The T-2 protocol, our solution for many-to-many knowledge authentication, is presented. It is a parallel composition of the T-1 protocol. It relies on prefix trees made from hash values to optimize communication complexity. The average-case communication complexity is experimentally estimated.

Chapter 10

Many-to-many Protocols (T-2)

In the previous chapter, we presented our T-1 protocol, which is a solution for 'comparing information without leaking it' in 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. The T-1 protocol can be generalized to the many-to-many case: the case where two principals each have a large number of secrets and wish to determine the intersection of their secrets. In this chapter, we will present and explain our generalization, the T-2 protocol.[1]

The T-2 protocol offers the same security guarantees as the T-1 protocol, because the T-2 protocol can be seen as a group of parallel runs of the T-1 protocol, and the T-1 protocol is secure under parallel composition. The T-2 protocol is, for cooperative principals, very efficient in terms of communication complexity and computational complexity.

Moreover, the T-2 protocol offers *fairness*: the players can make sure that they only prove possession of secrets for which they receive a reciprocal proof in return.

10.1 Using Prefix Trees for Efficiency

The T-2 protocol heavily relies on prefixes and prefix trees. To grasp the working of the protocol, it is important to understand some properties of prefix trees, and how this relates to *set intersection*. This section will offer the necessary background on this subject.

[1] Discussions with Sieuwert van Otterloo have been of critical value for the work that is reported in this chapter. The results reported up to and including Section 10.3 are joint work.

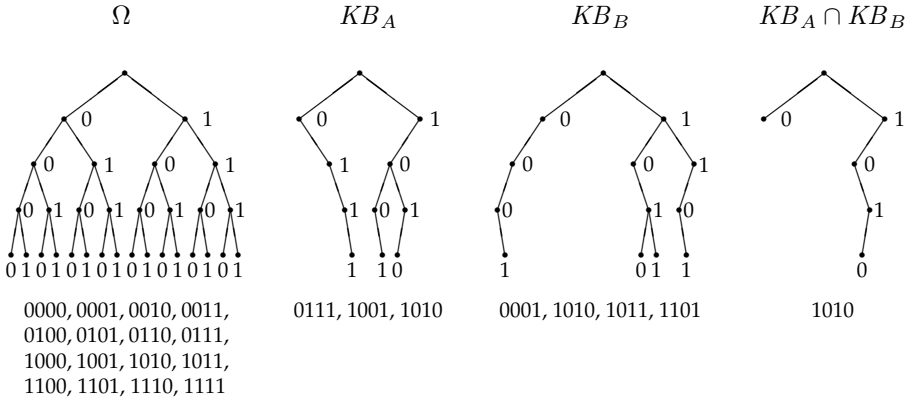

FIGURE 10.1: Sets KB_A, KB_A represented as binary hash value prefix trees. In this toy example, the length of the hash value is 4 bits. Leaf nodes at depth 4 represent full hash values. The prefix trees of the full domain Ω and the intersection $KB_A \cap KB_B$ are also shown. Below every prefix tree, the list of hash values in the tree is shown.

Consider two principals, Alice and Bob, with knowledge bases KB_A and KB_B. A simple way to generalize the T-1 protocol to the many-to-many case, is to repeat the protocol $|KB_A|$ times: in every run of the T-1 protocol, another single secret of Alice is compared with all Bob's secrets. For every secret of Alice, l bits (where l is the length of the hash value, a security parameter) have to be exchanged. A typical message would be: 'Bob, do you have the secret with hash `1ab1dabb2c90231e3182b15ffcacd732`?'[2] It turns out that if a specific secret of Alice is not a secret of Bob, the communication of l bits is largely a waste of communication capacity.

In the T-1 protocol, one player 'points at a secret', by sending a message of the form `ask`(h_1). The T-2 protocol differs from the T-1 protocol in that the process of pointing at secrets is performed interactively and iteratively by both principals. Instead of pointing at a secret by presenting a bit string of length k, very short bit strings are presented. A typical message would be: 'Bob, do you happen to have any secrets with hash prefix `1a`?'[3] Both Alice and Bob may in fact have many secrets with this hash prefix. If it is the case that Bob does not have any secret with hash prefix `1a`, he can say so to Alice: 'Alice, I do not have any secrets with hash prefix `1a`!'[4] As a result of such 'refusal' message, Alice knows that stating the full hash values which start with `1a` is a waste of time.

The hash values of a set of secrets can be represented in a binary prefix tree. Figure 10.1 shows the binary prefix trees for the toy example where the length of the hash value is four bits, Alice has three secrets, and Bob has four secrets.

The way that the T-2 protocol optimizes the communication complexity is

[2] Technically, the message that is exchanged is '`ask(1ab1dabb2c90231e3182b15ffcacd732)`'.
[3] Technically, the message that is exchanged is '`ask(1a)`'.
[4] Technically, the message that is exchanged is '`refuse(1a)`'.

	encoding of the prefix tree				
depth	1	2	3	4	encoding
tree	01	0101	01010101	0101010101010101	size (bits)
Ω	11	1111	11111111	1111111111111111	30
KB_A	11	0110	0111	010110	16
KB_B	11	1011	10 0110	01 1101	18
$KB_A \cap KB_B$	11	0010	01	10	10

TABLE 10.1: Binary encoding of some hash prefix trees. The depth corresponds to the depth of the corresponding tree (see Figure 10.1). The two bits at depth 1 signify the two possible branches at depth 1, the four bits at depth 2 signify the possible two branches at depth 2, and so on. Whitespace is inserted in the encodings to align the bits of the encoding with the table header.

by constructing (almost) precisely the prefix tree that belongs to the set $KB_A \cap KB_B$ (an *intersection prefix tree*). The efficiency of a protocol that does just this depends on the size of an intersection prefix tree (counted in nodes), and on the way an intersection prefix tree is encoded. Determining the expected size of an intersection prefix tree is far from trivial, and we will devote Section 10.4 to this.

Creating an efficient encoding scheme for a binary tree (without labels) is relatively simple: A binary tree can be represented in $(1 + 2 \cdot \#\text{nodes})$ bits. One bit is used to encode whether the root node exists. Then, for every existing node, two bits are used: one bit for each of the two branches that may spring from it. The bit is 0 if there is no such branch, and 1 if there is such a branch. There are a few options to order the bits; we choose for 'breadth-first' order: that is the order in which a breadth-first search algorithm would visit the nodes.[5] The binary strings that stem from this encoding scheme are self-delimiting.

The binary hash prefix trees we use are a specific subset of all possible binary trees, because the depth of the binary hash prefix trees is bounded by the size l of the hash values. This means that binary hash prefix trees can be encoded even more efficiently: the nodes that are at depth l cannot have any branches springing from it.[6] Therefore nodes at depth l need zero bits for their encoding. Also, when we assume that binary hash prefix trees are nonempty, we can omit the bit for the root node. Using this coding scheme, a binary hash prefix tree with maximum depth l can be encoded in this number of bits:

$$2 \cdot (\#\text{nodes} - \#\text{nodes at depth } l)$$

Table 10.1 illustrates our binary hash prefix tree coding scheme for the sets given in Figure 10.1.

[5] This is sometimes called 'level-order traversal'.
[6] We adhere to the convention that we consider the root node of a tree to be at depth 0.

message	meaning
`ask`(p_1)	A principal asks the other player to look whether he has any secrets which have a hash value that starts with p_1
`refuse`(p_1)	A principal tells the other player he will not prove possession of any secrets whose hash value starts with p_1
`challenge`(h_1, C)	A principal asks the other player to prove possession of a secret with hash value h_1, using the challenge C
`prove`(h_1, p_2)	A principal partially proves possession of a secret, identified by h_1, by presenting the prefix p_2.

TABLE 10.2: Basic messages used in the T-2 protocol. The prefixes can be of any length between 0 and l (where l is the length of the hash value, a security parameter).

