Part IV

Protocols

The T-1 protocol, our solution for 1-to-many knowledge authentication, is presented. It uses cryptographic hash functions to 'point at' secrets, and to prove possession of secrets. The T-1 protocol is proven secure in our extended version of GNY logic.

Chapter 9

1-to-many Protocols (T-1)

In the previous chapter, the problem of 'comparing information without leaking it' has been extensively explained. In this chapter, we will present our T-1 protocol[1], which is the most efficient solution for the 1-to-many case. That is the case where one principal has only one secret in mind, and the other player any number of secrets. In the latter part of this chapter, we will prove the T-1 protocol correct using an extended version of GNY logic. The aims and requirements of the T-1 protocol can be illustrated with the following story:

> Victor is a secret agent, and keeping secret his intelligence has a high priority. However, his mission is to protect Peggy from great dangers, so when needed, protecting Peggy takes priority over keeping his information secret. Now he is confronted with the following situation: Victor does not know whether certain information I known to him, is also known to Peggy. ('Peggy is kindly invited for a dinner at the Mallory's place.')[2] Victor knows that Mallory is a very malicious person. If Peggy does know that she is kindly invited, Victor would like to send her a warning message ('Don't go there, it is a trap. You will get killed in case you go there.'). However, if Peggy has somehow not received the invitation I, Victor would like to keep his warning for himself, as well as his knowledge of Peggy's invitation. Therefore, Victor asks Peggy

[1] The name of the protocol stems from naming convention in Table 8.1: the author of the protocol (Teepe) and a number to distinguish it from other protocols by the same author.

[2] For clarity, this information could be possession of a computer file stating the invitation. This sets apart the matter whether the information is truthful.

> to prove her knowledge of the invitation. Only after the proof, Victor will disclose his warning to Peggy. In the protocol, Peggy does not learn whether Victor actually knew about the invitation, other than from his possible next actions, such as sending a warning.
>
> Peggy is willing to prove her knowledge of the invitation I, but only if she can make sure that Victor does not cheat on her, by actually finding out about the invitation because he tricks her into telling him that she has been invited. That is, she only wants to prove her knowledge of the invitation if Victor actually knew about the invitation beforehand.

Actually, this description only describes the first one of three possible configurations of the T-1 protocol:

1. The verifier initiates ('can you prove to me that you know I?')
2. The prover initiates ('I'll show you that I know I!')
3. Mutual proof: both players simultaneously prove to one another that they possess I.

A situation where such *mutual* verification could be used in real life is 'cautious gossip', such as gossiping about the Geertje's pregnancy (explained in the opening of the previous chapter).

In this chapter we will mainly focus on configuration 1, though we stress that the proof for configuration 1 can easily be modified to prove the protocols for the other cases.

From here on we will call pieces of information 'information blocks', or IBs for short. Here follows a somewhat more formal description of the story:

> Peggy has a certain IB I. If and only if Victor also possesses this IB I, she wants to prove her possession of it to Victor upon his request. Furthermore, Peggy need not know whether Victor indeed possesses IB I, in order to execute the protocol safely.

Thus, if Victor has the same IB, he can verify that Peggy indeed has it, but if Victor does not have the same IB, he does not learn anything.

9.1 Prerequisites

The T-1 family of protocols relies on some assumptions and uses some cryptographic primitives. Furthermore, we use some terminology in the protocol and its analysis. These prerequisites will be explained in this section.

The basic assumptions are that the communication channel cannot be modified by an adversary, and that it is authenticated. That is, the principals in the protocol know with whom they are communicating. Their communication

message	meaning
`ask`(h_1)	A principal asks the other player to look whether he knows a file which has the hash value h_1.
`halt`	A principal stops proving and/or verifying possession.
`challenge`(C)	A principal asks the other player to prove possession of a file, using the challenge C.
`prove`(h_2)	A principal proves possession of a file, by presenting the hash value h_2.

TABLE 9.1: Basic messages used in the T-1 protocol.

may be overheard, but not modified. To obtain such a communication channel, standard cryptographic techniques can be used, such as asymmetric cryptography.

The most important cryptographic primitive used is the non-incremental cryptographic hash function $H(\cdot)$, which has been explained extensively in Chapter 3. This function $H(\cdot)$ is publicly known and available to all protocol particupants and malicious parties alike. The most important properties of a non-incremental cryptographic hash function $H(\cdot)$ are:

- that it is easy to compute;
- that its inverse is not easy to compute: given $H(I)$, it is infeasible to infer any property of I; and
- that it is impossible to compute $H(I)$ from other inputs than I itself (I has to be present to compute $H(I)$).

In particular, for the non-incremental cryptographic hash function the *random oracle* model is used (see Section 3.3), which is roughly the assumption that the output of $H(\cdot)$ is indistinguishable from random noise.

This primitive is sufficient for the protocols, but it is possible to improve the computational complexity of the protocol if also some form of encryption is used (see Section 9.3). For our purposes, it does not really matter whether the encryption is symmetric or asymmetric, but where we are required to choose, we will choose asymmetric encryption. (Remember that it is not unlikely that asymmetric cryptography is used already to maintain the authenticity of the communication channel.)

In the protocols of the T-1 family, a number of basic messages is used. These basic messages are listed in Table 9.1. In these basic messages, the values h_1, h_2 and C occur. The first two are hash values, the latter is a piece of information with the sole purpose to be unpredictable to anybody but the sender of the message.

As explained in the introduction of this chapter, three configurations for the protocol exist. Two configurations are asymmetric in the sense that one

principal is only a prover, and the other principal is only a verifier.[3] In these configurations, we will refer to the principals with the names Peggy (P) for the *P*rover and Victor (V) for the *V*erifier. In the third configuration, both principals take both roles. In that configuration, we refer to the principals with the names Alice (A) and Bob (B). When we refer to a principal which could be any of the principals P, V, A or B, we use the name Q.

The abbreviations P, V, A and B are unique representations of the respective identities, such as a full name and birth date, or something like a passport number. An *information block* (IB) can be represented as a unique bit string I. The collection of IBs that a principal Q possesses is denoted KB_Q (thus, KB_P for Peggy, and so on).

