WOUTER TEEPE

PROVING POSSESSION OF ARBITRARY SECRETS WHILE NOT GIVING THEM AWAY: NEW PROTOCOLS AND A PROOF IN GNY LOGIC

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. 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. We extend and use the GNY authentication logic to prove correctness of these protocols.

1. INTRODUCTION

In zero-knowledge protocols, two players play a game in which the prover (player one) proves to the verifier (player two) 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") (Goldwasser et al. 1985; Blum et al. 1988; Bellare and Goldreich 1993; Goldreich 2002).

The type of knowledge that can be proven in zero-knowledge protocols, is limited to knowledge within a mathematical context: the two players in a protocol know some x *a priori*, 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, speaking loosely, 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.

Synthese (2006) 149: 409–443
Knowledge, Rationality & Action 155–189
DOI 10.1007/s11229-005-3879-4

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. Or 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 that our protocol solves is similar to, but different from, the problem described in Fagin et al. (1996). 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? For example, 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 (Jakobsson and Yung 1996; Boudot et al. 2001). 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) (Fagin et al. 1996). Solutions exist for CIWLI with reference (Fagin et al. 1996; Jakobsson and Yung 1996; Boudot et al. 2001). In Fagin et al. (1996) 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 cannot naively show the file to Bob and ask him to search for a matching file, because this will obviously result in 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 assumes the existence of collision-free one-way hash functions (Damgård 1988; Goldreich et al. 1991). 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 secrets can be computed, without leaking the secrets. This is also called the *list intersection problem* (Naor and Pinkas 1999). 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 Feigenbaum et al. (1991, 1992).

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. (For example, in the socialist millionaires' problem this means 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 (Yao 1982, 1986).[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 (Watanabe et al. 2003), 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 (van Ditmarsch 2003). Firstly, in CIWLI the number of "cards" is unlimited, 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 often 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 that the players voluntarily want to prove their knowledge to the other player.

In Section 2, we present our new protocols for CIWLI without reference, and informally explain how and why they work. In Section 3, we introduce authentication logics, explain why we use GNY logic (Gong et al. 1990), why we need to extend GNY logic, and how we extend GNY logic. Section 4 contains a proof of our protocols in GNY logic.

2. NEW PROTOCOLS

In this section, we will introduce our solution to the problem of CIWLI without reference. First, we will sketch a scenario in Section 2.1, and explain what primitives we use in our solution in Section 2.2. We describe two solutions in Sections 2.3 and 2.4, the former using only hash functions, the latter also using encryption.

2.1. *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. In the protocol, Peggy does not learn whether Victor actually knew about the invitation, other than from his possible next actions, such as sending a warning.

Peggy is willing to prove her knowledge of the invitation, but only if she can make sure that Victor does not cheat on her, and actually finds out about the invitation because he tricks her into

telling him that she has been invited. That is, she only wants to prove her knowledge of the invitation if Victor actually knew about the invitation beforehand.

Actually, this description only describes one of three possible configurations of the protocol.

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

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 to be the one to tell the other that 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.

In this paper we will mainly focus on case 1, though we stress that the proof for case 1 can easily be modified to prove the protocols for the other cases. For a more extended description of the protocols for all three configurations, see Teepe (2004a).

From here on I will call pieces of information "information blocks", or IB's for short. Here follows a somewhat more formal 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 that Peggy indeed has it, but if Victor does not have the same IB, he does not learn anything.

2.2. *Protocol Prerequisites and Assumptions*

We assume that the communication channel cannot be modified by an adversary, and that it is authenticated (Damgård 1988; Tsudik 1992; Bakhtiari et al. 1995; Schneier 1996). It is not relevant in what way the authentication is established. However, in Section 4, we will fill in a means for authentication, since GNY logic does not allow us to simply assume authentication without explicitly providing it.

Apart from authentication purposes, the only cryptographic primitive we use is the *one-way collision-free hash function*. This is a primitive that is one-way: it transforms a message into a fingerprint of fixed size in such a way that it is infeasible to infer any property of the message from the fingerprint. Moreover, this primitive is collision-free, which means that it is hard to find two inputs $X, Y, X \neq Y$ where $H(X) = H(Y)$ (Damgård 1988; Naor and Yung 1989; Zheng et al. 1990; Bakhtiari et al. 1995; Schneier 1996; Canetti et al. 1998).

When we apply a hash function in a formula, we use the notation $H(\cdot)$. We often loosely use a notation where $H(\cdot)$ has multiple arguments, in those cases we mean the hash of the *concatenation* of the arguments.[4]

An extra requirement is that for any two inputs X, Y, if $X \neq Y$, it should be hard to compute $H(Y)$, if $H(X)$ and the difference between X and Y is given. Thus, *any* modification to the input of the hash function, requires computations with a time complexity of at least the size of the input. Only the slightest modification in the input should force a full recomputation of the hash function.

Standard cryptographic hash functions, like MD5 and SHA-1 (National Institute of Standards and Technology 1992) do not satisfy this requirement. In fact, some cryptographic hash functions are even designed to be efficiently recomputable if only a small part of the input changes, like MDx-MAC (Bellare et al. 1995). However, using standard cryptographic hash functions, we can build cryptographic hash functions that *do* satisfy the extra requirement of necessary recomputation.

