Authentication logics (such as BAN and GNY) offer a way to reason about the knowledge of principals in a security protocol. It is explained how such logics work and what their caveats are.

Chapter 4

Authentication Logics

Formal analysis of security protocols requires one to reason about the knowledge of the participants (principals) in the protocol. Critical for a security protocol is that it should not only guarantee that certain information is communicated, but also that certain other information is *not* communicated. For example, external observers should typically not be able to infer session keys which are exchanged in a security protocol.

BAN logic [BAN89b, BAN89a][1], introduced by Burrows, Abadi and Needham, is an epistemic logic[2] crafted for analyzing security protocols. It models at an abstract level the knowledge of the principals in a protocol. The principals are supposed to have only polynomially many computational resources. It was the first logic of its kind, and has had a tremendous influence on protocol analysis: it has helped revealing weaknesses in known protocols, and many logics are based on it, among others GNY logic. We refer to BAN logic and all the variations of it as *authentication logics*. This is not to say that there has been no criticism of BAN logic. For one thing, a *full* semantics is lacking, and many attempts have been made to fix this problem [AT91, GNY90, KW94, WK96, vO93, SvO94, SvO96, Dek00]. Moreover, the logic fails to detect some very obvious protocol flaws [Nes90].

The general consensus about BAN-descendant logics appears to be that these logics are computationally sound (detected protocol flaws are indeed flaws), but certainly not computationally complete (they may fail to detect certain protocol flaws). Recent work includes attempts to bridge the gap between the formal (i.e., BAN-descendant) approach and the computational approach to security logics [AR02], and attempts to obtain completeness results for BAN-descendant logics in a kind of Kripke-style semantics [CD05a, CD05b]. In the

[1] See Appendix A.1 for a taxonomy of the papers presenting the BAN logic.

[2] For a thorough treatment of epistemic logic, consult [FHMV95, MvdH95].

Multi-Agent Systems world, BAN logic has been widely used (see for example, [AvdHdV01]).

Authentication logics play an important role in this thesis. In Chapter 5, we will prove that BAN logic is not 'sound', due to an inference rule that tries to model cryptographic hash functions. In Chapter 6, we will extend another authentication logic, GNY logic, in such a way that our protocols can be analyzed using GNY logic, which we do in Chapter 9. In the current chapter, we will explain the basics of authentication logics.[3] Where we write formulae, this will be formulae in GNY logic (which is summarized in Appendix B).

4.1 The Goals of an Authentication Logic

The main idea of authentication logics is that we model for each principal, what he knows, and what he can derive. What principals know includes what principals know about the knowledge of other principals: authentication logics are a special kind of *epistemic logic*. The knowledge of principals is modeled by explicitly stating what inference capabilities principals have, what principals know a priori, and what principals observe during a protocol run. The total knowledge of a specific principal is the *logical closure* of all capable inferences over the knowledge he has so far: A principal may know X a priori, infer from this Y, and if Z is inferrable from Y, the agent by definition also knows Z (and so on).

In most authentication logics, the malicious adversary model is used, which means that it is assumed that an adversary can overhear and intercept every message sent, and that an attacker can send any message he can synthesize. Message synthesis of the attacker is however limited to what is cryptographically feasible, i.e., the attacker has only polynomial computing power and cannot perform brute-force analysis to find secret keys [DY83]. (See also Section 2.6.)

Thus, principals are *logically omniscient* in the sense that they may apply some given inference rules unboundedly often. However, principals are not 'mathematically omniscient', i.e., they are not capable of all valid inferences.[4]

Proving that security protocols meet their specification generally involves proving three properties of a protocol:

1. the participating principals learn what they should learn,
2. what the participating principals learn is indeed true, and
3. no principal learns facts he ought not to know.

Thus, a correctness proof requires both a proof of things that are learned, and things that are *not* learned in course of a protocol run. Authentication

[3] For alternative introductions, consult [GSG99] and [SC01].

[4] In particular, principals cannot perform brute-force attacks on cryptographic primitives.

assumptions Alice knows Bob's public key, and Bob knows his own secret key.

the protocol itself Alice chooses a random number and sends it to Bob. Bob signs this number and sends it back to Alice.

claims After Alice receives Bob's message, she knows Bob received and read her message containing the random number.

FIGURE 4.1: The signing parrot protocol, plain description

logics generally focus on the learning part, and less if at all on the not-learning part. If an analysis of a protocol using an authentication logic does not expose a flaw, this means that properties 1 and 2 are not violated, of course assuming that the logic itself is 'correct'. In Chapter 6, we will extend GNY in such a way that property 3 is also addressed.

