New Protocols for Proving Knowledge of Arbitrary Secrets While not Giving Them Away

Wouter Teepe
University of Groningen
w.g.teepe@ai.rug.nl

Abstract

This paper introduces and describes new protocols for proving knowledge of secrets without giving them away: if the verifier does not know the secret, he does not learn it. Three role configurations exist for this type of protocols: (1) the prover may want to pro-actively prove knowledge of a secret, (2) a verifier may ask someone to prove knowledge of a secret, or (3) two players may mutually prove knowledge of a secret. Protocols for all three cases are shown in this paper. This can all be done while only using one-way hash functions. If also the use of encryption is allowed, these goals can be reached in a more efficient way, giving a total of six protocols (three without encryption and three with).

keywords protocols, zero knowledge, interactive proving, comparing information without leaking it (CIWLI), one-way hash functions, MAC's, list intersection problem

1 Introduction

In zero-knowledge protocols, two players play a game in which the prover (player one) proves to the verifier that the prover has some *special knowledge*. This special knowledge could be for example knowing a Hamiltonian tour for a graph, or a password to Ali Baba's cave. The verifier (player two) does not possess the special knowledge, nor does he learn it by means of the protocol. Thus, zero-knowledge protocols are convincing but yield nothing beyond the validity of the assertion proven (in the example "the prover knows a Hamiltonian tour") [17, 15, 3, 7].

The type of knowledge that can be proven, is limited to knowledge within a mathematical context: the two players in a protocol *a priori* know some x, and the prover proves his knowledge of some special object y. The object x may be a public key and y the corresponding private key, or x may be a graph and y the Hamiltonian tour of it, as in the example. The required mathematical relation between x and y is, loosely spoken, that it is NP-hard to compute y from x. It might seem that the requirement of a specific mathematical relation between x and y somehow restricts the possible applications of zero-knowledge protocols.

In this paper we show that we can create an NP-hard "puzzle" on the fly to prove knowledge of any y, *provided that the verifier also knows* y *a priori*. If the verifier does not know y a priori, he does not gain any information which helps him to compute y. Stated equivalently: This paper presents the first zero-knowledge protocols in which possession of *any* kind of knowledge can be proven. The knowledge need not be special in any mathematical or contextual way. The assertion "the prover

knows y" can only be verified if the verifier also knows (all of) y. The verifier never learns anything more than the prover's knowledge of y.

This new type of protocols has applications where securely comparing secrets allows transactions which could not be allowed otherwise. For example, secret agents might like to test each other's knowledge without exposing their own. Many examples can be found where privacy requirements or non-disclosure requirements are an obstruction for performing righteous tasks.

The type of problem the protocol solves is similar to, but different from, the problem described by Fagin et al. in [12]. We will first give a description which is broad enough to cover both problems, after which we will describe why our new type of protocols solves a fundamentally different problem.

By a secret, we mean information possessed by an agent a, of which agent a is not willing to share it with another agent. Whether other agents indeed possess this information as well is not relevant for it being a secret (of agent a). Here follows the problem "Comparing Information Without Leaking It" (CIWLI)[1]:

> Two players want to test whether their respective secrets are the same, but they do not want the other player to learn the secret in case the secrets do not match.

Not specified yet is *which* secrets are to be compared, and how it is *decided* which secrets are to be compared. Do the two players each take a specific secret into their mind which they compare? e.g. Is "the person I voted for" equal to "the person you voted for"? Or does one player take a secret "The General will attack tomorrow at noon" and does the other player see whether he knows this specific secret as well? In the former case, the two players first have to agree upon what they want to compare. I call this CIWLI "with reference". In the latter case, no a priori agreement is needed and I call it CIWLI "without reference", because of its lack of an agreement which refers to a secret.

CIWLI "With reference" is symmetric in the sense that both players have a specific secret in mind while performing the protocol, whereas in CIWLI "without reference", only one of the players has one specific secret in mind.

An example of CIWLI with reference is the Socialist Millionaires' problem, in which two players want to test their riches for equality, but do not want to disclose their riches to the other player [18, 8]. Another example is that two managers each have received a complaint about a sensitive matter, know this of one another, and would like to compare whether the complainer is the same person (without contacting the complainer) [12]. Solutions exist for CIWLI with reference [12, 8, 18]. In [12] a series of interesting applications is listed where protocols solving this problem could be used.

It could also be the case that it is not clear what the secret is about. In that case, we have CIWLI without reference. For example, Alice could have a file on her hard disk, and would like to know whether Bob possesses the same file as well. Alice can not naively show the file to Bob and ask him to search for a matching file, because this will obviously result is Bob obtaining the file (though Bob could be honourable and delete it voluntarily). In cases of CIWLI with reference, it is common that *two* specific secrets are tested for equality, whereas in cases without reference, one specific secret is tested against *numerous* secrets for equality. The file-comparison problem would be a case with reference if the two players would like to know whether two *specific* files are equal. ("Are the instructions you got from Carol the same as the instructions I got from Carol?")

This paper presents a solution for CIWLI without reference. It only assumes the existence of collision-free one-way hash functions [10, 16]. A more efficient solution, which depends on encryption as well as on collision-free one-way hash functions, is also shown. In a forthcoming paper, we will present results on CIWLI without reference in which the *intersection* of two or more *groups* of

[1] This is a slight variation from [12].

secrets can be computed, without leaking the secrets. This is also called the *list intersection problem*[20]. This will make it possible to create indexes on distributed, secured databases, which can be searched without leaking information on the contents of the databases. This will be similar to, but much more advanced than the approaches in [14, 13].