10.2 Specification of the T-2 Protocol

The prerequisites of the T-2 protocol are the same as the prerequisites of the T-1 protocol, given in Section 9.1. Here, we will mainly describe the parts of the T-2 protocol that differ from the T-1 protocol.

The T-2 protocol is a protocol for two principals, which we will call Alice (A) and Bob (B). They have their respective knowledge bases KB_A and KB_B. They want to determine the intersection $KB_A \cap KB_B$ without leaking any information on the items outside of the intersection $KB_A \cap KB_B$. Here we assume Alice and Bob want to mutually prove possession of the items in the intersection $KB_A \cap KB_B$, but the protocol can easily be adjusted to allow for a unidirectional proof. Also, we adopt the setting that A and B have established a shared secret nonce N, and that no encryption is used. The protocol can easily be adjusted to encryption instead of a nonce.

Both players know one another's names, A and B. They have established a common shared secret nonce N, and computed the hash $H(I_Q, N)$ for each $I_Q \in KB_Q$. The length of the hash values used in the protocol is l bits. These hash values are stored in a binary prefix tree, with the corresponding files at the leaf nodes. Since hash values have an equal length, all leaf nodes in the prefix tree are at depth l.

The basic messages that are exchanged in the T-2 protocol are listed in Table 10.2.[7] Note that sending a message of the form `refuse`(p_1) is an explicit refusal: the principal that sends this message states that he does not have any

[7] It is easy to see that the basic messages of the T-1 protocol are special cases of these messages:

- `ask`(h_1) is a special case of `ask`(p_1) where the length of p_1 is equal to l.
- `halt` is a special case `refuse`(p_1) where the p_1 is of zero length.
- `challenge`(C) is s special case of `challenge`(h_1, C) where h_1 is equal to the h_1 of the previously sent `ask`(h_1) message.
- `prove`(h_2) is a special case of `prove`(h_1, p_2) where h_1 is equal to the h_1 of the previously sent `ask`(h_1) message and the length p_2 is equal to l.

secrets with hash prefix p_1 of which he is willing to prove possession in this protocol run. Conversely, sending a message of the form $\texttt{ask}(p_1)$ is *not* an explicit confirmation: the principal that sends this message may have not a single secret with hash prefix p_1 of which is he willing to prove possession in the current protocol run.

One might think that sending a message of the form $\texttt{ask}(p_1)$ *should* be an explicit confirmation, but this is neither necessary nor sensible. It is not necessary because the only *convincing* confirmation is a $\texttt{prove}(h_1, p_2)$ message where p_2 meets certain criteria. It is not sensible because one cannot verify whether the player sending the $\texttt{ask}(p_1)$ message actually has a message with hash prefix p_1.

A run of the T-2 protocol consists of many subprotocols. There are two types of subprotocols: one for determining the approximation $KB_{AB?}$ of the intersection $KB_A \cap KB_B$ (described in Section 10.2.1), and one for performing the mutual proof of possession of the elements in the intersection $KB_A \cap KB_B$ (described in Section 10.2.2).

In the text, ϵ, p, p_Q, h, h_n and s denote binary strings, and $\cdot$ denotes string concatenation. The empty string is denoted as ϵ. When applied to numbers, $\cdot$ denotes multiplication. The length (in bits) of the hash values is l, which is the same for all hash values. The length (in bits) of the challenges is l_c, which is the same for all challenges.

10.2.1 Subprotocol for Determining Intersection

The subprotocol for determining the approximation $KB_{AB?}$ of the intersection of KB_A and KB_B takes a recursive divide-and-conquer approach. By means of messages, the domain of possible mutually owned files is divided into a number of smaller domains, and for each of these domains a new 'subprotocol' is started. By means of recursion, these domains get smaller, and eventually the players either state that they do not have files within the domain, or the domain contains only a single file. In the latter case, a subprotocol for proving possession is started (described in Section 10.2.2).

A subprotocol is started by a player sending an `ask` message, which is typically $\texttt{ask}(\epsilon)$. When a player (let us say Alice) sends a message $\texttt{ask}(p)$ with $\text{length}(p) < l$, the other player (let us say Bob) is obliged to respond with a set of messages R_p such that Bob gives a full account of the domain denoted by p. Roughly, for every part of the domain, Bob has to tell whether he has secrets with the corresponding hash prefix. The amount of detail in the response of Bob is hardly restricted. The only constraint for Bob is that if Bob does not refuse having any secrets in the domain, his description has to be *more* detailed than the question of Alice. *How much more* detailed it will be, is up to Bob to decide. (A special case of the protocol, which we will introduce shortly, is where the amount of extra detail is fixed.)

More formally, a response set is a set $R_p = \{M | M = \texttt{ask}(p \cdot s) \text{ or } M = \texttt{refuse}(p \cdot s)\}$, such that there is a set S_p of binary strings which satisfies the following properties:

$$S_p = S_p^{\texttt{ask}} \cup S_p^{\texttt{refuse}}, \tag{10.1}$$
$$S_p^{\texttt{ask}} = \{s | \texttt{ask}(p \cdot s) \in R_p\}, \tag{10.2}$$
$$S_p^{\texttt{refuse}} = \{s | \texttt{refuse}(p \cdot s) \in R_p\}, \tag{10.3}$$
$$S_p^{\texttt{ask}} \cap S_p^{\texttt{refuse}} = \emptyset, \tag{10.4}$$
$$\forall s \in S_p^{\texttt{ask}} : \text{length}(s) \geq 1, \tag{10.5}$$
$$\forall s \in S_p^{\texttt{refuse}} : \text{length}(s) \geq 0, \tag{10.6}$$
$$\forall s \in S_p : \text{length}(s) \leq (l - \text{length}(p)), \tag{10.7}$$
$$\forall s \in S_p : \neg\exists s' \in S_p : \exists s'' \in \{0,1\}^* : s = s' \cdot s'', \tag{10.8}$$
$$\forall h \in \{0,1\}^l : \exists s \in S_p : \exists s'' \in \{0,1\}^* : h = s \cdot s''. \tag{10.9}$$

Thus, the binary string p is suffixed with a number of strings s. Some of the suffixes correspond to `ask` messages (10.2), and some to `refuse` messages (10.3). There are no suffixes which are used in an `ask` message *and* in a `refuse` message simultaneously (10.4). Suffixes which are part of an `ask` message have minimum length one (10.5), and suffixes which are part of a `refuse` message have minimum length zero (10.6).

When all suffixes are taken together into S_p (10.1) the following hold: Every suffix is length-bounded by $l - \text{length}(p)$ (i.e., $\text{length}(p \cdot s) \leq l$) (10.7). There are no two suffixes such that one suffix is a prefix of the other suffix (10.8). Every binary string of length l has a prefix in S_p (10.9).

Note that within a `refuse` message, s may be of zero length, but not in an `ask` message. If an s of zero length would be allowed in an `ask` message, there would be no guarantee that the T-2 protocol would ever terminate: every `ask` message could be answered with exactly the same message, creating an infinite loop.

Because both players know what messages they send, and all messages sent are assumed to arrive, both players can detect whether the protocol has terminated, i.e., whether all obligations have been met. The most efficient protocol run possible for determining the intersection is a run in which both players only send $\texttt{ask}(p_1)$ messages if they indeed possess secrets whose hash value has the prefix p_1, and send $\texttt{refuse}(p_1)$ messages otherwise. This strategy can however not be enforced.[8]

To illustrate how a collection of subprotocols establishes the intersection of two knowledge bases, let us suppose that hash values are only four bits long, A and B have restricted themselves to strings s of length 1. Moreover, suppose that A possesses files with prefixes 0111, 1001 and 1010, and B possesses files with prefixes 0001, 1010, 1011 and 1101. These sets correspond with the sets KB_A and KB_B in Figure 10.1. Within this context, Table 10.3 shows how the protocol may develop.