The specification of a protocol consists of the the following information:

Assumptions A description of the situation before the protocol is executed: who are the players involved (the *principals*), what is the knowledge of the principals, the reasoning powers of the principals, and the knowledge and reasoning powers of those who do not participate in the protocol.

The Protocol Itself A description of the messages that are exchanged in the protocol: how they are constructed, by whom, and when they are sent.

Claims A description of what the protocol supposedly provides: in what way the knowledge of the principals has changed since the beginning of the protocol, and in what way the knowledge of principals, including external observers, has *not* changed since the beginning of the protocol.

For example, consider the protocol shown in Figure 4.1[5]. We will use this protocol in the rest of this chapter in our examples.

4.2 The Taxonomy of Any Authentication Logic

In authentication logics, messages are not modeled as bit strings (as they are in the 'real world'), but as formulae. The principals in a protocol send one another formulae, and privately possess formulae. When a principal possesses a formula this just means he has it stored in his memory. Principals can construct (synthesize) new formulae by applying certain given methods. For example, one such method is symmetric encryption: if a principal possesses a message M and a key K, he can construct the message $\{M\}_K$, the symmetric encryption of M under K.

[5] This trivial protocol does not appear to have a name yet. For easier discussion, we will baptize it the *signing parrot protocol*. This protocol falls in the category often referred to as "Don't try this at home!" Bob's behavior of signing any message he sees, is very unwise.

What happens in the protocol however, *does* have a name. In strand-space terminology, it is called *incomping authentication* [Gut01, Gut02].

Similarly, principals can analyze (deconstruct) messages: if a player possesses $\{M\}_K$ and K, then he can decrypt $\{M\}_K$ and infer M, so he can add M to his 'possessions'. If on the other hand a principal possesses $\{M\}_K$ but not the decryption key K, it is (of course) not possible to infer M. In this way, certain parts of the communication may be hidden from some principals or external observers.

An authentication logic essentially consists of (at least) four parts:

1. A **Formal Language** that describes what *formulae* exist (i.e., what kind of messages can be exchanged). Moreover, the language describes what kinds of assertions can be made. Assertions typically relate formulae to principals. The formal language of GNY logic is summarized in Appendix B.1.

   The expressive power of the language directly influences the class of messages that can be expressed in the logic. For example, if there is no notation accommodated for asymmetric encryption, the logic cannot reason about it. A similar point also holds for statements: if for example the language does not distinguish between possessing a formula and believing the formula, one has to assume a principal believes all formulae he possesses.

2. A **Protocol Idealization Method** that describes how to translate a protocol description into the formal language of the logic. This results in a protocol description in the form of an ordered list of *send statements* $S_1, \cdots, S_n$, each S_i in the form $P \to Q\colon X$ (read: 'P sends Q message X') where P and Q are principals ($P \neq Q$) and X is a formula in the formal language of the logic.

   A protocol idealization method typically omits implementation details and tries to focus on the beliefs that are to be conveyed in the message. Protocol idealization is a manual task performed by humans.

3. A **Protocol Parser** which is an algorithm that translates an idealized protocol into an ordered list of *step transitions*. This list is the basis for further analysis of the protocol.

   Most protocol parsers are very trivial, but sometimes the parser does something that could be considered a kind of advanced annotation of the protocol.[6]

4. A **List of Inference Rules** or **Logical Postulates** which defines what a principal can do to construct and analyze formulae. In the following generic presentation of an inference rule with name N, $X_1, X_2, \cdots X_n$, $Y_1, Y_2, \cdots Y_m$ represent statements in the formal language of the logic.

$$\textbf{N} \qquad \frac{X_1, X_2, \cdots X_n}{Y_1, Y_2, \cdots Y_m}$$

[6] Our notion of a protocol parser is a generalization of the protocol parser notion described in [GNY90].

assumptions $A \ni +K, \quad A \models \overset{+K}{\mapsto} B, \quad B \ni -K, \quad A \ni N, \quad A \models \sharp(N)$

the protocol itself 1. $A \to B \colon N$
2. $B \to A \colon \{N\}_{-K}$

claims $A \models B \ni N$

FIGURE 4.2: GNY idealization of the signing parrot protocol. Alice and Bob are denoted A and B. Bob's public key is $+K$ and his private key is $-K$. The randomly chosen number is denoted N. For a summary of the formal language of GNY logic which is used here, consult Appendix B

> If $X_1, X_2, \cdots X_n$ all hold, then Y_1 and Y_2 up to Y_m may all be inferred.
>
> The inference rules of GNY logic are summarized in Appendix B.2.
>
> The list of inference rules is typically hand-crafted, small, and often fully given. The set of inference rules should be constructed in such a way that all notions attributed to elements of the language are expressed in the inference rules. For example, if it is possible to formulate a statement essentially saying 'principal P knows X and Y', then there should be inference rules accommodating the inference of 'principal P knows X' and 'principal P knows Y' from *the original sentence*.