In the 'simple' versions of the T-1 protocol, those that do not depend upon encryption, it is assumed that the two principals have agreed upon a commonly known secret nonce N beforehand. Here, a nonce is a piece of information with the sole purpose to be unpredictable to anybody but the participating principals. The nonce N functions as a secret key shared between the participating principals.[4] The set $I_Q\star$ is the set of IBs I in possession of principal Q, for which $H(I_Q, N)$ is equal to h_1. Thus, $I_Q\star = \{I_Q \in KB_Q | H(I_Q, N) = h_1\}$.

In the 'elaborate' versions of the T-1 protocol, those that do depend upon encryption, it is assumed that the principals have set up encryption keys in such a way that the initiator of the protocol can send messages in such a way that only the not-initiating (but participating) principal can read these messages. The opposite is not required: the non-initiating principal need not be capable of sending encrypted messages to the initiating principal. The set $I_Q\star$ is the set of IBs I in possession of principal Q, for which $H(I_Q)$ is equal to h_1. Thus, $I_Q\star = \{I_Q \in KB_Q | H(I_Q) = h_1\}$.

Now that the basic prerequisites are explained, we can introduce the 'simple' versions of the T-1 protocol in which no encryption is used.

9.2 Protocol Description (Simple, no Encryption)

There are three configurations of the T-1 protocol, as mentioned in the introduction of this chapter. Of these three configurations we will first show the 'simple' version, that is the version that does not depend upon encryption. These protocols are shown in Figures 9.1 (the verifier initiates), 9.2 (the prover initiates), and 9.3 (mutual proof).

A crucial step in the protocol is the computation of the set $I_Q\star$. By computing this set, a principal 'interprets' the other principal's $\texttt{ask}(h_1)$ message. The set $I_Q\star$ is the set of principal Q's IBs that match h_1. If a set $I_Q\star$ is empty,

[3] The difference between these two configurations is whether it is the prover or the verifier that initiates the protocol.

[4] Note that N does not function like the key in a keyed cryptographic hash. A keyed cryptographic hash offers hardly any guarantees in case the key is compromised. See Section 3.2 for more details.

1. Victor chooses an IB $I_V \in KB_V$ of which he wants to test Peggy's knowledge; Victor computes $h_1 = H(I_V, N)$; Victor computes $I_V\star \subseteq KB_V$; Victor generates a random challenge C
2. Victor sends Peggy the messages $\texttt{ask}(h_1)$ and $\texttt{challenge}(C)$
3. Peggy computes $I_P\star \subseteq KB_P$
4. For each $I_{P_i} \in I_P\star$ of which Peggy is willing to prove her knowledge to Victor, the following happens:
   (a) Peggy computes $h_{2_i} = H(I_{P_i}, N, P, C)$
   (b) Peggy sends Victor the message $\texttt{prove}(h_{2_i})$
   (c) Victor verifies whether h_{2_i} is equal to any $H(I_{V_j}, N, P, C)$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes that I_{P_i} equals the matching I_{V_j}, and thereby verifies that Peggy knows the matching I_{V_j}.
5. Peggy sends Victor the message `halt`
6. Victor concludes that no more $\texttt{prove}(h_{2_i})$ messages will follow

FIGURE 9.1: The T-1 protocol where the verifier initiates and no encryption is used

---

1. Peggy chooses an IB $I_P \in KB_P$ of which she wants to prove her knowledge to Victor; Peggy computes $h_1 = H(I_P, N)$
2. Peggy sends Victor the message $\texttt{ask}(h_1)$
3. Victor computes $I_V\star \subseteq KB_V$
4. if $I_V\star = \emptyset$, Victor sends Peggy the message `halt` and the protocol is halted
5. ($I_V\star \neq \emptyset$) Victor generates a random challenge C
6. Victor sends Peggy the message $\texttt{challenge}(C)$
7. Peggy computes $h_2 = H(I_P, N, P, C)$
8. Peggy sends Victor the message $\texttt{prove}(h_2)$
9. Victor verifies whether h_2 (received from Peggy) is equal to any $H(I_{V_j}, N, P, C)$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes that I_P equals the matching I_{V_j}, and thereby verifies that Peggy knows I_{V_j}

FIGURE 9.2: The T-1 protocol where the prover initiates and no encryption is used

1. Alice chooses an IB $I_A \in KB_A$ of which she wants to prove her knowledge to Bob, and of which she wants to test Bob's possession; Alice computes $h_1 = H(I_A, N)$; Alice computes $I_A\star \subseteq KB_A$; Alice generates a random challenge C_A
2. Alice sends Bob the messages $\texttt{ask}(h_1)$ and $\texttt{challenge}(C_A)$
3. Bob computes $I_B\star \subseteq KB_B$
4. If $I_B\star = \emptyset$, Bob sends Alice the message `halt` and the protocol is halted
5. ($I_B\star \neq \emptyset$) Bob generates a random challenge C_B
6. Bob sends Alice the message $\texttt{challenge}(C_B)$
7. Alice computes $h_{2_A} = H(I_A, N, A, C_B)$
8. Alice sends Bob the message $\texttt{prove}(h_{2_A})$
9. Bob verifies whether h_{2_A} (received from Alice) is equal to any $H(I_{B_i}, N, A, C_B)$, where $I_{B_i} \in I_B\star$ (locally computed). If they are equal, Bob concludes that I_A equals the matching I_{B_i}, and thereby verifies that Alice knows the matching I_{B_i} (which we will call I_B from here on)
10. If Bob is not willing to prove his knowledge of I_B to Alice, Bob sends Alice the message `halt` and the protocol is halted
11. (Bob is willing to prove his knowledge of I_B to Alice) Bob computes $h_{2_B} = H(I_B, N, B, C_A)$
12. Bob sends Alice the message $\texttt{prove}(h_{2_B})$
13. Alice verifies whether h_{2_B} (received from Bob) is equal to $H(I_A, N, B, C_A)$ (locally computed). If they are equal, Alice concludes that I_A equals I_B, and thereby verifies that Bob knows the matching I_A

FIGURE 9.3: The mutual T-1 protocol, where no encryption is used

principal Q has no IB to prove or verify knowledge of. If there is one IB in the set, the agent may prove or verify knowledge of this IB.