In CIWLI with reference, a commitment is required of both parties that their inputs to the protocol satisfy the reference, i.e. they are truthful. (e.g. in the socialist millionaires' problem that the inputs correspond to the wealth of the players.) In fact, these protocols can only be used to test whether the two inputs are equal, and only assuming truthfulness one can say something about, for example, the riches of the players. Furthermore, it is required that player A cannot infer anything on the input of player B, in case their inputs do not match. This includes that it should not be possible for player A to test the input of player B for likely values, that is to guess and verify whether the guess is correct. This is called semantic security [26, 27][2]. The semantic security is important in CIWLI with reference, because what is tested is not whether the other player can *imagine* or *guess* some input [25], but whether he actually *states* the input. Thus, cases with reference should withstand guessing attacks.

In case of CIWLI without reference, there is no need to withstand guessing attacks of the players. Basically this is because cases without reference test whether the other player possesses a specific file, which is roughly equivalent to being able to *imagine* or *guess* it within the limits of its storage capacity and computational resources. In fact, the protocol we describe in this paper is based on the fact that a player can verify the other player's knowledge of a file by correctly "guessing" it. Semantic security is still required in the sense that if a player cannot guess the *complete* input of the other player, he should not be able to infer *anything* of the input of the other player. And, of course, there must be full semantic security with respect to eavesdroppers: third persons other than the two players.

Regarding truth for CIWLI without reference, a player can always fake not possessing a certain file, while he actually does possess the file. A player can however never fake possessing something which he does not possess (or only with negligible probability).

It may need notice that CIWLI problems are very different from card deal problems such as Van Ditmarsch's Russian cards problem [11]. Firstly, in CIWLI the number of "cards" is unlimited in number, and it is not publicly known which "cards" exist. Secondly, in CIWLI there is no such thing as *exclusive* possession of a "card".

Throughout this paper we will sometimes loosely use the verb "knowing X" where we technically mean "possessing information X, which may be false", because *knowledge* is a more intuitive notion for the examples.

The protocols we describe assume the players voluntarily want to prove their knowledge to the other player. In some circumstances it can be detected when a player does not adhere to this intention. This will be addressed in sections 5.3 and 5.4.

In section 2, we will further narrow down our problem description, and we will elaborate on our assumptions in section 3. Section 4 presents three of our new protocols, which will be thouroughly analysed in section 5. In section 6 we will show more efficient versions of the protocols, without making them any weaker. We conclude with a discussion, conclusion and sketch of our future reseatch on the issue.

[2]Informally, an encryption scheme is semantically secure, if cyphertexts leak no information about the plaintext.

2 Problem description

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 known to him, is also known to Peggy. ("Peggy is kindly invited for a dinner at the Mallory's place.")[3] 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, Victor would like to keep his warning for himself, as well as his knowledge of Peggy's invitation. Therefore, Victor asks Peggy to prove her knowledge of the invitation. Only after the proof, Victor will disclose his warning to Peggy.

Peggy is willing to prove her knowledge of the invitation, but only if she can make sure Victor does not cheat on her, and actually finds out about the invitation because he tricks her into telling him (she has been invited). That is, she only wants to prove her knowledge of the invitation if Victor actually knew about the invitation beforehand.

The protocol described in section 4 facilitates the described situation. Both Victor and Peggy can initiate the protocol. 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. The protocol could however easily be used for Victor to prove his knowledge to Peggy afterwards, or even in parallel.

A situation where such *mutual* verification could be used in real life is "cautious gossip". Alice and Bill would like to gossip about the pregnancy of Georgia, but wouldn't want the to be the one to tell the other Georgia is indeed pregnant. Therefore, it is not allowable just to ask "Did you know Georgia is pregnant?". Only after mutually establishing both Alice and Bill know of Georgia's pregnancy, they can start gossiping.

For the sake of clarity, we will focus mainly on the simpler, asymmetric protocol throughout this paper. The reasoning is however easily extendible to the more symmetric variant of the protocol.

From here on I will call pieces of information "information blocks", or IB's for short. A bit more formally, we arrive to this description:

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

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

3 Protocol prerequisites and assumptions

We assume the communication channel cannot be modified by an adversary, and that it is authenticated [23, 2, 10, 24].

The protocols depend heavily on two important cryptographic functions. If you are not very familair with cryptography, you may skim over this paragraph. The two functions are:

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

keyed padding function A function *pad*(M, n), where M is a message consisting of l bits, and n is a collection of nonces N_i wit corresponding sizes l_i (again in bits). It should generate a new message M_2, based on M and the collection n, in such a way that at least l bits in M_2 depend on M and on all nonces $N_i \in n$: any single change to M or a nonce $N_i \in n$ must result in a different M_2. The entropy of M_2 should be (at least) the entropy of $M \cup n$.

Many ways exist to implement a function satisfying these properties. A simple implementation is this one:

1. Set M_2 to an empty message, append the number l to M_2.
2. For each nonce $N_i \in n$, append l_i and N_i to M_2.
3. XOR M with all the nonces in n, and append this M to M_2.

one-way hash A collision-free one-way hash function *hash*(M) [21, 9, 10, 28, 2].

The way these functions are used is *hash*(*pad*(M, n)). If a player knows $h_1 =$ *hash*(*pad*(M, n)), and this player is required to compute $h_2 =$ *hash*(*pad*(M, n')), where $n \neq n'$, he *must* know M. It is precisely this property that will be exploited in the protocols.