To illustrate what the formal language and an idealized protocol look like, the GNY idealization of the signing parrot protocol is shown in Figure 4.2. Appendix B lists the formal language of GNY logic that is used in the figure.

The formal language of authentication logics often has a rather limited expressive power. In particular, explicit negations and disjunctions are oddities (with the notable exception of SVO logic [SvO94]). Implicit negations can however be found easily: most formal languages have a construct denoting that some cryptographic key K is *only* known to two (explicitly named) principals. In BAN and GNY logic, which will be explained later, this is the construct $P \overset{K}{\leftrightarrow} Q$. This of course implies that other principals do *not* know the key K. However, this very same construct also suggests that none of the two named principals would disclose the key K, and it remains the question whether this is realistic. And while disjunctions are not facilitated by the formal language, there are often inference rules which resemble disjunction elimination[7].

Of course, with poor support of disjunctions and negations, these logics can hardly if at all model protocols which have conditional branching: where whether some protocol steps are executed depends on the outcome of previous protocol steps. If one wants to analyze such a protocol using an authentication logic, one has to do this 'outside of the logic', that is: rely on natural language and while doing so, remain precise and concentrated.

[7] For an example of such an inference rule, look at rule **I3** of the GNY logic, shown in Appendix B on page 186. The $*$ sign in the first condition denotes that P did not send the message, and $P \overset{S}{\leftrightarrow} Q$ denotes that only P and Q know S. From this it is inferred that Q sent the message.

$$
\begin{gathered}
[A \ni +K, \quad A \mid\!\equiv \overset{+K}{\mapsto} B, \quad B \ni -K, \quad A \ni N, \quad A \mid\!\equiv \sharp(N)] \\
(A \rightarrow B \colon N) \\
[B \lhd *N] \\
(B \rightarrow A \colon \{N\}_{-K}) \\
[A \lhd *\{N\}_{-K}, \quad A \lhd \{N\}_{-K}, \quad A \ni \{N\}_{-K}, \\
A \ni H(N), \quad A \mid\!\equiv \phi(N), \quad A \mid\!\equiv B \mid\!\sim N, \quad A \mid\!\equiv B \ni N]
\end{gathered}
$$

FIGURE 4.3: GNY annotation ('correctness proof') of the signing parrot protocol. The assertions in the last postcondition are obtained by application of inference rules **T1**, **P1**, **P4**, **R6**, **I4** and **I6** (in that order). The very last assertion is equal to the claim of the protocol.

4.3 Using an Authentication Logic

When one analyzes a protocol using an authentication logic, one searches for a *legal annotation* of the protocol. A legal annotation is an annotation with some special properties. First, let us explain what an annotation is. An *annotation* of a protocol $S_1, \cdots, S_n$ is roughly something like a transcription in Hoare logic [Hoa69]:

$$[\text{assumptions}]\; S_1\; [\text{assertion } 1]\; \cdots\; [\text{assertion } n-1]\; S_n\; [\text{conclusions}]$$

An assertion is a (comma-separated) list of assertions in the formal language, interpreted as a conjunction. Obviously, the assumptions and the conclusions are a special type of assertion.

The protocol parser provides a list of *step transitions*. A step transition has the form

$$[\text{precondition}]\; (P \rightarrow Q : X)\; [\text{postcondition}]$$

For many logics, including BAN and GNY, the precondition is of the form Y, and the postcondition is of the form $Y, Q \lhd X$, which essentially means that whatever was true before the protocol step remains true afterward, and principal Q observes what he is told, namely X. Moreover, the assertion $Q \lhd X$ may be inserted directly after the protocol step. The protocol step itself is the justification for this assumption[8].

Note that the protocol parser does not enforce that the sending party is actually capable of sending the message (i.e., $P \ni X$ is not a formal precondition). In particular, the protocol assumption $B \ni -K$ remains unused in the analysis,

[8] The GNY protocol parser sometimes inserts the not-originated-here sign ($*$) into the inserted assertion. The rules for adding this sign are somewhat complicated, and will for simplicity not be explained in this thesis. These rules can be found in [GNY90, Section 5].