It is extremely unlikely that there will be more than one IB in the set $I_Q\star$. However, the protocol easily copes with the situation if it occurs.[5] If this protocol is widely adopted and applied, it can be expected that *somewhere* this

[5] If there is more than one IB in the set $I_Q\star$, an 'external' collision of the hash function has occurred [PvO95]. This is highly improbable, but not impossible. In such a case the principal wants to discriminate between the members of the set. He can do this by making sure his challenge C_Q yields a different hash $H(I_{Q_i}, N, P, C_Q)$ for each element $I_{Q_i} \in I_Q\star$.

Ensuring this is easy because it is extremely unlikely for two IBs I and I' that both $H(I)$ and $H(I')$ clash *and* that $H(I, C_Q)$ and $H(I', C_Q)$ clash as well. If this would not be extremely unlikely, this would be a very severe problem of the supposedly cryptographic hash function. In practice, principal Q may choose a C_Q at random and check for security's sake whether there are new clashes, and choose another C_Q if this would be the case.

This whole process of generating the challenge ensures that each possible h_2 corresponds to exactly one $I_{Q_i} \in I_Q\star$. In the figures, we summarize this process as 'generating a random challenge such that it discriminates'.

A	Hey! You know what?	A	Hey! You know what?
B	Huh, What?	B	Huh, What?
A	Well, you know, don't you?	A	Well, you know, don't you?
B	I don't know what you are talking about	B	Ahh, yeah, of course
A	Well, never mind	A	Thank you, goodbye

FIGURE 9.4: A rough paraphrase of the T-1 protocols. The above are two different conversations between A and B. On the left is the conversation which can be considered an equivalent of an unsuccessful protocol run. The conversation on the right can be considered an equivalent of a successful protocol run.

situation will occur. If the protocol could not handle this situation well, data corruption would be the result. Therefore, the ability to handle such unlikely situations still is an important feature.

Note that without the challenge C in the protocol, the prover could fool the verifier if the prover could somehow obtain h_1 and h_2 without ever knowing I. Therefore, the challenge C should be unpredictable to the prover, because it makes such a scenario infeasible. The challenge is there to prevent that the prover can store and present precomputed, stored values.

Without the nonce N in the protocol, any eavesdropper who happens to know I can analyze and interpret the protocol, which is undesirable. When the eavesdropper does not know the N, this analysis and interpretation is no longer possible. In the next section we further elaborate on eavesdroppers and their abilities to interpret messages of this protocol. In typical applications of one-way hashes, the input to the hash is more or less public knowledge. This protocol on the other hand exploits the fact that the input may *not* be publicly known. Successful completion depends on one of the players being able to 'invert' the one-way hash, since it knows the original input to the hash function.

To summarize, the protocol 'obscures' messages in such a way that only recipients with specific a priori knowledge can interpret the messages. A rough paraphrase[6] of the T-1 protocols can be found in Figure 9.4. It is very sketchy, but illustrates the non-intuitivity of the protocols in an intuitive way.

9.3 Making the Protocol More Efficient (Elaborate, Encryption)

The 'simple' version of the T-1 protocol, as presented in the previous section, has a constant communication complexity[7], which leaves no room for improvement. The computational complexity does leave some room for improvement.[8]

[6] With thanks to Marius Bulacu, who came up with this paraphrase.
[7] More precisely, constant for every secret of which possession is proven.
[8] For a basic introduction to complexity, consult Section 2.3.

1. Create the look-up table, with the columns *hash* and *IB location*. *IB location* is some information on how to locate the IB on the local system. (If IBs are files, this would typically be the file name.) Make the table efficiently searchable on at least the *hash* column.
2. For each IB $I_Q \in KB_Q$, compute $H(I_Q)$, and insert $(H(I_Q), \mathit{location}(I_Q))$ into the table. (Computing the hash value has a time complexity of $\mathit{size}(I_Q)$.)
3. With each modification of personal knowledge, update the look-up table:
   (a) For each added IB I_Q, insert $(H(I_Q), \mathit{location}(I_Q))$.
   (b) For each removed IB I_Q, remove $(H(I_Q), \mathit{location}(I_Q))$.
   (c) Consider each modified IB as an old IB to be removed, and a new IB to be added.

FIGURE 9.5: The initialisation and maintenance of the look-up table, needed by any non-initiating player of the protocol

The computationally most expensive part in the protocol is the computation of the set $I_Q\star$. The time complexity of this computation is $O(\mathit{size}(KB_Q) + |KB_Q|)$, where $\mathit{size}(KB_Q) = \sum_{I_Q \in KB_Q} \mathit{size}(I_Q)$, $\mathit{size}(I_Q)$ is the number of bits in I_Q, and $|KB_Q|$ is the number of items in KB_Q (i.e., the cardinality). Note that this time complexity essentially is the space required to store all IBs.

In this section we will show how we can improve the computational complexity of the protocol by shifting this computational load to a precomputation step which is only executed once. The improved protocol can be run any number of times without requiring this hefty computation.

This process of computing $I_Q\star$ can be divided into two steps:

1. Precomputing a look-up table of size $O(|KB_Q|)$ once, which can be used in all runs of the protocol which share the same nonce. Generating the look-up table still has computational complexity $O(\mathit{size}(KB_Q) + |KB_Q|)$.
2. Looking up received hashes h_1 in the table. When an efficient storage technique for the look-up table is used, this has a time complexity of only $O(\ln |KB_Q|)$.

If principal Q learns a new IB I_Q, the principal has to update his look-up table, which has a time complexity of $O(\ln |KB_Q| + \mathit{size}(I_Q))$. How to initialize and maintain the look-up table is described in Figure 9.5.