This use may seem equivalent to the notion of a message authentication code (MAC), or of a keyed hash[2] *MAC*(M, n), but *hash*(*pad*(M, n)) is a stronger notion. Many MAC's allow a player to compute $h_2 =$ *MAC*(M, n') from certain intermediary states of the computation of $h_1 =$ *MAC*(M, n), without knowing M. Hereby players would be allowed to bypass the strict requirement to know M.[4]

In all noninterrupted runs of the presented protocols, two hash values will be computed:

$$H_1 = \textit{hash}(\textit{pad}(M, n)) \quad (1)$$

$$H_2 = \textit{hash}(\textit{pad}(M, n')) \quad (2)$$

If a player knows only one equation with all variables but M, he cannot infer M since *hash*$(.)$ is one-way.

In a noninterrupted run, both players always know n, n', H_1 and H_2. Thus we have two equations (1) and (2) with only one variable, the message M. If the protocol is run multiple times using the same M, numerous instances of these equations might be collected[5], so we may well have many formulas with only one variable. However, a requirement of the protocols is that no player (nor any eavesdropper) may learn anything from which he feasibly can compute any property of M. This means we want the following statement to be true:

> It is infeasibly expensive to compute any information on M, no matter how many instances of $H =$ *hash*(*pad*(M, n)) with H and n known are available.

If the hash function is not vulnerable to differential cryptoanalysis[5], the desired statement is true. Luckily, resistance to differential cryptoanalysis is a very common if not standard requirement for the design of cryptographic one-way functions. The one-way hash SHA-1 [1] is believed not to be vulnerable to this attack.

[4] In fact, the ability to recompute a MAC from intermediary states is often rightfully considered a *feature* rather than a problem. In [4] this is called *incrementality*.

[5] Though the protocol may be run multiple times with the same M, it will be impossible to detect this from the protocol runs alone. However, even assuming that it can be detected or properly guessed what protocol runs share the same M, M cannot be deduced, as shown.

1. Peggy chooses an IB $I_P \in KB_P$ of which she wants to prove her knowledge to Victor
2. Peggy sends Victor the message $\{H_1 = \mathit{hash}(\mathit{pad}(I_P, \{N\}))\}$
3. Victor computes $I_V\star \subseteq KB_V$
4. Victor does one of the following:
   - Victor generates a random challenge C such that it discriminates within $I_V\star$, and sends Peggy the message $\{C\}$
   - Victor sends Peggy the message $\{\mathit{halt}\}$ and the protocol is halted
5. Peggy sends Victor the message $\{H_2 = \mathit{hash}(\mathit{pad}(I_P, \{N, P, C\}))\}$
6. Victor verifies whether H_2 (received from Peggy) is equal to any $\mathit{hash}(\mathit{pad}(I_{V_j}, \{N, P, C\}))$, 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 1: The protocol where Peggy initiates

4 Protocol description

The collections of IB's possessed by Peggy the *P*rover and Victor the *V*erifier are KB_P and KB_V, respectively. P Is a unique representation of Peggy's identity, such as her full name and birth date, or something like her passport number. Peggy and Victor have agreed upon a commonly known secret nonce N beforehand. The purpose and necessity of this nonce will be addressed in section 5.5. Upon first time reading, the option of the responding player to halt the protocol may be ignored. In section 5.4 the purpose of this option will be explained.

Both Peggy and Victor may initiate the protocol, this is shown in figure 1 and figure 2 respectively. The set $I_a\star$ is the set of IB's in possession of agent a, of which $\mathit{hash}(\mathit{pad}(I_a, \{N\}))$ is equal to H_1. If a set $I_a\star$ is empty, agent a 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.

If there is more than one IB in the set $I_i\star$, an (external[22]) collision of the hash function has occurred. This is highly improbable, but not impossible. In such a case Victor wants to discriminate between the members of the set. He can do this making sure his challenge C yields a different hash $\mathit{hash}(\mathit{pad}(I_V, \{N, P, C\}))$ for each element I_V of $I_V\star$. Ensuring this is easy because it is extremely unlikely for two IB's A and B that both $\mathit{hash}(A)$ and $\mathit{hash}(B)$ clash and that $\mathit{hash}(\mathit{pad}(A, \{C\}))$ and $\mathit{hash}(\mathit{pad}(B, \{C\}))$ clash as well.[6] In practice, Victor may choose a C at random and check for security's sake whether there are new clashes, and choose another C if this would be the case. This whole process of generating the challenge makes sure each possible H_2 corresponds to exactly one I_{V_j} in $I_V\star$. In the figures, we summarize this process as "generating a random challenge such that it discriminates".

Depending on the output size of the hash function, and the degree of confidence to be transferred, generating the challenge and verification of H_2 might be repeated a number of times, making it a

[6] Or stated otherwise: if this would not be extremely unlikely, this would be a a very severe problem of the supposedly one-way hash function.

1. Victor chooses an IB $I_V \in KB_V$ of which he wants to test Peggy's knowledge
2. Victor computes $I_V\star \subseteq KB_V$ and generates a random challenge C such that it discriminates within $I_V\star$
3. Victor sends Peggy the message $\{H_1 = \mathit{hash}(\mathit{pad}(I_V, \{N\}))$,$C\}$
4. Peggy generates $I_P\star \subseteq KB_P$
5. For each $I_{P_i} \in I_P\star$ of which Peggy is willing to prove her knowledge to Victor, Peggy sends Victor the message $\{H_2 = \mathit{hash}(\mathit{pad}(I_{P_i}, \{N, P, C\}))\}$
6. For each H_2 received from Peggy, Victor verifies whether H_2 is equal to any $\mathit{hash}(\mathit{pad}(I_{V_j}, \{N, P, C\}))$, 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}.