The not-originated-here sign ($*$) rougly means, that the principal who receives a message $*X$, has not previously sent a message X.

1	$A \ni +K$			$(B \rightarrow A\colon \{N\}_{-K})$	
2	$A \mid\equiv \overset{+K}{\mapsto} B$		7	$A \triangleleft *\{N\}_{-K}$	[2]
3	$B \ni -K$		8	$A \triangleleft \{N\}_{-K}$	**T1**(7)
4	$A \ni N$		9	$A \ni \{N\}_{-K}$	**P1**(8)
5	$A \mid\equiv \sharp(N)$		10	$A \ni H(N)$	**P4**(4)
	$(A \rightarrow B\colon N)$		11	$A \mid\equiv \phi(N)$	**R6**(10)
6	$B \triangleleft *N$	[1]	12	$A \mid\equiv B \mid\sim N$	**I4**(8, 1, 2, 11)
			13	$A \mid\equiv B \ni N$	**I6**(12, 5)

FIGURE 4.4: Heavy GNY annotation of the signing parrot protocol. After each assertion which is not an assumption, a justification is placed. This justification is either the result of a communication step inserted by the protocol parser, denoted with the protocol step between [square brackets], or the name of the inference rule applied, together with the statement numbers of its satisfied preconditions.

while it is neccessary since otherwise B cannot send the message $\{N\}_{-K}$. This issue is addressed further in Section 6.2.2.

To create a *legal annotation* of a protocol is to weld all step transitions of a protocol together such that:

- the precondition of the first step contains only the protocol assumptions;
- the postcondition of the last protocol step contains the protocol claims;
- except for the protocol assumptions and assertions added by the protocol parser, any assertion is derivable (by means of the inference rules) from its prefix[9].

We will use the final requirement, derivability, in a rather strict sense: every statement is obtainable from its prefix by application of at most one inference rule. This does not narrow the class of protocols for which legal annotations can be found, while at the same time it makes it easy to verify whether an annotation is legal. Moreover, we allow repetitions of assertions to be omitted for ease of reading. An example of a legal annotation of a protocol is shown in Figure 4.3.

This type of annotation can be difficult to read and interpret, and therefore in this thesis, we will be a bit more explicit. We will write every statement on an individual line, together with a line number and the applied inference rules for easy reference. An example of such a 'heavy annotation' is shown in Figure 4.4.

It should be noted that this type of logic is monotonic within a single protocol run: all assertions are stable, i.e., once true they remain true (for the time of the protocol run). To prove that a protocol is correct with repsect to a particular claim to give a (constructive) proof of the protocol claim by means of a (heavy) annotation.

[9] i.e., the protocol annotation before ('left of') the statement in question.

4.4 The BAN Logic Debate

Authentication logics provide a very *intuitive* means of analyzing security protocols. Whether the approach is also *accurate* has been subject of a very extended debate. The main criticisms of BAN logic regard the following properties of the logic:

1. the semantics,
2. the notion of belief,
3. the protocol idealization method,
4. the honesty assumption, and
5. the incompleteness.

Appendix A.2 discusses these criticisms in more detail. All existing criticisms of authentication logics might suggest that the approach is rather worthless, but on proper inspection the host of critiques deserves another interpretation: The way of reasoning in authentication logics is highly valuable, and that is why so much effort has been taken to *improve* the original BAN logic. In this thesis, the constructive contribution to authentication logics can be found in Chapter 6.

Our opinion on authentication logics is that the general approach is simply wonderful, while the operationalization of the approach is rather troublesome. In particular, authentication logics *in general* should not be blamed for problems with *early, individual instances* of authentication logics, such as BAN logic. The methodology is simple, intuitive and powerful. Therefore, we feel authentication logics deserve to be one of the basic tools for protocol analysis (next to other methodologies such as strand spaces and the computational approach).

We are not blind to the shortcomings of BAN logic and its descendants. In Chapter 5, we even prove that BAN logic is not 'sound', and Section 6.1 is not particularly praising, either.

4.5 Conclusion

In this chapter, we have briefly explained how an authentication logic works. The role of the formal language, the inference rules, the protocol parser, assertions, annotations and legal annotations have been explained. We have introduced the concept of a *heavy annotation*, which is a legal annotation that is easy to verify.

In the next chapter (Chapter 5), we will prove that BAN logic is not 'sound', and in Chapter 6 we will extend GNY logic in such a way that our protocols can be analyzed in Chapter 9.

For further reading on authentication logics, consult [SC01].