[8] That is, enforcing such a strategy would result in a far less efficient protocol, while guaranteeing protocol efficiency would be the purpose of enforcing the strategy.

step	player	p	messages (R_p)	message meaning
1	A		{ask(ϵ) }	I've got some secrets whose prefix is ϵ.
2	B	ϵ	{ask(0) ,	I have got some secrets with prefix 0
			ask(1) }	and some with prefix 1.
3	A	0	{refuse(00) ,	I do not have secrets with prefix 00,
			ask(01) }	but I do have some with prefix 01;
		1	{ask(10) ,	moreover, I have secrets with prefix 10,
			refuse(11) }	but I don't have any with prefix 11.
4	B	01	{refuse(010) ,	} Sorry, no secrets with prefix 01 here,
			refuse(011) }	
		10	{refuse(100) ,	also no secrets with prefix 100,
			ask(101) }	but indeed some with 101 here.
5	A	101	{ask(1010) ,	Some secrets with prefix 1010,
			refuse(1011)}	but none with 1011 here.

TABLE 10.3: A sample run of interleaved subprotocols for establishing the intersection. In the message meaning column, 'some' should be read as 'zero or more', and 'secret with prefix' should be read as 'secret with hash value with prefix'.

It can be seen that A and B in turn increase the prefixes in their messages by one bit. Every ask(p_1) message obliges the opposing player to respond. More specifically, step 2 is a response to step 1, step 3 contains two responses to step 2, step 4 contains two responses to step 3, and finally step 5 is a response to the last part of step 4. Step 5 should lead to a subprotocol for mutually proving possession of the file with prefix 1010. In course of the protocol run, A has said she does not possess files whose hashes start with either 00, 11 or 1011, and B has said he does not possess files whose prefixes start with either 010, 011 or 100. Thus, 9/16 of the full hash domain has been refused by A, and 6/16 by B. There remains 1/16 of the domain of which neither player has refused possession, though the subprotocol for determining intersection has terminated. This means that this remaining 1/16 part of the domain denotes possible candidates $KB_{AB?}$ for actual list intersection. For each element in the remaining set, a subprotocol for proving possession must be invoked. In this case, the remaining set contains only one hash value, 1010.

The protocol run shown in Table 10.3 can also be depicted as a binary tree that grows as the protocol proceeds. Figure 10.4 shows this binary tree. The binary tree after protocol execution (the rightmost one) closely resembles the hash value prefix tree that belongs to $KB_A \cap KB_B$, shown at the right of Figure 10.1.

The protocol states shown in Figure 10.4 can also be depicted as colored surfaces, which is done in Figure 10.2. The whole surface denotes the whole set Ω. As the protocol progresses, the structure of the surface becomes finer. Light gray blocks denote parts of the domain for which Alice has sent a refuse mes-

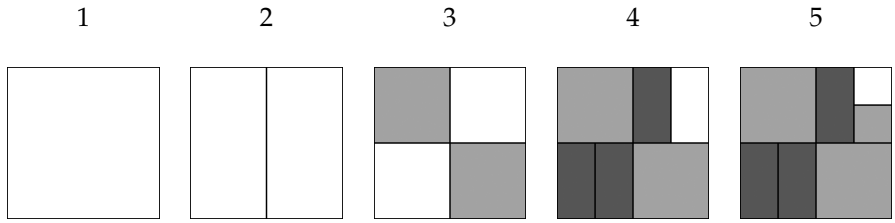

FIGURE 10.2: Interleaved subprotocols for establishing the intersection, shown as a colored surface, with $l = 4$, $|KB_A| = 3$, $|KB_B| = 4$, $|KB_A \cap KB_B| = 1$. The protocol run depicted here corresponds with the protocol run shown in Table 10.3 and Figure 10.4. See the text for explanation.

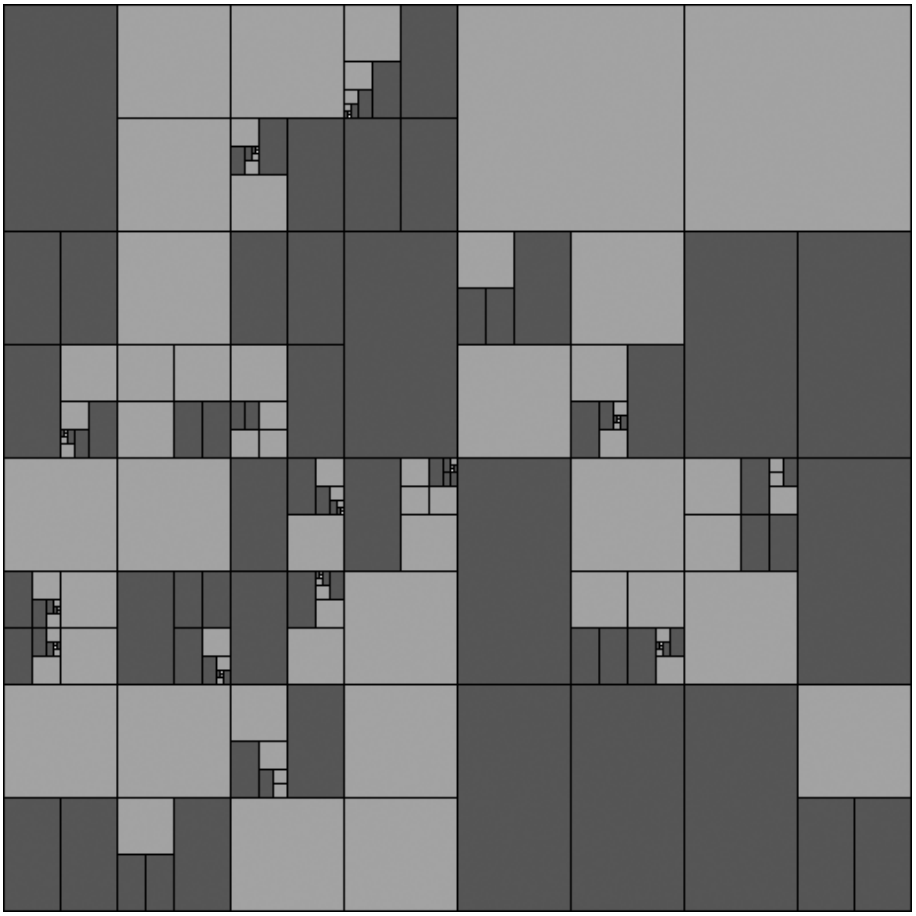

FIGURE 10.3: Interleaved subprotocols for establishing the intersection, shown as a colored surface, with $l = 16$, $|KB_A| = 40$, $|KB_B| = 40$, $|KB_A \cap KB_B| = 10$. See the text for explanation.

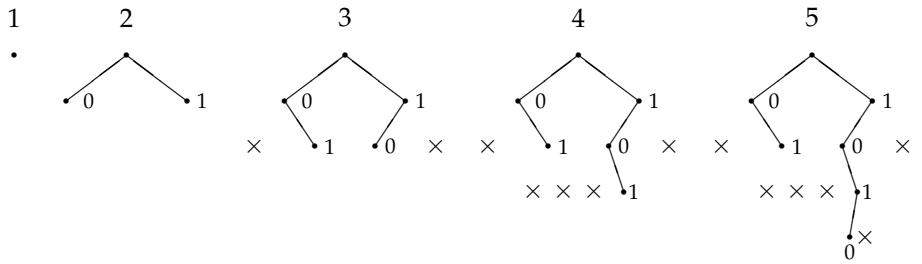

TABLE 10.4: Interleaved subprotocols for establishing the intersection, shown as a growing binary tree. The protocol run depicted here corresponds with the protocol run shown in Table 10.3. Every '×' corresponds with a `refuse` message. Every node corresponds with an `ask` message. The leftmost tree, which contains only the root node, corresponds with the protocol state after the first message ($\texttt{ask}(\epsilon)$). The rightmost tree corresponds with the protocol state after completion of the protocol, when $KB_{AB?} = \{1010\}$ has been established.

sage. Dark gray blocks represent parts of the domain for which Bob has sent a `refuse` message. White blocks represent parts of the domain for which neither Alice nor Bob has sent a `refuse` message. The sizes of the blocks correspond with the proportion of the domain that has been `refused` in the corresponding message.

