple left-hand-sides in send clauses – this can probably be fixed by modifying the proof. Another problem is to find a tighter complexity bound for the decision problem.

In addition, it would be interesting if anonymous constants could be used as keys in this model. This is not trivial as the definition of an ADAG relies on the bounded number of keys occurring in a protocol. Furthermore, allowing complex keys or algebraic properties of XOR leads to undecidability in the general case [KT06]; but for restricted classes of protocols it is possible to model properties of XOR while still maintaining decidability [KT06]. A future direction could be to combine the approach of [KT06] with anonymous constants.

Acknowledgment

I would like to thank the people who helped me to write this thesis. First of all, I owe thanks to my supervisors, Prof. Dr. Thomas Wilke and PD Dr. Ralf Küsters, who helped me with their stream of ideas, supported and challenged me. I would also like to thank Tomasz Truderung, on whose work this thesis is based – the fruitful discussions about his approach helped a great deal. Detlef Kähler and Max Tuengerthal as well as my family deserve special credit for their support and encouragement. Last but by no means least, very special thanks go to Claudia for her love and unconditional support.
List of Figures

1.1 Decidability of Secrecy with Respect to Different Restrictions .............. 8
2.1 The Run of a Protocol ........................................... 19
3.1 The Theory of a Protocol $\Phi_P$ ........................................... 29
3.2 Stages of a Protocol .................................................. 37
3.3 Illustration for Part 1 of the Proof of Lemma 3.11 (Stage Theory) ........ 41
3.4 Graph of an Attack (ADAG) .......................................... 45
3.5 ADAG Conditions (ADAG iii a) to (ADAG iii d) .............................. 47
4.1 Illustration 1 for the Proof of Theorem 4.8 (Simple ADAGs) .............. 62
4.2 Illustration 2 for the Proof of Theorem 4.8 (Simple ADAGs) .............. 67
# Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-Fact</td>
<td>16</td>
</tr>
<tr>
<td>Access Relation</td>
<td>30</td>
</tr>
<tr>
<td>ADAG</td>
<td>45</td>
</tr>
<tr>
<td>Simple</td>
<td>58</td>
</tr>
<tr>
<td>Anonymous Constant</td>
<td>14</td>
</tr>
<tr>
<td>Anonymous Substitution</td>
<td>14</td>
</tr>
<tr>
<td>Anonymous Variable</td>
<td>14</td>
</tr>
<tr>
<td>Assumption</td>
<td>16</td>
</tr>
<tr>
<td>Atomic Formula</td>
<td>16</td>
</tr>
<tr>
<td>Attack</td>
<td>21</td>
</tr>
<tr>
<td>$\beta$-Edge</td>
<td>52</td>
</tr>
<tr>
<td>$\beta$-Path</td>
<td>52</td>
</tr>
<tr>
<td>C-Fact</td>
<td>16</td>
</tr>
<tr>
<td>Challenge</td>
<td>23</td>
</tr>
<tr>
<td>Clause</td>
<td>16</td>
</tr>
<tr>
<td>Compatibility</td>
<td></td>
</tr>
<tr>
<td>Embeddings and Registers</td>
<td>44</td>
</tr>
<tr>
<td>Conclusion</td>
<td>16</td>
</tr>
<tr>
<td>Constant</td>
<td>14</td>
</tr>
<tr>
<td>Anonymous Constant</td>
<td>14</td>
</tr>
<tr>
<td>DAG</td>
<td>44</td>
</tr>
<tr>
<td>DAG of the Attack</td>
<td>45</td>
</tr>
<tr>
<td>Dolev-Yao Model</td>
<td>6</td>
</tr>
<tr>
<td>Edge</td>
<td>44</td>
</tr>
<tr>
<td>$\beta$-Edge</td>
<td>52</td>
</tr>
<tr>
<td>Embedding</td>
<td>44</td>
</tr>
<tr>
<td>Compatibility</td>
<td>44</td>
</tr>
<tr>
<td>Embedding Function</td>
<td>45</td>
</tr>
<tr>
<td>Execution Scheme</td>
<td>21</td>
</tr>
<tr>
<td>Fact</td>
<td>16</td>
</tr>
<tr>
<td>A-Fact</td>
<td>16</td>
</tr>
<tr>
<td>C-Fact</td>
<td>16</td>
</tr>
<tr>
<td>Fresh Token</td>
<td>2</td>
</tr>
<tr>
<td>Fresh Variable</td>
<td>14</td>
</tr>
<tr>
<td>Freshness Function</td>
<td>45</td>
</tr>
<tr>
<td>Goal</td>
<td>53</td>
</tr>
<tr>
<td>Horn Clause</td>
<td>16</td>
</tr>
<tr>
<td>Usage and History</td>
<td>10</td>
</tr>
<tr>
<td>Horn Proof</td>
<td>16</td>
</tr>
<tr>
<td>Normal</td>
<td>40</td>
</tr>
<tr>
<td>Horn Theory</td>
<td>16</td>
</tr>
<tr>
<td>Index</td>
<td>47</td>
</tr>
<tr>
<td>Instance</td>
<td>43</td>
</tr>
<tr>
<td>Intruder</td>
<td>20</td>
</tr>
<tr>
<td>Intruder Theory</td>
<td>20</td>
</tr>
<tr>
<td>Key</td>
<td>17</td>
</tr>
<tr>
<td>Labeling Function</td>
<td>45</td>
</tr>
<tr>
<td>Message</td>
<td>17</td>
</tr>
<tr>
<td>Needham-Schroeder Protocol</td>
<td>6</td>
</tr>
<tr>
<td>Threat Model</td>
<td>6</td>
</tr>
<tr>
<td>Node</td>
<td>44</td>
</tr>
<tr>
<td>Fixed Node</td>
<td>53</td>
</tr>
<tr>
<td>Free</td>
<td>76</td>
</tr>
<tr>
<td>Goal</td>
<td>53</td>
</tr>
<tr>
<td>Static Node</td>
<td>53</td>
</tr>
<tr>
<td>Nonce</td>
<td>2</td>
</tr>
<tr>
<td>Normal Horn Proof</td>
<td>40</td>
</tr>
</tbody>
</table>
Bibliography


Ralf Küsters and Tomasz Truderung. On the Automatic Analysis of Recursive Security Protocols with XOR. To be Published, Jul 2006.


Erklärung


Kiel, den 17. Juli 2006

Klaas Ole Kürtz