Computing a look-up table and performing the protocol once, has the same computational complexity as performing the protocol without any precomputations. Doing precomputations has two benefits. Firstly, the speed of execution of the protocol is much higher, because there are no expensive computations to wait for. Secondly, we can only re-use the look-up table as far as it is

1. Victor chooses an IB $I_V \in KB_V$ of which he wants to test Peggy's knowledge; Victor looks up $h_1 = H(I_V)$; Victor looks up $I_V\star \subseteq KB_V$; Victor generates a random challenge C
2. Victor sends Peggy the message $\{\texttt{ask}(h_1), \texttt{challenge}(C)\}_K$
3. Peggy decrypts the message from Victor and obtains the messages $\texttt{ask}(h_1)$ and $\texttt{challenge}(C)$; Peggy looks up $I_P\star \subseteq KB_P$
4. For each $I_{P_i} \in I_P\star$ of which Peggy is willing to prove her knowledge to Victor, the following happens:
   (a) Peggy computes $h_{2_i} = H(I_{P_i}, P, C)$
   (b) Peggy sends Victor the message $\texttt{prove}(h_{2_i})$
   (c) Victor verifies whether h_{2_i} is equal to any $H(I_{V_j}, P, C)$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes that I_{P_i} equals the matching I_{V_j}, and thereby verifies that Peggy knows the matching I_{V_j}.
5. Peggy sends victor the message `halt`
6. Victor concludes that no more $\texttt{prove}(h_{2_i})$ messages will follow

FIGURE 9.6: The protocol where the verifier initiates and encryption is used

---

1. Peggy chooses an IB $I_P \in KB_P$ of which she wants to prove her knowledge to Victor; Peggy looks up $h_1 = H(I_P)$
2. Peggy sends Victor the message $\{\texttt{ask}(h_1)\}_K$
3. Victor decrypts the message from Peggy and obtains the message $\texttt{ask}(h_1)$; Victor looks up $I_V\star \subseteq KB_V$
4. if $I_V\star = \emptyset$, Victor sends Peggy the message `halt` and the protocol is halted
5. ($I_V\star \neq \emptyset$) Victor generates a random challenge C
6. Victor sends Peggy the message $\texttt{challenge}(C)$
7. Peggy computes $h_2 = H(I_P, N, P, C)$
8. Peggy sends Victor the message $\{\texttt{prove}(h_2)\}_K$
9. Victor decrypts the message from Peggy and obtains the message $\texttt{prove}(h_2)$; Victor verifies whether h_2 (received from Peggy) is equal to any $H(I_{V_j}, N, P, C)$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes that I_P equals the matching I_{V_j}, and thereby verifies that Peggy knows I_{V_j}

FIGURE 9.7: The T-1 protocol where the prover initiates and encryption is used

1. Alice chooses an IB $I_A \in KB_A$ of which she wants to prove her knowledge to Bob, and of which she wants to test Bob's possession; Alice looks up $h_1 = H(I_A)$; Alice looks up $I_A\star \subseteq KB_A$; Alice generates a random challenge C_A
2. Alice sends Bob the message $\{\texttt{ask}(h_1), \texttt{challenge}(C_A)\}_K$
3. Bob decrypts the message from Alice and obtains the messages $\texttt{ask}(h_1)$ and $\texttt{challenge}(C_A)$; Bob looks up $I_B\star \subseteq KB_B$
4. If $I_B\star = \emptyset$, Bob sends Alice the message `halt` and the protocol is halted
5. ($I_B\star \neq \emptyset$) Bob generates a random challenge C_B
6. Bob sends Alice the message $\texttt{challenge}(C_B)$
7. Alice computes $h_{2_A} = H(I_A, A, C_B)$
8. Alice sends Bob the message $\{\texttt{prove}(h_{2_A})\}_K$
9. Bob decrypts the message from Alice and obtains the message $\texttt{prove}(h_{2_A})$; Bob verifies whether h_{2_A} (received from Alice) is equal to any $H(I_{B_i}, A, C_B)$, where $I_{B_i} \in I_B\star$ (locally computed). If they are equal, Bob concludes that I_A equals the matching I_{B_i}, and thereby verifies that Alice knows the matching I_{B_i} (which we will call I_B from here on)
10. If Bob is not willing to prove his knowledge of I_B to Alice, Bob sends Alice the message `halt` and the protocol is halted
11. (Bob is willing to prove his knowledge of I_B to Alice) Bob computes $h_{2_B} = H(I_B, N, B, C_A)$
12. Bob sends Alice the message $\texttt{prove}(h_{2_B})$
13. Alice verifies whether h_{2_B} (received from Bob) is equal to $H(I_A, N, B, C_A)$ (locally computed). If they are equal, Alice concludes that I_A equals I_B, and thereby verifies that Bob knows the matching I_A

FIGURE 9.8: The symmetric protocol with encryption.

safe to re-use the nonce that was used to construct the look-up table. However, for each distinct nonce used, the player still needs to generate such a look-up table, which is by far the most expensive part of the T-1 protocols.

Therefore, we can improve dramatically on speed if we can find a way to safely re-use nonces, or to use no nonces at all. The reason to use nonces was to make sure we have semantic security with respect to any third party observing the conversation. Semantic security can also be achieved by means of encryption of some crucial parts of the protocol. The parts that need to be encrypted are those of which an eavesdropper could either infer the IB[9], or could verify the proof. To prevent inference of the IB, h_1 should be encrypted. To

[9] With 'infer', we mean 'properly guess'.

prevent verification of the proof, or the possibility to infer IB by a brute-force attack, at least one of C and h_2 should be encrypted. Since C and h_2 are always sent by opposing players, we may choose to encrypt the one sent by the player that also sent h_1, which is the player that initiated the protocol. Thus only the initiator needs to be able to send encrypted messages.

The adjusted ('elaborate') versions of the T-1 protocol are shown in Figures 9.6 (the verifier initiates), 9.7 (the prover initiates), and 9.8 (mutual proof).

By using encryption and no nonce (or a constant nonce), any responding player of the protocol needs to generate the look-up table *only once*. The need to establish a common nonce is no longer there, but the need for key exchange has come in its place. Since the protocol requires authentication, it may well be that key exchange is required anyway.

9.4 Correctness Proof in GNY Logic

In the remainder of this chapter, we will use GNY logic[10] to prove the T-1 protocols correct. To be precise, we will prove correct the configuration where the verifier initiates and no encryption is used (depicted in Figure 9.1). The configurations in which the prover initiates the protocol and where a mutual proof is exercised do not shed new light on the security analysis of the protocols. The proofs for these configurations can be obtained by slight adaptation of the proof for the configuration where the verifier initiates. The proof for the 'elaborate' version where encryption is used is very similar to the proof for the 'simple' version where no encryption is used. Therefore, we will only analyze the simple version.

Our proof uses *knowledge preconditions*, a special type of assumptions. These are explained in Section 9.4.1. The GNY idealization of the protocol and the precise protocol claims are given in Section 9.4.2. Then, in Section 9.4.3, we give an analysis of the protocol without encryption up to the point where the newly introduced inference rule **H2** is needed. A discussion on how to complete the proof, including the proof completion itself, is shown in Section 9.4.4.