For a first impression of how the protocol scales up, and how this affects the protocol state, Figure 10.3 shows the final state of a protocol where $l = 16$, $|KB_A| = 40$, $|KB_A| = 40$, $|KB_A \cap KB_B| = 10$, all suffixes s are of length 1 and all hash values are randomly chosen.[9]

While meeting their protocol obligations, the participants have a lot of freedom, of which we mention a few important aspects:

1. It is not a protocol violation to send $\texttt{ask}(p_1)$ messages if the player actually does *not* have any secret $I_V \in KB_V$ of which p_1 is a hash prefix. The player may 'act as if' he has some secrets which he in fact does not have.

2. The set R_p (sent in response to a message $\texttt{ask}(p)$) does not have to be sent at once. It may be sent in parts, interleaved with other response sets $R_{p'}$. Parts of R_p may even be sent only after the opposing player has performed some moves. This also means that the exact details of R_p can be chosen in response to the opposing player's future moves.

3. The length of the string s may be longer than 1. Thus, a player can choose to surrender multiple bits of information to the opposing player within one step.

These freedoms allow participants in the protocol to choose between a lot of different strategies. In Section 10.3 we will elaborate on various strategies.

[9] Because the hash values come from a cryptographic hash function, it is safe to assume such a random distribution.

The approximation $KB_{AB?}$ of the intersection $KB_A \cap KB_B$ is the set of hash values p for which one of the players has sent a message $\texttt{ask}(p)$ with $\text{length}(p) = l$. It is guaranteed that every secret of which both Alice and Bob are willing to mutually prove possession, has a corresponding hash value in the set $KB_{AB?}$. However, there is no guarantee that there are no bogus hash values in the set $KB_{AB?}$: hash values for which either Alice or Bob does not know a corresponding secret.[10] For every member of the set $KB_{AB?}$ a subprotocol for proving possession has to be executed to determine whether the hash value is bogus or not.

10.2.2 Subprotocol for Proving Possession

When the subprotocols for determining intersection have been completed, Alice and Bob have constructed a set $KB_{AB?}$ of hash values for which it is claimed that both Alice and Bob possess a secret with the corresponding hash values. This set can be transformed into the set $KB_A \cap KB_B$ by application of a subprotocol for proving possession to every element $h_1 \in KB_{AB?}$. If a subprotocol is convincing for a principal Q, this principal considers the secret $I \in KB_Q$ for which $h_1 = H(I, N)$ holds to be an element of the intersection $KB_A \cap KB_B$. Thus, the subprotocol for proving possession is a sifting on the set $KB_{AB?}$.

An instance of the subprotocol for proving possession is invoked when within a subprotocol for determining intersection, a message $\texttt{ask}(p)$ has been sent, with $\text{length}(p) = l$. This 'prefix' p actually contains a complete hash value h_1. This hash value h_1 refers to a unique secret I, and an instance of the subprotocol is purely dedicated to one specific secret. Essentially, the two players give one another a NP-complete puzzle which they can only solve feasibly if they indeed possess the secret. The solution is again a hash value, but now for both players the hash value is different: the solutions to the puzzles prove possession of the same file, but the puzzles and the answers themselves differ. Alice has to compute and show $h_A = H(I, N, A, C_B)$, whereas Bob has to compute and show $h_B = H(I, N, B, C_A)$.

For running an instance of the subprotocol for proving possession, a principal has to maintain seven state variables, which are listed in Table 10.5.

Every message of the subprotocol contains h_1, to distinguish the messages of the current subprotocol from other subprotocols. In the beginning of the subprotocol, C_A and C_B are exchanged without further ado. Using these challenges, both players can compute h_2^A and h_2^B. Next, the players in turn send one another prefixes of their proof messages m_A and m_B. In every step, the prefix must be longer than the prefix sent in the previous step. If in the end of the protocol it turns out that $m_A = h_A$ and $m_B = h_B$, the players have indeed mutually proven possession of the file denoted by h.

[10] In extremely rare cases, it might happen that Alice and Bob want to mutually prove possession of different secrets which have the same hash value. Such hash values are also included in $KB_{AB?}$.

variable	meaning
h_1	the hash value that denotes the secret for which the subprotocol has been invoked, $h_1 = H(I, N)$
C_A	the challenge chosen by Alice
C_B	the challenge chosen by Bob
h_2^A	the hash value that constitutes Alice's proof, $h_2^A = H(I, N, A, C_B)$
h_2^B	the hash value that constitutes Bob's proof, $h_2^B = H(I, N, B, C_A)$
p_A	the proof message disclosed by Alice
p_B	the proof message disclosed by Bob

TABLE 10.5: State variables in a subprotocol for proving possession

More formally, we arrive to the following description of the subprotocol for proving possession:

- When a principal Q sends a message `ask`(p), with $\text{length}(p) = l$, the player is obliged to also send a message `challenge`(h_1, C_Q) with $h_1 = p$. The challenge C_Q is of fixed length and its contents may be freely chosen by the sender Q.

- When a principal Q receives a message `challenge`(h_1, C), and he has not yet sent a message `challenge`(h_1, C_Q) (that is, a challenge for the same value of h_1), this player is obliged to send two messages: `challenge`(h_1, C_Q) where C_Q may be freely chosen by the sender, and `proof`(h_1, p_Q) where p_Q is a binary string with $\text{length}(p_Q) \leq l$.

- When a player receives a message `proof`(h_1, p), with $\text{length}(p) \leq l$, then he has to respond with a message `proof`(h_1, p_Q), with $p_Q = p'_Q \cdot s$, where p'_Q is the last prefix the principal Q has disclosed within the current subprotocol (or ϵ if the player has not yet sent a `proof` message within the current subprotocol). There is a required minimum length of p_Q, which depends on p. If $\text{length}(p) = l$, then p_Q should be of length l as well. Otherwise, it should be the case that $\text{length}(p_Q) > \text{length}(p)$. The subprotocol terminates when both players have sent a prefix of length l.

What actually happens within this subprotocol, is that the players in turn disclose a few bits of the value they must compute, the solution to their NP-hard puzzle. In each step, the prefix shown must be at least one bit longer than the last one shown by the opposite player.

Similar to the subprotocol for determining intersection, the hash values are disclosed in parts, more specifically in ever longer prefix strings. As in the protocol for determining intersection, it is important to appreciate what information is actually transferred within the messages. The bit strings p_A and p_B may be equal to h_2^A and h_2^B respectively, but this is not required. Of course, if these values are (pairwise) not equal, this implies that the secrets that correspond to h_1 will not be considered an element of $KB_A \cap KB_B$.

step	player	messages	message meaning
1	A	{ask(1010), challenge(1010, 0110)}	Let's run the protocol for 1010, I challenge you, $C_A = 0110$.
2	B	{challenge(1010, 1110), proof(1010, 1)}	I challenge you, $C_B = 1110$, the first bit of my proof is 1.
3	A	{proof(1010, 01)}	The first two bits of my proof are 01.
4	B	{proof(1010, 110)}	The first three bits of my proof are 110.
5	A	{proof(1010, 0100)}	The four bits of my proof are 0101.
6	B	{proof(1010, 1101)}	The four bits of my proof are 1101.

TABLE 10.6: A sample run of the subprotocol for proving possession. The hash value secret for which possession can be proven in this protocol is 1010. For completeness, the ask message leading to the protocol is included in the first step.