Figure 2: The protocol where Victor initiates

multiple-step interactive protocol.

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_P. Therefore, the challenge C should be unpredictable to the prover, because it makes such a scenario infeasible. The challenge is there to assure the verifier the prover does not present precomputed, stored values.

Figure 3 shows the symmetric variant of the protocol, in which both players prove their possession of the involved IB. Since both players both prove and verify, their names are changed into the role-neutral names Alice and Bob. In the figure, Bob waits to send H_{2_B} until Alice has sent H_{2_A}. It may well be the case that Bob is willing to send H_{2_B} as early as directly after step 4. However, before step 6 there might still be some ambiguity as to which IB Alice wants to have proven. Waiting until after step 7 prevents Bob from unnecessarily proving possession of IB's Alice did not mean to refer to.

5 Analysis and properties of the protocols

The prover will not know whether the verifier is convinced. However, the verifier could in turn prove his knowledge of I_P to the prover.

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. It is the player not initiating the protocol that has to do this. There is only a negligible probability of false completions because the probability of guessing two corresponding hashes without the knowledge of the input is negligibly small.

The protocol satisfies a number of properties common to zero-knowledge protocols. I will prove these properties heuristically by two cases depending on whether verifier already knows I_P beforehand. Furthermore, I will analyse what the results are of "faking" players. Then, I will make some comments on the possibilities for an eavesdropper. In a forthcoming paper we will present a formal proof.

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
2. Alice computes $I_A\star \subseteq KB_A$ and generates a random challenge C_A such that it discriminates within $I_A\star$
3. Alice sends Bob the message $\{H_1 = \mathit{hash}(\mathit{pad}(I_A, \{N\})), C_A\}$
4. Bob computes $I_B\star \subseteq KB_B$
5. Bob does one of the following:
   - Bob generates a random challenge C_B such that it discriminates within $I_B\star$, and sends Alice the message $\{C_B\}$
   - Bob sends Alice the message $\{\mathit{halt}\}$ and the protocol is halted
6. Alice sends Bob the message $\{H_{2_A} = \mathit{hash}(\mathit{pad}(I_A, \{N, A, C_B\}))\}$
7. Bob verifies whether H_{2_A} (received from Alice) is equal to any $\mathit{hash}(\mathit{pad}(I_{B_i}, \{N, A, C_B\}))$, 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)
8. If Bob is willing to prove his knowledge of I_B to Alice, Bob sends Alice the message $\{H_{2_B} = \mathit{hash}(\mathit{pad}(I_B, \{N, B, C_A\}))\}$
9. Alice verifies whether H_{2_B} (received from Bob) is equal to $\mathit{hash}(\mathit{pad}(I_A, \{N, B, C_A\}))$ (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 3: The symmetric protocol. Since both players both prove and verify, their names are changed into the more role-neutral names Alice and Bob. A And B are unique representations of Alice's and Bob's identity. Some variation is possible in the order of the steps, this is explained in the text.

5.1 Case 1: The verifier already knows I_P beforehand

All the following properties hold in this case:

The verifier cannot learn anything from the protocol The verifier already knows I_P, so leaking of I_P is of no issue. The verifier *does* learn the prover knows I_P.

The prover cannot cheat the verifier The verifier can verify whether the prover knows I_P, because the prover needs I_P to be able to compute $\mathit{hash}(\mathit{pad}(I_P, \{N, P, C\}))$.

The verifier cannot cheat the prover The only communication the verifier performs is sending a challenge to the prover. Since this challenge may be anything, the verifier cannot "harm" the prover.

The verifier cannot pretend to be the prover to any third party The verifier *can* prove knowledge of I_P to a third party, but he could also have done this before executing the protocol with the prover. So in this sense, this really is an irrelevant requirement.

5.2 Case 2: The verifier did not know I_P beforehand

All the following properties hold in this case:

The verifier cannot learn anything from the protocol The verifier only hears *hash*(*pad*$(I_P, \{N\})$) and *hash*(*pad*$(I_P, \{N, P, C\})$), and it is computationally infeasible to compute any information on I_P from these two hashes. Therefore learning the two hashes can be considered equal to learning two encrypted messages, which is learning nothing of any value.

The prover cannot cheat the verifier When the verifier computes the collection $I_V\star$, it will not contain an I_{V_j} equal to I_P. In practice, $I_V\star$ will be empty, in which case the verifier cannot even guess what the prover wants to prove. In the unlikely case $I_V\star$ is not empty, the verifier will guess that the prover wants to prove possession of one of the members of $I_V\star$. However, since $I_P \notin I_V\star$, the verifier will not receive any H_2 corresponding to any member in $I_V\star$.

The verifier cannot cheat the prover The verifier only hears two hash samples, and the prover will not provide more than these two hash samples (or the number of agreed upon cut-and-choose rounds). From these samples only, he cannot deduce I_P.

In case the verifier initiates the protocol, he also possesses $H_1 =$ *hash*(*pad*$(I_V, \{N\})$). This might suggest to the prover that the verifier knows I_V and thereby the verifier can compute *hash*(*pad*$(I_V, \{N\})$), but this is not necessarily the case. The verifier may have made up H_1, or may have learned H_1 from some third party. Nevertheless, if the verifier does not have the corresponding I_V, the verifier cannot verify the prover's knowledge of I_V. Actually, while the verifier might be making up his values of H_1, so might the prover make up his values of H_2.