In Section 9.1, it has been noted that the communication channel should be authenticated and cannot be modified by an adversary. The most important prerequisite is that the last message of the protocol is clearly bound to its sender, more precisely that the receiver can verify who is the sender. For our protocols, it is not really relevant in what way this authentication is established. To keep the proofs of the protocols as simple as possible, we simply assume a specific way of authentication of the sender. We choose a public-key signature for this. This choice is not essential and if we change this authentication method, the protocol proofs can easily be adjusted to reflect this.

It may seem counter-intuitive to prove a protocol that does not use encryption to be correct by assuming signatures, which essentially is a special case of

[10] GNY logic is summarized in Appendix B; our extension of GNY logic is explained in Chapter 6, and authentication logics in general are extensively explained in Chapter 4.

encryption. However we would like to stress that this is just the easiest way to prove the protocol correct. The issue is that we do not have to assume encryption for our protocols to work, but only sender authentication.

9.4.1 Knowledge Preconditions

A *knowledge precondition* is a special type of protocol assumption. First of all, it is an assumption which states that a particular principal has certain positive knowledge[11]. Moreover, a knowledge precondition is a neccessary condition for a protocol to end in an accepting state. For some protocols, it is useful to distinguish certain knowledge preconditions from the other protocol assumptions in order to analyse the protocol. This is because whether the protocol ends in in an accepting state should coincide with the conjunction of all distinguished knowledge preconditions. In the analysis of the T-1 protocol we use knowledge preconditions.

In the T-1 protocol, an accepting state is a state in which the verifier is convinced of the knowledge of the prover, i.e., the verifier accepts. The knowledge preconditions of the T-1 protocol are that both the verifier and the prover know the secret. Thus, a correctness proof of the T-1 protocol in GNY logic (or more precisely, a derivation of the accepting protocol state) should critically depend on the truth value of the knowledge precondition. Any unsatisfied knowledge precondition should result in the impossibility of a GNY derivation of the accepting protocol state. This kind of proof requires a *completeness assumption*[12] about cryptographic hash functions.[13] Also, correct inference rules for cryptographic hash functions are required, most notably inference rule **H2**[14]. Our completeness assumption is:

> The rules **P4**, **I3** and **H2** capture all relevant properties of cryptographic hash functions.[15]

One of the major issues of the T-1 protocols is whether the prover can cheat by asking someone else to compute the proof in name of the prover, and just forward this proof. Making someone different from the prover compute the actual proof can be achieved by either a successful man-in-the-middle attack by the prover, or by a willing assistant of the prover which *does* have the knowledge referred to in the knowledge precondition. We should design protocols in such a way that successful man-in-the-middle attacks do not exist. Authentication logics such as GNY logic help in analyzing the existence of such attacks. However, we cannot fight willing assistants of provers. In some sense, this is also unnecessary, since the goal of the secret prover protocols is to test whether the prover has effective access to the secret, and one may reasonably claim that

[11] That is, a knowledge precondition cannot be of the form 'principal A does *not* know X'.
[12] Completeness assumptions are introduced in Section 6.1.2 (page 70).
[13] Cryptographic hash functions are extensively explained in Chapter 3.
[14] Rule **H2** is introduced in Section 6.2.1 on page 74.
[15] The rules **P4** and **I3** are introduced in Appendix B.2, on page 185.

the prover indeed has such access if she has an assistant who will perform computations on the secrets on behalf of the prover.

The T-1 protocols are protocols that by design should fail to complete if the knowledge precondition is not true at the start of a protocol run. Normally in GNY logic a protocol is proven correct if we can infer the desired end state using the assumptions, inference rule and communication steps. However, for a protocol to fail if an assumption is not met, means there should not exist proofs that do not depend on the critical assumptions. This leads to the possibly somewhat counterintuitive observation that *some GNY correctness proofs prove the incorrectness of a protocol*. Exactly the proofs that do not depend on all the knowledge precondition assumptions indicate that a protocol is incorrect. We call these proofs *invalidators*. Non-existence of invalidators can only be proven if we make a completeness assumption: we assume that we know all applicable inference rules, or at least all rules that can lead us to a proof of a protocol (some of which may be invalidators).

It should be noted that the absence of invalidators does not prove correctness of a protocol in a strict sense. Just as with normal authentication logics, it only shows that the protocol has passed a test of some not-so-obvious flaws.

9.4.2 Claims and GNY Idealization

In the T-1 protocol we have two participating principals, V the *V*erifier and P the *P*rover. We assert that, for any I, our protocols satisfy the following properties:

1. 'The verifier learns whether P knows I, iff the verifier knows I and the prover wants to prove her knowledge of I':
   $V \models P \ni I$ holds after the protocol run $\Longleftrightarrow$
   $P \ni I$ and $V \ni I$ hold before the protocol run,
   and P wants to prove possession of I before the protocol run.

2. 'Only the verifier learns whether anybody knows I by means of the protocol':
   For any principal Q, except V:

   (a) $Q \models P \ni I$ holds after the protocol run $\Longleftrightarrow$
   $Q \models P \ni I$ holds before the protocol run.

   (b) $Q \models V \ni I$ holds after the protocol run $\Longleftrightarrow$
   $Q \models V \ni I$ holds before the protocol run.

3. 'Nobody learns I by means of the protocol':
   For any principal Q,
   $Q \ni I$ holds after the protocol run $\Longleftrightarrow$
   $Q \ni I$ holds before the protocol run.

Here we should mention that all right-hand sides of the assetions should include 'or ... learns X by messages outside of the protocol', where X respectively reads I (1), $P \ni I$ (2a), $Q \ni I$ (2b), and I (3). Of course, a principal may

knowledge preconditions

A.1 $P \ni I$
A.2 $V \ni I$

assumptions

A.3 $P \ni P$ A.4 $V \ni P$ A.5 $P \ni -K$ A.6 $V \ni +K$ A.7 $V \models \overset{+K}{\mapsto} P$ A.8 $V \ni C$ A.9 $V \models \sharp(C)$ A.10 $P \ni N$ A.11 $V \ni N$

the protocol itself

1 $V \rightarrow P : H(I, N), C$
2 $P \rightarrow V : \{H(I, N, P, C)\}_{-K}$

claims See section 9.4.2

FIGURE 9.9: GNY idealization of the T-1 protocol where the verifier initiates and no encryption is used. The formal GNY language used is explained in Appendix B.

learn something by a message outside of the protocol. Learning in such a way has nothing to do with any property of the protocol, as it certainly not learned *by means of* the protocol.