To illustrate how a single subprotocol works, let us again suppose that hash values are only four bits long, and A has sent ask(1010) to B. Table 10.6 shows how the protocol may develop. At the end of the protocol run the full 'proofs' p_A and p_B are known: $p_A = 0100$ and $p_B = 1101$. Whether these 'proofs' are convincing depends on whether $p_A = h_2^A$ and $p_B = h_2^B$.

Again, the principals have a lot of freedom in how they act in the protocol:

1. The players are not obliged to actually send prefixes of the actual proof of possession; they are allowed so send any information they like, as long as they adhere to the syntactical rules.

2. The players may increase the length of their prefixes faster than the minimum requirement.

These options leave room for many strategies. Assuming that the players want to prove possession of the secret to one another and want to make sure they get a reciprocal proof in return, there is an optimal strategy which is very simple. As soon as both challenges are known, compute h_A and h_B. As long as the other player's shown prefix is a prefix of the proof he should send, reply 'truthfully' by sending the prefix of your own proof, which is only one bit longer than the other player's last prefix. As soon as the other player's shown prefix is not a prefix of the expected proof, stop sending parts of your own proof, but append random noise to your previously sent prefixes until the protocol terminates.

This strategy ensures that if the other player chooses to abort the protocol, the other player only has an advantage of one bit. Also, if the other player does not know the secret I_V, your proof of possession of the secret is not communicated to the other player (or at most one bit of it). When both players adopt this strategy, they will obtain a mutual proof when both players possess the secret I_V in question. As such, this strategy guarantees *fairness* (see page 23).

10.3 Making the Protocol Efficient by Restrictions

The protocol description leaves a lot of freedom to principals participating in the protocol (see the lists on pages 153 and 156). This means that the principals can develop and use various strategies while engaging in the T-2 protocol. For example, a *reluctant* strategy is to never send `refuse` messages. A *cooperative* strategy is to send send as many `refuse` messages as possible, while 'merging' `refuse` messages of adjacent prefixes.[11]

Due to this large amount of freedom in the protocol specification it is impossible to give precise measurements of the communication complexity of the protocol. Moreover, the freedom itself increases the communication complexity: many possible protocol actions imply the need for many bits to encode one single protocol action. In this section, we will impose restrictions on the protocol that allow (1) efficient encoding of the protocol actions and (2) precise computations and measurements of the communication complexity (given in Section 10.4).

The restrictions we impose are the following:

1. It is assumed Alice starts the protocol with the message $\texttt{ask}(\epsilon)$
2. In the subprotocols for determining intersection:
   (a) All suffixes s are of length 1.
   (b) All sets R_p are sent in order of increasing length of p.
   (c) All sets R_p with equal length of p are sent in the lexicographical order of p.
3. In the subprotocols for proving possession:
   (a) The challenges are sent as soon as possible in the lexicographical order of h_1.
   (b) For every h_1, the first $\texttt{proof}(h_1, p_Q)$ message, p_Q is of length 1. These first proofs are sent as soon as possible, but after all challenges have been sent.
   (c) All suffixes s are of length 2, except the last suffix, which is of length 1.
   (d) All messages $\texttt{proof}(h_1, p_Q)$ are sent in order of increasing length of p_Q.
   (e) All messages $\texttt{proof}(h_1, p_Q)$ with equal length of p_Q are sent in the lexicographical order of h_1.

[11] That is, it is not the sheer number of messages that counts, but the proportion of the domain Ω for which a principal sends `refuse` messages. If this proportion is maximal, the strategy is called *cooperative*.

messages (R_p)	encoding
$\{\texttt{refuse}(p \cdot 0), \texttt{refuse}(p \cdot 1)\}$	00
$\{\texttt{refuse}(p \cdot 0), \texttt{ask}(p \cdot 1)\}$	01
$\{\texttt{ask}(p \cdot 0), \texttt{refuse}(p \cdot 1)\}$	10
$\{\texttt{ask}(p \cdot 0), \texttt{ask}(p \cdot 1)\}$	11

TABLE 10.7: Encoding for sets R_p where $\forall s\colon |s| = 1$ and p may be omitted. Every `refuse` is encoded as a 0, and every `ask` as a 1.

These restrictions have a huge impact on the protocol runs: The first message message can be omitted (1). Bob (Alice) sends only sets R_p where the length of p is odd (even) (2a). The principals send all their sets R_p out as soon as possible (2b), which reduces the number of communication steps for determining intersection down to l. The principals send their sets R_p in a strictly imposed order (2c).

The challenges are sent in a strictly imposed order (3a). Bob (Alice) sends only proofs p_A (p_B) where the length of p_A (p_B) is odd (even) (3b and 3c). The principals send all their proofs p_Q out as soon as possible (3b and 3d), which reduces the number of communication steps for proving possession down to $l + 1$. The principals send their proofs p_Q in a strictly imposed order (3e).

As a result, the total number of communication steps in a protocol run where possession is proven is $2 \cdot l + 1$. If the set $KB_{AB?}$ is empty, the total number of communication steps is at most l. Moreover, as a result of the imposed order, it is always possible to reconstruct for which prefix p or hash value h_1 a message is bound to arrive. Thus, sending the prefix p or the hash value h_1 itself is redundant.

In the subprotocols for determining intersection, the suffixes s are of length 1, and the prefix p can be omitted from the message sets R_p. As a result, we can encode every set R_p in only two bits. This is shown in Table 10.7.

In the subprotocols for proving possession, h_1 can also be omitted because of the ordering of the messages. Also, sending the full prefix p_Q in every message is redundant, because a large part of the prefix has already been sent in a previous message. It is only needed to send the suffix s.

Using these coding schemes, the protocols shown in Tables 10.3 and 10.6 can be merged into one single protocol run, shown in Table 10.8. It is a run of the restricted T-2 protocol on the sets $KB_A = \{0111, 1001, 1010\}$ and $KB_B = \{0001, 1010, 1011, 1101\}$. The principals are cooperative, they send as many `refuse` messages as possible. The column 'message' denotes the actual communicated bits. Observe that from the bits in the 'message' column, it is possible to reconstruct the columns p and h_1, and with help from Table 10.7 it is possible to reconstruct the decoded messages. Moreover, observe that the first twelve bits communicated in the protocol (110110000110) correspond exactly to the binary encoding (explained in Section 10.1) of the prefix tree that is constructed in the protocol (shown at the right of Figure 10.4).

step	player	p	h_1	message	decoded message
0	A				$\{\texttt{ask}(\epsilon)\}$
1	B	ϵ		11	$\{\texttt{ask}(0), \texttt{ask}(1)\}$
2	A	0		01	$\{\texttt{refuse}(00), \texttt{ask}(01)\}$
		1		10	$\{\texttt{ask}(10), \texttt{refuse}(11)\}$
3	B	01		00	$\{\texttt{refuse}(010), \texttt{refuse}(011)\}$
		10		01	$\{\texttt{refuse}(100), \texttt{ask}(101)\}$
4	A	101		10	$\{\texttt{ask}(1010), \texttt{refuse}(1011)\}$
			1010	0110	$\texttt{challenge}(1010, 0110)$
5	B		1010	1110	$\texttt{challenge}(1010, 1110)$
			1010	1	$\texttt{proof}(1010, 1)$
6	A		1010	01	$\texttt{proof}(1010, 01)$
7	B		1010	10	$\texttt{proof}(1010, 110)$
8	A		1010	00	$\texttt{proof}(1010, 0100)$
9	B		1010	1	$\texttt{proof}(1010, 1101)$

TABLE 10.8: A sample protocol run of the restricted T-2 protocol, efficiently encoded. Only the bits in the column 'message' are communicated. From it, the column 'decoded message' can be reconstructed.

10.4 Determining Communication Complexity

Now that we have fully explained the T-2 protocol and its restricted version, we want to establish the communication complexity of the restricted T-2 protocol. Some steps in this process are simple, some are rather complicated. Our approach is simple: we compute the simple parts of the complexity, and we perform some experiments to estimate the complicated parts of the complexity.