The verifier cannot pretend to be the prover to any third party The third party will pose a new challenge C' to the verifier and require him to compute and show *hash*(*pad*$(I_P, \{N, P, C'\})$), which he can not.

The verifier cannot perform a man-in-the-middle attack, because the verifier only receives *hash*(*pad*$(I_P, \{N, P, C'\})$) from the prover, which has the identity of the prover incorporated. From this he cannot compute *hash*(*pad*$(I_P, \{N, V, C'\})$).[7], which must have the identity of the verifier (V) incorporated.

5.3 Faking players

The protocols prescribe the players to perform certain computations, and to post some of the results of those computations to the other player. This raises the question what would happen if one or both of the players actually does not post the results of the prescribed calculations, but some other information, possibly just made up.

The information to be sent to the other player consists of hashes values and challenges, and these "look like" uniformly distributed noise. Thus, instead of sending such a hash, a player could generate some random noise and send it to the other player. This is what I call a "faking player". The player receiving such faked messages cannot recognize they are fake by syntax.

A player which is not faking, only initiates a protocol if the IB corresponding to H_1 actually exists, and only sends a response H_2 if $I_a\star$ is nonempty. Obviously, this means that upon receiving an H_1 or H_2 a player may not assume any corresponding IB exists. Next to not making false assumptions, a player must gracefully handle these faked messages.

[7]The inclusion of the identity of the prover is very similar to the method used by Lowe in [19] to fix the Needham-Schröder public key authentication protocol.

The chances of a faked hash colliding with the hash of an existing IB is negligible, because the range of possible hash values is many orders of magnitude greater than the number of IB's ever possessed by a player. The result of this is that a faked H_1 will result in an empty $I_a\star$ (with a possibility negligibly close to 1). A faked H_2 will give no match with any $\mathit{hash}(\mathit{pad}(I_{V_j}, \{N, P, C_V\}))$, where $I_{V_j} \in I_V\star$ (again with a possibility negligibly close to 1).

Thus, due to the way the protocol works, a player needs to take no further action to gracefully handle faked messages. Faked messages result in protocol runs which do not convince the verifier.

In certain circumstances, it is possible to detect a player is faking. The prover may send a fake H_2, while the verifier possesses the IB which the protocol is run on. The verifier will detect a mismatch between the expected and the received H_2. Given that a good hash function with a large hash size is used, the possibility of the H_2 being non-fake is extremely small, because it would require a hash collision of two IB's, one residing at each player. The protocol nevertheless copes with such an unlikely hash collision. A verifier which often or always responds with a H_2 has either a huge number of colliding hash values, or is faking extensively.

5.4 The purpose of halting the protocol

In all protocols where the responding player also verifies, the responding player has the option to halt the protocol, instead of sending a challenge. Obviously, refusing to send a challenge and halting the protocol will result in no player being convinced of certain knowledge of the other player.

Alternatively, when the protocol is not halted, what happens to the knowledge and meta-knowledge of the players is much more complex. The verifier will be convinced, but only if the verifier possesses the relevant IB. The prover cannot infer whether the verifier gets convinced. The verifier might tell the prover whether he got convinced, but the prover has to trust this answer: The verifier might claim he was not convinced, while he actually was convinced by the protocol run. When the verifier was not convinced, this means he did not possess the relevant IB. If the verifier did not possess the relevant IB, $I_V\star$ (or $I_B\star$) was empty, or at least with a probability negligibly close to 0.

Before receiving any H_2 from the prover, the verifier can obviously claim he is not yet convinced. If the verifier wants to credibly claim he is not convinced, nor can be convinced, he can halt the protocol. Halting the protocol before sending the challenge prevents the prover from sending a H_2.

In the symmetric protocol, Bob could verify the possession of Alice, while in the meantime Bob could withhold Alice of the proof of Bob's possession. Bob could do this by sending no H_{2_B} or a fake H_{2_B}. In this situation, Alice cannot infer whether Bob has actually verified Alice's possession.

Obviously, it may be desirable that the symmetric protocol is an all-or-nothing protocol: either nobody gets convinced and this is known to both players, or both players get convinced and this is known to both players. Halting can play a crucial role here if Bob adopts a certain "halting strategy": if Bob has an empty $I_B\star$, he halts the protocol. In that case nobody gets convinced and this is known to both players. In case $I_B\star$ is nonempty, Bob sends a challenge and commits himself to truthfully proving I_B, which he can, with a probability negligibly close to 1. If Bob acts according to this commitment, both players get convinced and this is known to both players. If on the other hand Bob does not act according to this commitment, Bob is still convinced, but Alice detects Bob has broken his commitment.

If Alice and Bob do not jointly require a specific halting strategy for Bob, Bob may "lure" Alice into proving her possession, while Bob verifies Alice's possession. However, if Alice and Bob agree on the above mentioned halting strategy for Bob, the symmetric protocol is all-or-nothing, to the degree that Alice will detect any violation by Bob.

5.5 The eavesdropper learns nothing

How much an eavesdropper can learn from a protocol run depends on the secrecy of the nonce N and on whether the eavesdropper knows I_P beforehand. If the eavesdropper does not know the secret of the prover I_P beforehand, he cannot deduce any information on it, just like the verifier cannot.[8] If the eavesdropper does not know N beforehand, he cannot link any message from the prover to an IB.