We will prove these properties for the T-1 protocol where the verifier initiates and no encryption is used. The GNY idealization of this protocol is given in Figure 9.9. The $\Leftarrow$ part of property 1 will be proven in sections 9.4.3 and 9.4.4. The $\Rightarrow$ part of property 1 and property 3 will be proven in Section 9.4.5. Proving property 2 requires us to make some assumptions on the beliefs and possessions of an attacker. This will be done in Section 9.4.6.

9.4.3 The Easy Part of the Proof

With the GNY idealization given in Figure 9.9, we can analyze the T-1 protocol in a rather straightforward way. The first step is to apply the *protocol parser* to the idealized protocol, shown in Figure 9.10. As discussed in Sections 4.3 and 6.2.2, this gives for every communication step (step transition) in the protocol two statements. The first statement asserts that the sender possesses (can construct) the message he is sending, the second statement asserts what the receiver will see when he receives the message.

The protocol assumptions are given in Figure 9.9. Assumptions A.1 and A.2 express that the principals do indeed know the secret. Thus, these are the knowledge preconditions. Assumptions A.3 and A.4 reflect that both principals know the identity of P. Assumption A.5 expresses that the prover knows her private key, and assumption A.6 expresses that the verifier knows the corresponding public key. Assumption A.7 reflects that the verifier believes this public key indeed corresponds to the prover's private key. Assumptions A.8 and A.9 reflect respectively that the verifier knows his own challenge and that

protocol step	sender possession (precondition)	receiver learning (postcondition)
1	$V \ni (H(I,N),C)$	$P \lhd (*H(I,N),*C)$
2	$P \ni \{H(I,N,P,C)\}_{-K}$	$V \lhd *\{*H(I,N,P,C)\}_{-K}$

FIGURE 9.10: The output of the protocol parser for the T-1 protocol where the verifier initiates and no encryption is used.

the verifier believes its freshness. Assumptions A.10 and A.11 reflect that both principals know the nonce.

Just using these assumptions we can already infer a few lines which will be needed later on in the protocol. Namely the verifier can send message 1 of the protocol, and he can verify message 2 which the prover ought to send.

B.1	$V \ni (I,N)$	**P2**(A.2, A.11)
B.2	$V \ni H(I,N)$	**P4**(B.1)
B.3	$V \ni (H(I,N),C)$	**P2**(B.2, A.8)
B.4	$V \ni (I,N,P,C)$	**P2**(A.2, A.11, A.4, A.8)
B.5	$V \models \sharp(I,N,P,C)$	**F1**(A.9)
B.6	$V \ni H(I,N,P,C)$	**P4**(B.4)
B.7	$V \ni H(H(I,N,P,C))$	**P4**(B.6)
B.8	$V \models \phi(H(I,N,P,C))$	**R6**(B.7)

Now we start the actual protocol. The verifier sends a message to the prover. Thus, the verifier learns nothing new yet. The prover, however, can calculate the proof which she will send later on in message 2. The conveyed message is shown in line C.1.

C.1	$P \lhd (*H(I,N),*C)$	[1](B.3)
C.2	$P \lhd *H(I,N)$	**T2**(C.1)
C.3	$P \lhd *C$	**T2**(C.1)
C.4	$P \lhd C$	**T1**(C.3)
C.5	$P \ni C$	**P1**(C.4)
C.6	$P \ni (I,N,P,C)$	**P2**(A.1, A.10, A.3, C.5)
C.7	$P \ni H(I,N,P,C)$	**P4**(C.6)
C.8	$P \ni \{H(I,N,P,C)\}_{-K}$	**P8**(A.5, C.7)

The justification of line C.1 may require some explanation. On line C.1, it is reflected that the recipient of the first message in the protocol learns the message. The statement on line C.1 corresponds with the 'receiver learning' part of message 1 in the output of the protocol parser (Figure 9.10). The justification B.3 corresponds with the 'sender possession' part of message 1 in the output of the protocol parser.

9.4.4 Different Options to Complete the Proof

So far, the protocol analysis is plain and rather simple. Completing the proof from here on is not as straightforward as the easy part shown above. We will provide three options to complete the proof. The first two options are flawed,

and we will explain why. The third option we present is the 'correct one': it is not flawed.

1. There is a way to prove correctness in GNY logic of this protocol without introducing new inference rules. In that case, a rather appealing but weak assumption would have to be added:

   A.12 $V \models V \overset{N}{\leftrightarrow} P$

   This assumption states that V believes that only V and P know the secret N. Using this assumption, the proof goes as follows. The conveyed message is shown in line D.1.

D.1	$V \lhd *\{*H(I,N,P,C)\}_{-K}$	[2](C.8)
D.2	$V \lhd \{*H(I,N,P,C)\}_{-K}$	**T1**(D.1)
D.3	$V \lhd *H(I,N,P,C)$	**T6**(D.2, A.6)
D.4	$V \models P \mid\sim (I,N,P,C)$	**I3**(D.3, B.4, A.12, B.5)
D.5	$V \models P \ni (I,N,P,C)$	**I6**(D.4, B.5)
D.6	$V \models P \ni I$	**P3**(D.5)

   Note that in this proof, neither the identity of P, nor P's signature are used. Though the above proof is a correct GNY logic proof, it does not help us because it depends on assumption A.12. This assumption essentially states that the verifier should trust the prover on not disclosing the secret to someone else, since the verifier has no control over the truth value of this assumption. If the prover does disclose the secret, this opens up possibilities for a successful man-in-the-middle attack: the prover can use the same nonce with multiple different principals, and use a proof given by some principal Q to prove to principal V that she knows the secret, as long as the verifier V 'knows the name of Q' ($V \ni Q$).

   To see the man-in-the-middle attack easier, observe that a protocol which does not include the identity of P would have a virtually identical analysis in GNY logic. Even without $V \ni Q$, a verifier could be deceived in such a simplified protocol.