The solution is to perform some padding of the input of the hash function. What type of padding is required depends on the details of the hash function that is used. For hash functions based on a compression function, it suffices to repeat the message twice, and to use this as the input to the hash function. For other classes of cryptographic hash functions, it may be necessary to modify at least one in every few bits, by applying an XOR to the input. A detailed description of how to achieve this for different classes of cryptographic hash functions is out of the scope of this paper.

The way in which such an "extra secure" hash is used is $h_1 = H(M, n)$, i.e. to compute hash of the concatenation of a message M and some information n. If a player claims to know the M which corresponds to h_1, the player will be asked to compute $h_2 = H(M, n')$, where $n \neq n'$. Since $n \neq n'$, the player has to fully recompute the function, and therefore the player *must* know M. It is

precisely this property that will be exploited in the protocols. (The parameter n serves as a way to enforce recomputation.)

This use may seem equivalent to the notion of a message authentication code (MAC), or of a keyed hash (Bakhtiari et al. 1995), but $H(M, n)$ is a stronger notion. Many MAC's allow a player to compute $h_2 = MAC(M, n')$ from certain intermediate 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.

2.3. *Protocol Description*

There are three configurations of the protocol, as mentioned in Section 2.1. The corresponding protocols are shown in Figures 1 (the verifier initiates), 2 (the prover initiates), and 3 (mutual proof).

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 $\{h_1 = H(I_V, N), C\}$
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 $\{h_2 = H(I_{P_i}, N, P, C)\}$
6. For each h_2 received from Peggy, Victor verifies whether h_2 is equal to any $H(I_{V_j}, N, P, C)$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes that I_{P_i} equals the matching I_{V_j}, and thereby verifies that Peggy knows the matching I_{V_j}.

Figure 1. The protocol where Victor the *v*erifier initiates.

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 $\{h_1 = H(I_P, N)\}$
3. Victor computes $I_V\star \subseteq KB_V$, and does one of the following:
   – If $I_V\star$ is nonempty, Victor generates a random challenge C such that it discriminates within $I_V\star$, and sends Peggy the message $\{C\}$
   – If $I_V\star$ is empty, Victor sends Peggy the message $\{halt\}$ and the protocol is halted
4. Peggy sends Victor the message $\{h_2 = H(I_P, N, P, C)\}$
5. Victor verifies whether h_2 (received from Peggy) is equal to any $H(I_{V_j}, N, P, C)$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes that I_P equals the matching I_{V_j}, and thereby verifies that Peggy knows I_{V_j}.

Figure 2. The protocol where Peggy the *p*rover initiates.

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 $\{h_1 = H(I_A, N), C_A\}$
4. Bob computes $I_B\star \subseteq KB_B$, and does one of the following:
   – If $I_B\star$ is nonempty, Bob generates a random challenge C_B such that it discriminates within $I_B\star$, and sends Alice the message $\{C_B\}$
   – If $I_B\star$ is empty, Bob sends Alice the message $\{halt\}$ and the protocol is halted
5. Alice sends Bob the message $\{h_{2_A} = H(I_A, N, A, C_B)\}$
6. Bob verifies whether h_{2_A} (received from Alice) is equal to any $H(I_{B_i}, N, A, C_B)$, where $I_{B_i} \in I_B\star$ (locally computed). If they are equal, Bob concludes that I_A equals the matching I_{B_i}, and thereby verifies that Alice knows the matching I_{B_i} (which we will call I_B from here on)
7. If Bob is willing to prove his knowledge of I_B to Alice, Bob sends Alice the message $\{h_{2_B} = H(I_B, N, B, C_A)\}$
8. Alice verifies whether h_{2_B} (received from Bob) is equal to $H(I_A, N, B, C_A)$ (locally computed). If they are equal, Alice concludes that I_A equals I_B, and thereby verifies that Bob knows the matching I_A.

Figure 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.

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. In this article, a nonce is a piece of information with the sole purpose to be unpredictable to anybody but Peggy and Victor.

The protocol in the configuration we analyse in this paper, where Victor initiates the protocol, is shown in Figure 1. The set $I_a\star$ is the set of IB's I in possession of agent a, for which $H(I, N)$ is equal to h_1. Thus, $I_a\star = \{I \in KB_a | H(I_a, N) = h_1\}$. 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.

It is extremely unlikely that there will be more than one IB in the set $I_i\star$. However, the protocol can easily cope with the situation if it would occur.[5] If this protocol is widely adopted and applied, it can be expected that *somewhere* this situation will occur. If the protocol could not handle this situation well, data corruption would be the result. Therefore, the ability to handle such unlikely situations still is an important feature.

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

Without the nonce N in the protocol, any eavesdropper who happens to know I, can analyse and interpret the protocol, which is undesirable. When the eavesdropper does not know the N, this analysis and interpretation is no longer possible. In the next section we further elaborate on eavesdroppers and their abilities to interpret messages of this protocol.