The prover and the verifier cannot always make sure the eavesdropper does not possess I_P, but they can make sure the eavesdropper does not possess N, by secretly generating and agreeing upon a N. They could use a coin-flipping protocol [6, 23] for this, or a trusted third party to generate a N. If the prover and the verifier were to use a fixed, publicly known N, an eavesdropper could only learn something if he already knows I_P, which is in most cases quite unlikely. This means there might be applications in which N being publicly known would be acceptable. (Note that using a publicly known nonce N is equivalent to using no nonce at all.)

More generally, it is possible to make IB's unguessable to outsiders by adding (i.e. concatenating) random noise to an IB at the time it is created. Another IB will then only be equal to it if it matches the random noise as well. However, another agent can then only possess the same IB if it has gotten the IB by some (possibly indirect) way from the creator of the IB. In this scheme, an IB can only have one creator, and cannot be dynamically generated on the spot, because the generator cannot guess the random noise. Thus, only if the applications creating the IB's can practically be adapted to add noise, and IB's always have one unique creator, a publicly known nonce is allowable.

Another way of securing the system against an eavesdropper is to make sure an eavesdropper will not observe the hashes. This is described in the next paragraph.

6 Making the protocol more efficient by encryption

The computation of $I_a\star$ has a time complexity of $O(\mathit{size}(KB_a) + |KB_a|)$, where $\mathit{size}(KB_a) = \sum_{I_a \in KB_a} \mathit{size}(I_a)$, $\mathit{size}(I_a)$ is the number of bits in I_a, and $|KB_a|$ is the number of I_a's in KB_a. Note that this time complexity essentially is the space required to store all IB's.

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

1. Precomputing a look-up table of size $O(|KB_a|)$ 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 $O(\mathit{size}(KB_a) + |KB_a|)$.

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 $O(^2\log |KB_a|)$.

If an agent learns a new IB I_a, this agent has to update the look-up table, which has a time complexity of $O(^2\log |KB_a| + \mathit{size}(I_a))$. How to initialise and maintain the look-up table is described in figure 4.

Computing a look-up table and performing the protocol once, has the same time 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 re-use the look-up table as far as it is 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

[8] We here elaborate on the protocol where the prover initiates. The analysis for the protocols where the verifier initiates or where the secret is mutually proven, is analogous. I_P Should be replaced with I_V or I_A respectively.

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 IB's are files, this would typically be the file name.) Make the table efficiently searchable on at least the *hash* column.
2. For each IB $I_a \in KB_a$, compute *hash*(*pad*$(I_a, \{N\})$), and insert (*hash*(*pad*$(I_a, \{N\})$), *location*(I_a)) into the table. (Computing the hash value has a time complexity of *size*(I_a).)
3. With each modification of personal knowledge, update the look-up table:
   (a) For each added IB I_a, insert (*hash*(*pad*$(I_a, \{N\})$), *location*(I_a)) into the table.
   (b) For each removed IB I, remove (*hash*(*pad*$(I_a, \{N\})$), *location*(I_a)) from the table.
   (c) Consider each modified IB an old IB to be removed, and a new IB to be added.

Figure 4: The initialisation and maintenance of the look-up table, needed by any non-initiating party of the protocol

generate such a look-up table, which is by far the most expensive part of the protocols described in this paper so far.

So 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 described in section 5.5: 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 adjusted protocol for the case where Peggy initiates is shown in figure 5.

The parts that need to be encrypted are those of which an eavesdropper could either infer the involved IB[9], or could verify the proof. To prevent inferral of the involved IB, H_1 should be encrypted. To 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.

If none of C and H_2 are encrypted, an eavesdropper has to solve the equation

$$H_2 = \textit{hash}(\textit{pad}(I_P, \{P, C\}))$$

which has only one unknown variable, I_P. If the eavesdropper possesses I_P, he could guess it correctly. Withholding either C or H_2 from the eavesdropper makes it an equation with two unknown variables, which cannot be solved. 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, i.e. the player that initiated the protocol. Thus only the initiator needs to be able to send encrypted messages.

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] With "infer", we mean "properly guess" as described in the introduction.

1. Peggy chooses an IB $I_P \in KB_P$ of which she wants to prove her knowledge to Victor
2. Peggy sends Victor the message $\{\textit{encrypt}(H_1 = \textit{hash}(\textit{pad}(I_P, \emptyset)))\}$
3. Victor decrypts the message from Peggy and obtains H_1
4. Victor looks up $I_V\star$ in his look-up table: $I_V\star$ is the group of IB's whose corresponding *hash* column in the table is equal to H_1. Victor generates a random challenge C such that it discriminates within $I_V\star$
5. Victor sends Peggy the message $\{C\}$
6. Peggy sends Victor the message $\{\textit{encrypt}(H_2 = \textit{hash}(\textit{pad}(I_P, \{P, C\})))\}$
7. Victor decrypts the message from Peggy and obtains H_2
8. Victor verifies whether H_2 (received from Peggy) is equal to any $\textit{hash}(\textit{pad}(I_{V_j}, \{P, C\}))$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes I_P equals the matching I_{V_j} from $I_V\star$, and thereby verifies Peggy knows I_{V_j}.

Figure 5: The protocol where Peggy initiates, and encryption is used

7 Discussion

The protocols described use one-way hash functions in a way that has not been shown before, namely to identify files, i.e. to "point at them". We also use one-way hash functions for verification of possession. This latter use is also described in [23]:

> "If you want to verify someone has a particular file (that you also have), but you don't want him to send it to you, then you ask him for the hash value. If he sends you the correct hash value, then it is almost certain that he has that file."