2. In order to prove correctness without relying on assumption A.12, we need new inference rules. In Section 6.2.1 we have discussed various rules, and we will apply them here. Using rule **H1**, we can finish the protocol proofs. The proof is as follows:

D'.1	$V \lhd *\{*H(I,N,P,C)\}_{-K}$	[2](C.8)
D'.2	$V \lhd \{*H(I,N,P,C)\}_{-K}$	**T1**(D'.1)
D'.3	$V \lhd *H(I,N,P,C)$	**T6**(D'.2, A.6)
D'.4	$V \models P \mid\sim (I,N,P,C)$	**H1**(D'.3, B.4)
D'.5	$V \models P \ni (I,N,P,C)$	**I6**(D'.4, B.5)
D'.6	$V \models P \ni I$	**P3**(D'.5)

   Note that in this proof, P's identity is not used. As discussed in Section 6.2.1, rule **H1** is dubious.

B.1	$V \ni (I, N)$	**P2**(A.2, A.11)
B.2	$V \ni H(I, N)$	**P4**(B.1)
B.3	$V \ni (H(I, N), C)$	**P2**(B.2, A.8)
B.4	$V \ni (I, N, P, C)$	**P2**(A.2, A.11, A.4, A.8)
B.5	$V \models \sharp(I, N, P, C)$	**F1**(A.9)
B.6	$V \ni H(I, N, P, C)$	**P4**(B.4)
B.7	$V \ni H(H(I, N, P, C))$	**P4**(B.6)
B.8	$V \models \phi(H(I, N, P, C))$	**R6**(B.7)
C.1	$P \lhd (*H(I, N), *C)$	[1](B.3)
C.2	$P \lhd *H(I, N)$	**T2**(C.1)
C.3	$P \lhd *C$	**T2**(C.1)
C.4	$P \lhd C$	**T1**(C.3)
C.5	$P \ni C$	**P1**(C.4)
C.6	$P \ni (I, N, P, C)$	**P2**(A.1, A.10, A.3, C.5)
C.7	$P \ni H(I, N, P, C)$	**P4**(C.6)
C.8	$P \ni \{H(I, N, P, C)\}_{-K}$	**P8**(A.5, C.7)
D″.1	$V \lhd *\{*H(I, N, P, C)\}_{-K}$	[2](C.8)
D″.2	$V \lhd \{*H(I, N, P, C)\}_{-K}$	**T1**(D″.1)
D″.3	$V \lhd *H(I, N, P, C)$	**T6**(D″.2, A.6)
D″.4	$V \models P \mid\sim *H(I, N, P, C)$	**I4**(D″.2, A.6, A.7, B.8)
D″.5	$V \models P \mid\sim (I, N, P, C)$	**H2**(D″.4, B.4)
D″.6	$V \models P \ni (I, N, P, C)$	**I6**(D″.5, B.5)
D″.7	$V \models P \ni I$	**P3**(D″.6)

FIGURE 9.11: GNY proof of the T-1 protocol where the verifier initiates and no encryption is used.

3. Using the well-justified rule **H2** from Section 6.2.1, we can also finish the protocol proofs. The proof is as follows:

D″.1	$V \lhd *\{*H(I, N, P, C)\}_{-K}$	[2](C.8)
D″.2	$V \lhd \{*H(I, N, P, C)\}_{-K}$	**T1**(D″.1)
D″.3	$V \lhd *H(I, N, P, C)$	**T6**(D″.2, A.6)
D″.4	$V \models P \mid\sim *H(I, N, P, C)$	**I4**(D″.2, A.6, A.7, B.8)
D″.5	$V \models P \mid\sim (I, N, P, C)$	**H2**(D″.4, B.4)
D″.6	$V \models P \ni (I, N, P, C)$	**I6**(D″.5, B.5)
D″.7	$V \models P \ni I$	**P3**(D″.6)

Note that changing this proof to use rule **H3** is trivial: P only needs to insert "I know" into its signed messages, and V only needs to verify that this token is indeed present in the message.

Thus, the whole GNY proof of the T-1 protocol is as in Figure 9.11. If we observe that the prover will only engage in the protocol if he wants to prove possession of i, we have proven the $\Leftarrow$ part of property 1 (stated in Section 9.4.2).

9.4.5 Proving principals do not learn too much

So far, of the properties stated in section 9.4.2, we have only proven the $\Leftarrow$ part of property 1. In this section, we will prove the the $\Rightarrow$ part of property 1, and we will prove property 3.

1. 'The verifier learns whether P knows I, iff the verifier knows I and the prover wants to prove her knowledge of I', $\Rightarrow$ part: We assume $V \models P \ni I$ holds after the protocol run (the verifier has been convinced of prover's possession of I).

   For the prover to be able to actually prove possession of I, she has to use it while constructing message 2. If she does not possess I, she cannot satisfy step C.6, which is necessary for C.8, which states message 2. This is because the only way in which the prover can obtain C.7 is by application of inference rule **I3**.[16] Thus, $P \ni I$ holds before the protocol run.

   If the prover would not want the verifier to possibly learn that the prover knows I, the prover would not have sent message 2. Thus, the prover wants to prove her knowledge of I.

   For the verifier to be able to verify the proof, he has to possess I as well. More specifically, the verifier has to verify whether the message he sees in line D''.3 (or equivalently, in D.3 or D'.3) equals the value the verifier computed at line B.6.[17] Thus, $V \ni I$ holds before the protocol run.

2. 'Nobody learns I by means of the protocol': We prove this by contradiction. Let us assume that principal Q does learn I by means of the protocol, and that by analyzing messages Q managed to reconstruct I. I itself is never conveyed except as an argument to a one-way hash function. Thus, Q managed to invert a one-way hash function. Obviously, this is impossible.

   In GNY logic, this is reflected in the inference rules of our completeness assumption (**P4**, **I3** and **H2**): in every rule in which something is inferred from a term which involves a term of the form $H(X)$ (i.e., rules **I3** and **H2**), the principal that learns something by means of the inference rule, must possess X as a condition for the inference rule to apply[18].

---

[16] See also the completeness assumption about cryptographic hash functions on page 134.

[17] Note that this protocol does not depend on the recognizability constraint of rule **I4**, as used in line D''.3 of the last proof. Even if the verifier can *always* recognize the message sent by the prover, as needed for verification of the signature, the verifier still cannot verify the proof, as the verifier has nothing to compare the message with. If we change the protocol to use rule **H3**, introduction of the "I know"-token will lead to immediate recognizability of the signed message. This will not invalidate the proof.