Although in the previous section, we have restricted the freedom of the principals dramatically, the principals still have room for various strategies. We have already mentioned the two most important strategies: the *cooperative* and the *reluctant* strategy. When applied to the restricted T-2 protocol, these strategies are implemented as follows:

cooperative Choose the sets R_p in such a way that the number of ones in the encoding of R_p shown in Table 10.7 is minimized. This leads to an intersection prefix tree of minimal size.

reluctant Choose the sets R_p in such a way that the number of ones in the encoding of R_p shown in Table 10.7 is maximized. This leads to an intersection prefix tree of maximal size.

Both Alice and Bob can independently choose their strategy. There are more strategies than the cooperative and the reluctant strategies, but all other strategies fall complexity-wise 'in between' the complexities of these two strategies.

strategy		upper bound		
Alice	Bob	size of the binary rep. of the prefix tree	$\lvert KB_{AB?}\rvert$	communication complexity (bits)
coop.	coop.	$2 \cdot l \cdot \min(\lvert KB_A\rvert, \lvert KB_B\rvert)$	$\lvert KB_A \cap KB_B\rvert$	$2 \cdot l \cdot \min(\lvert KB_A\rvert, \lvert KB_B\rvert)$ $+2 \cdot (l + l_c) \cdot \lvert KB_A \cap KB_B\rvert$
coop.	rel.	$4 \cdot \lvert KB_A\rvert \cdot l$	$2 \cdot \lvert KB_A\rvert$	$2 \cdot \lvert KB_A\rvert \cdot (3 \cdot l + l_c)$
rel.	coop.	$4 \cdot \lvert KB_B\rvert \cdot l$	$2 \cdot \lvert KB_B\rvert$	$2 \cdot \lvert KB_B\rvert \cdot (3 \cdot l + l_c)$
rel.	rel.	$2 \cdot (2^l - 1)$	2^l	$2^{l+1} \cdot (1 + l + l_c) - 2$

TABLE 10.9: The worst case communication complexity for the restricted T-2 protocol, depending on the strategies of Alice and Bob. The communication complexity, shown at the right, is the size of the binary representation of the prefix tree plus $2 \cdot (l + l_c) \cdot \lvert KB_{AB?}\rvert$.

Therefore, it is sufficient to analyze these two strategies to form an impression of how the communication complexity depends on the strategies chosen.

- If both Alice and Bob use the cooperative strategy a prefix tree of minimal size is constructed. In the case of the running example of this chapter, this corresponds to the tree shown at the right of Figure 10.4.
- If both Alice and Bob use the reluctant strategy, a prefix tree spanning the full domain Ω is constructed. In the case of the running example of this chapter, this corresponds to the tree shown at the left of Figure 10.1. The size of this tree in binary encoding is $2 \cdot (2^l - 1)$.
- If Alice uses the cooperative strategy and Bob the reluctant strategy, a prefix tree is constructed that closely matches the hash value prefix tree of Alice. In the case of the running example of this chapter, this corresponds to the tree shown under KB_A in Figure 10.1. The size of this tree in binary encoding is bounded by $4 \cdot \lvert KB_A\rvert \cdot l$.
- The case where Alice uses the reluctant strategy and Bob uses the cooperative strategy is symmetric to the previous case.

The total communication complexity of a protocol run is the sum of the communication complexities of the subprotocols for determining intersection and the subprotocols for proving possession. The former is precisely the size of the binary encoding of the constructed tree; the latter is precisely $2 \cdot (l + l_c) \cdot \lvert KB_{AB?}\rvert$, where l is the length of the hash values in bits, and l_c is the length of the challenges in bits.

The worst case communication complexities can be calculated rather easily. The results are shown in Table 10.9. The only case for which the worst case communication complexity is not so trivial is the case where both principals use the cooperative strategy. The biggest prefix tree that can be constructed in this setting occurs in case the prefix trees corresponding to KB_A and KB_B overlap almost completely, in which case the size of the prefix tree of the intersection is just a little below $2 \cdot l \cdot \min(\lvert KB_A\rvert, \lvert KB_B\rvert)$. This is however extremely

unlikely to actually happen, as the prefix trees belonging to the sets KB_A and KB_B have a uniform random distribution.

For the settings where at least one of the principals uses the reluctant strategy, the average communication complexity is equal to the worst case communication complexity. The average case communication complexity for the case where both principals use the cooperative strategy can be expected to be much lower than the worst case.

Observe that both principals can force the communication complexity to be at most $2 \cdot |KB_Q| \cdot (3 \cdot l + l_c)$ by using the cooperative strategy.

The average case communication complexity for the case where both principals use the cooperative strategy can be derived mathematically, but this is very complicated.[12] So instead, we did some experiments to estimate the communication complexity in this setting.

The communication complexity for the T-2 protocol consists of two contributions:

- *communication resulting from secrets that* are *shared*
  This communication is heavily influenced by $|KB_{AB?}|$. For cooperative principals $|KB_{AB?}|$ will be equal[13] to $|KB_A \cap KB_B|$.
  - *communication in subprotocols for determining intersection*
    This is bounded by $2 \cdot l \cdot |KB_{AB?}|$.
  - *communication in subprotocols for proving possession*
    This is exactly $2 \cdot (l + l_c) \cdot |KB_{AB?}|$ bits.
- *communication resulting from secrets that* are not *shared*
  This is only communication in the subprotocols for determining intersection. We estimated this experimentally.

When these contributions are added up, there will be a little bit of double counting, as some communication in the subprotocols for determining intersection is due to both shared secrets and not-shared secrets.

To estimate the communication resulting from secrets that are not shared, we performed an experiment in ten different conditions. The conditions differ in the sizes of the sets KB_A and KB_B, and these are shown in Table 10.10. In every condition, $l = 256$ and $KB_A \cap KB_B = \emptyset$. The hash values corresponding to the sets KB_A and KB_B were taken randomly from the domain 2^l where every element had an equal probability. For every condition, the experiment was performed 1000 times, and the number of bits communicated is recorded.

The results of the experiment are shown in Figure 10.4 and Table 10.11. Table 10.11 reports for each of the conditions the minimum and maximum observations, the median, the average and the standard deviation, and also the

[12] We have not yet succeeded in establishing a formula which expresses the communication complexity in which we have sufficient confidence. Our gratitude goes to various people who have tried to help us in finding this formula, most notably to Gerard te Meerman.

[13] There is a negligible chance that $|KB_{AB?}|$ is larger than $|KB_A \cap KB_B|$; in that case one or more *collisions* of the hash function must have occurred.

condition	$\|KB_A\|$	$\|KB_B\|$	$\frac{\|KB_A\|}{\|KB_B\|}$
1	1	1	1
2	1	10	0.1
3	1	100	0.01
4	1	1000	0.001
5	10	10	1
6	10	100	0.1
7	10	1000	0.01
8	100	100	1
9	100	1000	0.1
10	1000	1000	1

TABLE 10.10: The ten conditions of the experiment to estimate the average communication complexity of the restricted T-2 protocol with cooperative principals. Every condition can be identified by two of the three variables $|KB_A|$, $|KB_B|$ and $\frac{|KB_A|}{|KB_B|}$.

bounds of the interval in which the middle 95% of the observations lie. Figure 10.4 is a density (local frequency) plot of the data.[14]

At first glance, the only conclusion that can be drawn is that larger sets lead to more communication. At closer observation, one can see that there are peaks at approximately 5,5, 55, 550 and 5500 bits that correspond to conditions where $\frac{|KB_A|}{|KB_B|} = 1$, in increasing order of $|KB_A|$. Similarly, there are peaks at approximately 13,6, 136 and 1360 bits that correspond to conditions where $\frac{|KB_A|}{|KB_B|} = 0.1$. Also, there are peaks at approximately 23,4 and 234 bits that correspond to conditions where $\frac{|KB_A|}{|KB_B|} = 0.01$. This suggests two findings:

1. An increase of a factor 10 in the sizes of both $|KB_A|$ and $|KB_B|$ leads to an increase of a factor 10 of the communicated bits.