Unfortunately the avobe described procedure does not guarantee possession of the particular file: It is trivial to inform someone about hash values of numerous files, without giving the person the files. This person can now "prove" possession of the files he does not have. Of course this is very undesirable, and that is what the challenges C in the described protocols are for: The proving player can only prove possession by computing $\textit{hash}(\textit{pad}(M, n))$ with $C \in n$. Maintaining hash value look-up tables for all possible challenges is not feasible, and therefore presenting a value equal to $\textit{hash}(\textit{pad}(M, n))$ proves the possession of a player.

The quote above does not mention how to determine which file is the *particular file*. Verification of possession can only be done if the file has been *pointed at*. Sending H_1 in the protocols can be interpreted as pointing at a file of which possession will or should be proved.

Maintaining a look-up table with only one, or only a few possible values for n, is feasible. This feasibility can be used to compute $I_a\star$, which is "finding out which file has been pointed at by the other player".

The need for a strong nonce has also been acknowledged for message authentication, specially by Tsudik [24]. In his paper he describes an authentication protocol which has some similarities with the

protocols presented in this paper. However, his protocol does not include pointing at a file, since the file is supposed to be known publicly. He also thoroughly elaborates on some keyed padding functions to be used in a MAC, though the functions he proposes have been shown insufficient in [22]. In [4] this problem is more or less solved. It should nevertheless be stressed that but the notion of a MAC is not strong enough for the purposes presented in this paper. More elaboration on the requirements and design considerations of keyed padding functions is required.

In [20], Naor and Pinkas present a way to solve the list intersection problem, and a way to solve special case of it: the one-to-many intersection. This one-to-many intersection problem is very similar though not equivalent to CIWLI without reference. The protocol they present is very expensive, and it leaks some meta-information. The amount of communication required is of a complexity of $O(|KB_V|)$, and the protocol leaks $|KB_V|$ to the other player. ($|KB_a|$ is the number of secrets held by player a.) Moreover, the precomputations required have a complexity of $O(|KB_a|^2 \mathit{size}(KB_a))$, which is dramatically worse than our solution. It effectively prohibits $|KB_a|$ to be large.

If in the presented protocols the prover has signed its messages, the verifier can prove to a third party knowing I_P, that the prover did try to prove the prover's knowledge of I_P to somebody.

8 Conclusion

We have given protocols which allow a player to prove his knowledge of a secret to another player, without leaking the secret in case the verifier did not know the secret. Both the prover and the verifier can initiate such a protocol. Also, a protocol is given in which two players mutually prove their knowledge. This mutual protocol has the property that it is all-or-nothing: either no player can verify the knowledge of the other, or both players know the secret and prove this to the other. Any violation of this mutual protocol can easily be detected.

Two sets of protocols are presented: one group that meets all these requirements solely by the use of one-way hash functions, and one group that meets all these requirements by also using encryption.

All protocols require a precomputation by the non-initiating player of the protocol. A precomputation has a time complexity equal to the space required to store all the player's secrets. The protocols only using one-way hashes require one precomputation for each different communication partner. The protocols also using encryption require only one single precomputation ever.

The protocols themselves take a constant number of communcation steps, and require computations by each player of a time complexity of at most $O(\mathit{size}(I))$, where I is the secret being proven or verified.

9 Future research

We have just finished building a working implementation of the described protocols. It allows users to test one another's possession of files over the internet. This software and its documentation can be found at `http://www.ai.rug.nl/~woutr/provingsecrets/`

Our attention now focuses on two issues. Firstly, we want to prove the properties of the presented protocols in a more formal way. Secondly, we intend to use the described protocols as a means to solve the list intersection problem [20] in a very efficient way.

The Dutch police offers us a very interesting application area for our protocols. Police investigation teams typically want to keep their files secret, but *do* want to know whether other teams are investigating on the same persons or locations. It is intended that our protocols will be used in this application area.

References

[1] Proposed federal information processing standard for secure hash standard. *Federal Register*, 57(21):3747–3749, 1992.

[2] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Cryptographic hash functions: A survey. Technical Report 95-09, Department of Computer Science, University of Wollongong, July 1995.

[3] M. Bellare and O. Goldreich. On defining proofs of knowledge. *Lecture Notes in Computer Science*, 740:390–420, 1993.

[4] Mihir Bellare, Roch Guerin, and Phillip Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. *Lecture Notes in Computer Science*, 963, 1995.

[5] Eli Biham and Adi Shamir. Differential cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer (extended abstract). *Lecture Notes in Computer Science*, 576:156+, 1991.

[6] M. Blum. Coin flipping by telephone: A protocol for solving impossible problems. *Proceedings of the 24th IEEE Computer Conference (COMPCON)*, pages 133–137, 1982.

[7] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 103–112, Chicago, Illinois, 2–4 May 1988.

[8] F. Boudot, B. Schoenmakers, and J. Traoré. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111(1-2):23–36, 2001.

[9] R. Canetti, D. Micciancio, and O. Reingold. Perfectly one-way probabilistic hash functions (preliminary version). In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 131–140, Dallas, 1998.

[10] I.B. Damgård. Collision free hash functions and public key signature schemes. In D. Chaum and W.L. Price, editors, *EUROCRYPT*, volume 304 of *Lecture Notes in Computer Science*, pages 203–216, Berlin, 1988. Springer.

[11] H.P. van Ditmarsch. The russian cards problem. *Studia Logica*, 75:31–62, 2003.

[12] R. Fagin, M. Naor, and P. Winkler. Comparing information without leaking it. *Communications of the ACM*, 39(5):77–85, 1996.