[18] In rule **I3**, P learns something as a result of observing $H(X, S)$, but only if $P \ni (X, S)$ is also satisfied, i.e., if P knows the protected secret already. Similarly, in rule **H2**, V learns something as a result of believing P conveyed $H(X, P)$, but only if $V \ni (X, P)$ is also satisfied, i.e., if V knows the protected secret already. Note that the principal names P and V in the rules **I3** and **H2** do not only apply to the principals P and V in the protocol, but to *any* principal.

Except that the protocol works, it is also very efficient. Both the verifier and the prover only need to perform a constant number of steps. The prover will, upon seeing $*H(I, N)$, look whether she has a matching secret I. Only after establishing that she actually does, she will start further computations. The bottleneck of course is recognizing an I which matches the sent $H(I, N)$. A principal can in fact generate a look-up table in advance, which stores for each possible I the corresponding $H(I, N)$ value. This is a one-time operation whose cost is proportional to the total size of all secrets that a player wants to be able to look up. This has to be done for each value of N the principal is interested in. If however the protocol which uses encryption is used, this dependency on N disappears.

9.4.6 Modeling the beliefs and possessions of an attacker

In the previous section we have shown that no principal can learn I itself from observing the protocol. However, we are also interested in anything that an eavesdropper *could* learn. Could an eavesdropper become convinced that the prover or the verifier knows I? This is what property 2 of section 9.4.2 is about. Or, less bad but still undesirable: could an eavesdropper learn about what secret I the protocol is run if she already knows the secret herself?

Let us assume that, at the start of the protocol, Eve the eavesdropper knows everything the participating principals know, except P's private key, the nonce N and the challenge C, but including the secret I:

E.1	$E \ni I$	E.3	$E \models \overset{+K}{\mapsto} P$	E.5	$E \models \sharp(C)$
E.2	$E \ni +K$	E.4	$E \ni P$		

In the course of the protocol, E will learn both $\{H(I, N, P, C)\}_{-K}$, C and $H(I, N)$. Since Eve does not know N, she will never be able to infer what secret I the protocol is run about, since in all messages where I is communicated, it is 'mixed' with N in a one-way hash function. For the same reason Eve cannot verify P's proof. Thus, all three values Eve learns are indistinguishable from random noise (as per the *random oracle model*, see Section 3.3). In the case of the protocol that uses encryption instead of a nonce, E will learn $\{H(I), C\}_{+K}$ and $\{H(I, P, C)\}_{-K}$. E cannot decrypt the first message, and therefore never learns C, which is needed to be able to interpret $\{H(I, P, C)\}_{-K}$.

An eavesdropper knowing everything except private keys and the shared nonce does not learn anything sensible from observing the protocol. This is a strong result. One of its implications is that N may be known to any principal who is either (1) trusted by the verifier, or (2) not capable of intercepting messages from or to any principal using the nonce N.

One last question is whether one of the participants could be a passive attacker. In that case, the attacker would also possess N. For the case the attacker is the verifier, the proof is trivial, since the goal of the protocol is that the verifier *does* learn. For the case where the attacker is the prover, the prover *will* indeed learn what secret the protocol is about. However, the prover will not learn whether the verifier really possesses I: the verifier might have learned

$H(I, N)$ from someone else.

9.5 Conclusion

In this chapter, we have presented our T-1 protocol. Let us return to Victor, the secret agent who wants to protect Peggy from great danger (the story which opened this chapter, page 123). Peggy is invited by Mallory, but will get killed by Mallory if she accepts the invitation. Peggy is reluctant to disclose the invitation to Victor. Let us see how the T-1 protocol handles this situation.

1. *How do Victor and Peggy make sure the invitation is not disclosed in case the other does not know of the invitation?*

   Peggy and Victor only send cryptographic hash values of the invitation. From a cryptographic hash value, one cannot infer the pre-image (Section 3.2). Thus, nobody can learn the invitation from the communicated messages.

2. *How do Victor and Peggy establish about what secret they are communicating?*

   Victor sends Peggy the cryptographic hash value of the invitation I. If Peggy knows the invitation, she recognizes the hash value.

3. *How does Victor become convinced of Peggy's knowledge of the invitation?*

   Victor asks Peggy to present a cryptographic hash value of the invitation and a *challenge* C chosen by Victor. Only if Peggy has the invitation, she will be able to construct the requested cryptographic hash value.

4. *How is a man-in-the-middle attack prevented?*

   In the pre-image of the hash value that must convince of possession, the identity of the prover must be incorporated.

   In a man-in-the-middle attack, Peggy would try to deceive Victor by initiating a concurrent protocol with someone else (say, Cecil) and passing the messages from Cecil to Victor.

   If Peggy sends Cecil's messages to Victor, these hash values will be constructed with Cecil's identity, and not Peggy's. Victor expects hash values in which Peggy's identity P has been incorporated. He can detect it if this is not the case. Thus, if Peggy mounts a man-in-the-middle attack, Victor will not be convinced.

5. *How do Peggy and Victor make sure that eavesdroppers cannot learn anything from the protocol?*

   There are two solutions:

   - They use a commonly agreed, secret nonce N, which is incorporated into every pre-image before the hash is computed. As the eavesdropper does not know the nonce, the eavesdropper cannot learn anything.

- They use encryption to hide the hash values from which the eavesdropper could learn something.

6. *What if one of the principals does not know the invitation?*

   In that case the ignorant principal sees bits which he/she cannot distinguish from random noise. The ignorant principal may continue the protocol, but will not convince the other principal.

7. *What if not Victor, but Peggy wants to initiate the protocol?*

   In that case, Peggy sends the first message of the protocol. All configurations of the protocol are listed in Figures 9.1–9.3 (using a nonce N), and Figures 9.6–9.8 (using encryption).

8. *What would an actual protocol run look like, if we omit all the technicalities?*

   Look at Figure 9.4 on page 129.

The T-1 protocol is proven correct in our extended version of GNY logic.[19] The T-1 protocol works for knowledge authentication[20] in the 1-to-many case: where one of the principals 'points at' a secret, and the other principal looks whether he/she knows the same secret. In the next chapter, the T-1 protocol is generalized to the many-to-many case.

[19] GNY logic is summarized in Appendix B, and our extensions are explained in Chapter 6.
[20] Knowledge authentication is introduced in Chapter 8.