2. The fraction $\frac{|KB_A|}{|KB_B|}$ influences the number of communicated bits.

To investigate these hypotheses, we divide the communication by the sum of the set sizes, which gives us Figure 10.5 and Table 10.12. Figure 10.5 is a density plot[15] of the communicated bits divided by $|KB_A|+|KB_B|$, and Table 10.12 gives descriptive statistics of the data (similar to Table 10.11). The conditions have been re-ordered to highlight the structure that can be seen in Figure 10.5.

The findings are very clear:

[14] Technically, Figure 10.4 is not a density plot. It is a density plot of the base 10 logarithm of the observations with modified labels at the x-axis. The labels at the x-axis are 10^x where it should technically read x. In this way, the data plotted is easy to read, while the 'visual surface' for each distribution is equal.

[15] As opposed to Figure 10.4, Figure 10.5 is a 'true' density plot. The x-axis is in linear scale, and the surfaces below the lines are both 'visually' and mathematically equal (i.e., equal to 1).

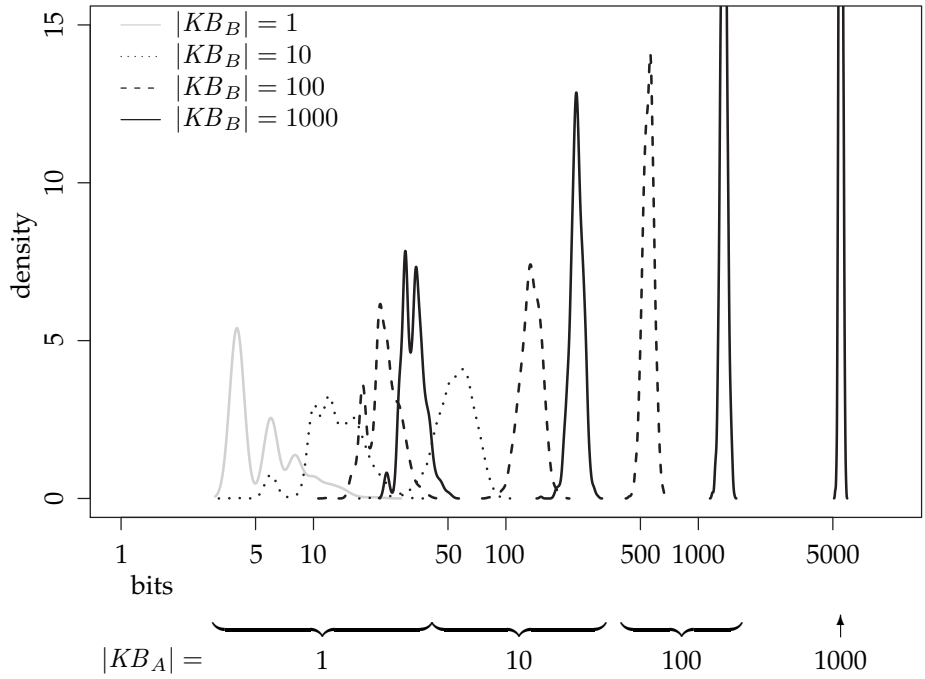

FIGURE 10.4: The number of communicated bits in the restricted T-2 protocol with cooperative participants, shown as a compressed density plot.[14] For every distribution, $|KB_A|$ can be found by finding its peak, and looking straigt down to where either a brace or an arrow is found. At the other side of the brace or arrow, $|KB_A|$ is printed.

$\|KB_A\|$	$\|KB_B\|$	min	-95	med	$+95$	max	avg	stdev
1	1	4	4	4	14	22	5,97	2,77
1	10	4	6	12	24	30	13,67	4,24
1	100	12	16	22	34	40	23,47	4,40
1	1000	24	24	34	44	52	33,38	4,61
10	10	20	34	56	78	94	55,67	11,75
10	100	82	104	136	168	196	135,82	16,95
10	1000	152	200	234	268	300	233,84	18,25
100	100	438	484	552	620	660	551,16	35,26
100	1000	1180	1256	1356	1470	1534	1358,29	54,01
1000	1000	5142	5298	5510	5736	5860	5508,95	110,98

TABLE 10.11: Descriptive statistics of the number of communicated bits in the restricted T-2 protocol with cooperative participants. For every condition, 1000 experiments were done. Shown are the minimum and maximum observations (min, max), the bounds of the interval where the middle 95% of the observations lie (-95, $+95$), the median andaverage (med, avg), and the standard deviation (stdev).

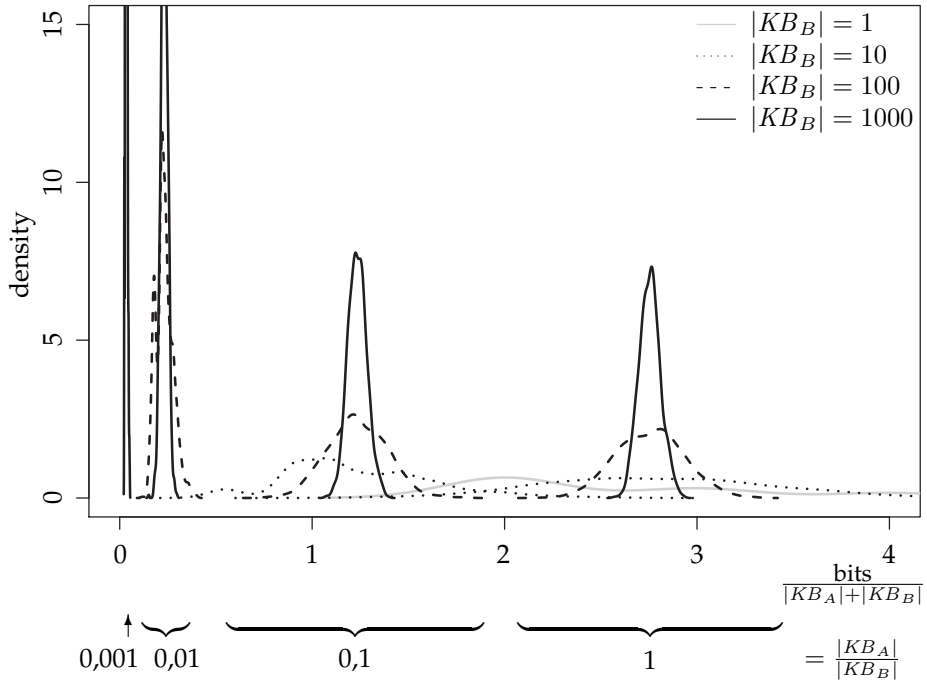

FIGURE 10.5: The number of communicated bits per compared secret in the restricted T-2 protocol with cooperative participants, shown as a density plot. For every distribution, $\frac{|KB_A|}{|KB_A|}$ can be found looking straight beneath its peak.

$\frac{\lvert KB_A\rvert}{\lvert KB_B\rvert}$	$\lvert KB_B\rvert$	min	-95	med	$+95$	max	avg	stdev
0,001	1000	0,024	0,024	0,034	0,044	0,052	0,034	0,005
0,01	100	0,119	0,158	0,218	0,317	0,396	0,232	0,044
0,01	1000	0,151	0,198	0,232	0,265	0,297	0,232	0,018
0,1	10	0,364	0,545	1,091	2,182	2,727	1,243	0,386
0,1	100	0,745	0,945	1,236	1,527	1,782	1,234	0,154
0,1	1000	1,073	1,142	1,233	1,337	1,395	1,235	0,049
1	1	2,000	2,000	2,000	7,000	11,000	2,986	1,386
1	10	1,000	1,700	2,800	3,900	4,700	2,783	0,588
1	100	2,190	2,420	2,760	3,090	3,300	2,756	0,176
1	1000	2,571	2,649	2,755	2,868	2,930	2,755	0,055

TABLE 10.12: Descriptive statistics of the number of communicated bits per compared secret in the restricted T-2 protocol with cooperative participants. For every condition, 1000 experiments were done. Shown are the minimum and maximum observations (min, max), the bounds of the interval where the middle 95% of the observations lie (-95, $+95$), the median and average (med, avg), and the standard deviation (stdev).

protocol	upper bound on average communication complexity (bits)
iterated T-1	$(l+1)\cdot\|KB_A\| + (2\cdot l + 2\cdot l_c)\cdot\|KB_A \cap KB_B\|$
restricted T-2 with cooperative principals	$2,76\cdot\|KB_A \cup KB_B\| + (4\cdot l + 2\cdot l_c)\cdot\|KB_A \cap KB_B\|$

TABLE 10.13: Bounds on average communication complexities of the T-1 and the T-2 protocol.