[13] J. Feigenbaum, E. Grosse, and J.A. Reeds. Cryptographic protection of membership lists. *Newsletter of the International Association for Cryptologic Research*, 9(1):16–20, 1992.

[14] J. Feigenbaum, M. Liberman, and R. Wright. Cryptographic protection of databases and software, 1991.

[15] O. Goldreich. Zero-knowledge twenty years after its invention. Technical report, Department of Computer Science, Weizmann Institute of Science, 2002.

[16] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proofs. *JACM*, 38:691–729, 1991.

[17] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *Proceedings of the seventeenth annual ACM symposium on Theory of Computing*, pages 291–304, 1985.

[18] M. Jakobsson and M. Yung. Proving without knowing: On oblivious, agnostic and blindfolded provers. *Lecture Notes in Computer Science*, 1109:186–200, 1996.

[19] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.

[20] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 245–254, 1999.

[21] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing. (May 15–17 1989: Seattle, WA, USA)*, pages 33–43, New York, NY 10036, USA, 1989. ACM Press.

[22] B. Preneel and P.C. van Oorschot. MDx-MAC and building fast MACs from hash functions. *Lecture Notes in Computer Science*, 963:1–14, 1995.

[23] B. Schneier. *Applied Cryptography*. John Wiley & Sons, New York, 1996.

[24] G. Tsudik. Message authentication with one-way hash functions. In *INFOCOM (3)*, pages 2055–2059, 1992.

[25] Y. Watanabe, J. Shikata, and H. Imai. Equivalence between semantic security and indistinguishability against chosen ciphertext attacks. *Lecture Notes in Computer Science*, 2567:71–84, 2003.

[26] A.C. Yao. Protocols for secure computations. *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[27] A.C. Yao. How to generate and exchange secrets. *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[28] Y. Zheng, T. Mashumoto, and H. Imai. Provably secure one-way hash functions.

A The remaining protocols with encryption

The paper did not show the protocol where the verifier initiates the protocol, nor the one where both players prove their possession. Though hese protocols can be deduced relatively easily, we fully describe these 2 protocols for reference. These protocols are described in figures 6 and 7 respectively.

B Acknowledgements

The author would like to thank Kathy Cartrysse, Rineke Verbrugge, Rafael Accorsi Marius Bulacu, and the anonymous referees for their feedback and numerous comments on drafts of this paper and the ideas therein.

1. Victor chooses an IB $I_V \in KB_V$ of which he wants to test Peggy's knowledge
2. Victor computes $I_V\star \subseteq KB_V$ and generates a random challenge C such that it discriminates within $I_V\star$
3. Victor sends Peggy the message $\{\textit{encrypt}(\{H_1 = \textit{hash}(\textit{pad}(I_V, \emptyset)), C\})\}$
4. Peggy decrypts the message from Victor and obtains H_1 and C
5. Peggy generates $I_P\star \subseteq KB_P$
6. For each $I_{P_i} \in I_P\star$ of which Peggy is willing to prove her knowledge to Victor, Peggy sends Victor the message $\{H_2 = \textit{hash}(\textit{pad}(I_{P_i}, \{P, C\}))\}$
7. For each H_2 received from Peggy, Victor verifies whether H_2 is equal to any $\textit{hash}(\textit{pad}(I_{V_j}, \{P, C\}))$, 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}.

Figure 6: The protocol where Victor initiates, and encryption is used

C Paraphrasing the protocol

With thanks to my office mate, Marius Bulacu, I can present you a rough paraphrase of the type of protocols described in this paper. It is very sketchy, but illustrates the non-intuitivity of the protocols in an intuitive way. Consider this conversation by A and B:

A Hey! You know what?

B Huh, What?

A Well, you know, don't you?

B I don't know what you are talking about

A Well, nevermind

This could be considered a rough real-world equivalent of an unsuccessful protocol run. Finally, consider this "successful" protocol run:

A Hey! You know what?

B Huh, What?

A Well, you know, don't you?

B Ahh, yeah, of course

A Thank you, goodbye

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
2. Alice computes $I_A\star \subseteq KB_A$ and generates a random challenge C_A such that it discriminates within $I_A\star$
3. Alice sends Bob the message $\{\textit{encrypt}(H_1 = \textit{hash}(\textit{pad}(I_A, \emptyset)), C_A\})\}$
4. Bob decrypts the message from Alice and obtains H_1 and C
5. Bob computes $I_B\star \subseteq KB_B$
6. Bob does one of the following:
   - Bob generates a random challenge C_B such that it discriminates within $I_B\star$, and sends Alice the message $\{C_B\}$
   - Bob sends Alice the message $\{\textit{halt}\}$ and the protocol is halted
7. Alice sends Bob the message $\{\textit{encrypt}(H_{2_A} = \textit{hash}(\textit{pad}(I_A, \{A, C_B\})))\}$
8. Bob decrypts the message from Alice and obtains H_{2_A}
9. Bob verifies whether H_{2_A} (received from Alice) is equal to any $\textit{hash}(\textit{pad}(I_{B_i}, \{N, A, C_B\}))$, 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 willing to prove his knowledge of I_B to Alice, Bob sends Alice the message $\{H_{2_B} = \textit{hash}(\textit{pad}(I_B, \{B, C_A\}))\}$
11. Alice verifies whether H_{2_B} (received from Bob) is equal to $\textit{hash}(\textit{pad}(I_A, \{B, C_A\}))$ (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 7: The symmetric protocol with encryption.