1. For every fraction $\frac{|KB_A|}{|KB_B|}$, the average amount of communicated bits per $|KB_A|+|KB_B|$ is practically identical. In the worst case, where $\frac{|KB_A|}{|KB_B|} = 1$, the average communication complexity is approximately $2,76$ bits per $|KB_A| + |KB_B|$.

2. When the fraction $\frac{|KB_A|}{|KB_B|}$ decreases (i.e., the difference between $|KB_A|$ and $|KB_B|$ grows), the average amount of communicated bits per $|KB_A| + |KB_B|$ decreases.

3. As $|KB_B|$ (and $|KB_A|$) grow larger, the standard deviation clearly declines.

This is good news. The first finding essentially means that the communication complexity of the restricted T-2 protocol with cooperative principals is linear in $|KB_A|+|KB_B|$. The second finding means that if $|KB_A|$ and $|KB_B|$ are of dissimilar size, the efficiency of the protocol increases; the average communication complexity is always below (approximately) $2,76\cdot 2\cdot\max(|KB_A|,|KB_B|)$. The third finding entails that the protocol scales up very well: as the set sizes increase the actual communication complexity for a particular run will be very close to the expected communication complexity (which is linear in $|KB_A| + |KB_B|$).

Table 10.13 gives a first impression of the average communication complexity of the restricted T-2 protocol with cooperative principals. Also, the communication complexity of the alternative, iteration of the T-1 protocol, is shown.

For the T-1 protocol, the actual communication complexity is exactly the formula given. The principals could still optimize the complexity, if they first establish whether $|KB_A|$ or $|KB_B|$ is smaller, and change roles in case $|KB_A| > |KB_B|$. With this optimization, the term $(l+1)\cdot|KB_A|$ can be replaced with $(l+1)\cdot\min(|KB_B|,|KB_B|)$, at the cost of an extra term that signifies the communication complexity of the protocol that determines whether $|KB_A| > |KB_B|$ is the case.

For the restricted T-2 protocol with cooperative principals, with $|KB_A| = |KB_B|$ the average communication complexity is slightly lower than the formula given (due to double counting of some communicated bits). If $|KB_A| \neq |KB_B|$, the average communication complexity improves even more. When $\frac{\min(|KB_A|,|KB_B|)}{\max(|KB_A|,|KB_B|)} = 0,1$, the factor 2,76 reduces to approximately 1,24. When

$\frac{\min(|KB_A|,|KB_B|)}{\max(|KB_A|,|KB_B|)} = 0,01$, the factor 2,76 reduces to approximately 0,23. The precise relation between the fraction and the factor remains subject to future research. Nevertheless, it is not too hard to see that the factor decreases so fast that the average communication complexity of the T1 protocol is always larger than the average communication complexity for the restricted T-2 protocol with cooperative principals.[16]

10.5 Conclusion

In this chapter, we have generalized the T-1 protocol into the T-2 protocol that does many-to-many intersection. Thus, using the T-2 protocol, the following is possible, in an efficient way[17]:

- Airline carriers can allow the authorities of (for example) the United States of America to check whether wanted terrorists are on board of the airplane without disclosing the identities of the non-criminals.
- Police officers can compare the electronic dossiers of their investigations without relying on a trusted third party (i.e., the VROS, see Section 1.5).
- Intelligence agencies can compare their secrets without disclosing them.

In cases where the set sizes $|KB_A|$ and $|KB_B|$ are public information, the T-2 protocol is just as secure as the T-1 protocol. The T-2 protocol works by means of two principals disclosing hash value prefixes, in increasing length. In this way, they can efficiently and adaptively survey the complete domain of possible secrets.

The T-2 protocol leaves room for various strategies to be used by the principals. Due to the security of the protocol, there is no informational benefit in choosing one strategy over the other. The chosen strategy influences the communication complexity. If the T-2 protocol is restricted in a particular way, efficient coding is used, and both principals use a particular strategy, then the protocol satisfies the *fairness condition*.

[16] There are two ways in which this can be seen:

1. If iterated T-1 would be faster than T-2, this would mean that in the restricted T-2 protocol with cooperative principals, the branches of the tree that correspond with the KB_A would 'escape' the intersection prefix tree at a depth greater than l. That would mean that many collisions of the cryptographic hash function would occur. The chance of this happening is negligible, and is certainly nowhere close to average case behavior.
2. Observe that the communication complexity of the T-2 protocol where Alice uses the cooperative strategy and Bob the reluctant strategy (shown in Table 10.9), is equal to the communication complexity for the case where both principals use the cooperative strategy and $KB_B = \Omega$ (Bob holds the whole domain). That communication complexity, $2 \cdot |KB_A| \cdot (3 \cdot l + l_c)$ is only larger than $(l+1) \cdot |KB_A| + (2 \cdot l + 2 \cdot l_c) \cdot |KB_A \cap KB_B|$ by a constant factor. (If $l = l_c$, the factor is $8/5$). But as KB_B is only a sparse subset of Ω, this factor will be defeated.

[17] See Section 8.1 for a detailed description of these application areas.

Let us see how the T-2 protocol works:

1. *How are the individual secrets of the players protected?*

   The T-2 protocol is a parallel composition of the T-1 protocol. See Section 9.5 for a summary of why the T-1 protocol protects the secrets of the players.

2. *Is the number of secrets that a player possesses hidden from the other player?*

   No. If a player wants to infer how many secrets the other player has, and the other player plays the *cooperative* strategy[18], then the first player can play the *reluctant* strategy to find out how many secrets the other player has. However, when both players play the reluctant strategy, the communication complexity (and thus the run time) of the T-2 protocol is exponential.

   In the analysis of the T-2 protocol, one should consider the set sizes $|KB_A|$ and $|KB_B|$ to be public knowledge.

3. *How can the players enforce fairness?*

   The proofs of possession are disclosed bit by bit, turn by turn. Both players know which bits should be sent by the other player. As soon as the bit sent by the other player is different from the expected bit, a player stops sending his own proof, and starts sending random noise bits. In this manner, the advantage that a player can get over the other player is limited to one bit of the proof.

4. *How is the communication complexity optimized?*

   The hash values of the secrets that are only known to one player, are only partially communicated. Of these hash values, only a prefix is communicated that is just long enough for the 'ignorant' player to determine he does not possess a secret corresponding to the hash value prefix.

   Moreover, all hash values and prefixes are compressed by representing them in a tree structure.

5. *How many communication steps are required?*

   When the players have a secret in common, the number of communication steps is $2 \cdot l + 1$, where l is the length of the hash values in bits. When the players have no secrets in common, the number of communication steps depends on the number of secrets they possess; in practice the number of communication steps will be much smaller than l.

6. *How many bits need to be communicated?*

   For every secret that is possessed by only one player, on average at most three bits are communicated. For every secret that is possessed by both players, $4 \cdot l + 2 \cdot l_c$ bits are communicated, where l is the length of the hash values in bits, and l_c is the length of the challenge.

[18] The cooperative and reluctant strategy are explained in Section 10.4.