In typical applications of one-way hashes, the input to the hash is more or less public knowledge. This protocol on the other hand exploits the fact that the input may *not* be publicly known. Successful completion depends on one of the players being able to "invert" the one-way hash, since it knows the original input to the hash function. To make sure that eavesdroppers cannot learn from observing the protocol, it has to be ensured that eavesdroppers do not know the full input to the hash function. That is the purpose of the nonce N: to achieve semantic security.

2.4. *Making the Protocol more Efficient by Encryption*

The computation of $I_a\star$ has a time complexity of $O(size(KB_a) + |KB_a|)$, where $size(KB_a) = \sum_{I_a \in KB_a} size(I_a)$, $size(I_a)$ 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 be 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 $O(size(KB_a) + |KB_a|)$.
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 $O(^2\log |KB_a| + size(I_a))$. How to initialise and maintain the look-up table is described in Figure 4.

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 $H(I_a, N)$, and insert $(H(I_a, N), location(I_a))$ 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 $(H(I_a, N), location(I_a))$.
   b) For each removed IB I, remove $(H(I_a, N), location(I_a))$.
   c) Consider each modified IB as 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 player of the protocol.

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

Therefore, we can improve dramatically on speed if we can find a way to safely re-use nonces, or to use no nonces at all. The reason to use nonces was to make sure we have semantic security with respect to any third party observing the conversation. Semantic security can also be achieved by means of encryption of some crucial parts of the protocol. The adjusted protocol is shown in Figure 5.

The parts that need to be encrypted are those of which an eavesdropper could either infer the IB,[6] or could verify the proof. To prevent inferral of the 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. 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.

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 $\{encrypt(\{h_1 = H(I_V), C\})\}$
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 $\{h_2 = H(I_{P_i}, P, C)\}$
7. For each h_2 received from Peggy, Victor verifies whether h_2 is equal to any $H(I_{V_j}, P, C)$, where $I_{V_j} \in I_V\star$ (locally computed). If they are equal, Victor concludes that I_{P_i} equals the matching I_{V_j}, and thereby verifies that Peggy knows the matching I_{V_j}.

Figure 5. The protocol where Victor the *v*erifier initiates, and encryption is used.

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.

3. THE GNY AUTHENTICATION LOGIC

In this section, we explain what tools we use to prove our solution to CIWLI without reference correct. Section 3.1 introduces BAN logic (Burrows et al. 1990) and its enhancement GNY logic (Gong et al. 1990), and explains what these logics can offer and what they cannot offer us. Section 3.2 explains in what ways we will apply GNY, and introduces the concepts of *knowledge preconditions* and *invalidators*, which are central parts of our proof. Section 3.3 summarizes the language of GNY logic. Section 3.4 elaborates on inference rules of the GNY logic for *knowledge authentication*, and adds new inferences rules to GNY logic which will be needed in our proofs. Finally, Section 3.5 addresses the issue of how to prove that principals can *not* learn specific things during a protocol run.

3.1. *BAN, AT and GNY Logic, and its Value*

BAN logic (Burrows et al. 1990), introduced by Burrows, Abadi and Needham, is a means for analysing security protocols, and

especially authentication protocols. Using BAN logic helps to determine the subtle assumptions a protocol depends on, and to investigate whether the protocol is *correct*. BAN logic has succesfully been applied to find undetected flaws in protocols. Unfortunately, BAN logic has no well-defined semantics. Abadi and Tuttle (1991) have introduced a slightly modified version of this logic (AT logic), which has a semantics. However, in both BAN logic and AT logic, there is no clear distinction between possession and belief: an agent cannot possess a formula without believing it. Gong, Needham and Yahalom (Gong et al. 1990) modified the BAN logic into an elegant logic which, among other features, clearly distinguishes between possession and belief. This logic is commonly referred to as GNY logic. Unfortunately, GNY logic does not have a clear semantics as AT logic does. These three logics are generally considered sound (detected flaws are indeed flaws) but certainly not complete (some flaws may remain undetected).

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

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

All three of the abovementioned logics mainly address properties 1 and 2. If an analysis of a protocol using one of the mentioned logics does not expose a flaw, this means that no flaws exist that violate properties 1 and 2, of course assuming that the logic itself is "correct", whatever that may be. However, some silly protocols that for example disclose keys to the public, are not tagged defective by these logics (see for example, Nessett 1990).

If one wants to prove property 3, that principals can *not* infer specific facts, one has to model the limitations of these agents, and show that the limitations effectively obstruct principals from inferring these facts. To model the inference limitations of principals, we need to model what inference rules are available to an agent. This is where a nasty property of the abovementioned logics comes in: none of the authors of these logics claim that the list of inference rules provided in the logic is indeed complete in the sense that no more inference rules can be added. Assuming that the list of given inference rules is complete is a necessary step in proving property 3 of a protocol. We do not believe nor claim that completeness of the

inference rule list is sufficient for proving property 3 of a protocol. This issue will be discussed in Section 3.5.

Lacking a means to prove property 3, the meaning of a correctness proof in one of the mentioned logics is only limited. If the use of one of these logics exposes a flaw in a protocol, the protocol will indeed be flawed. However, not finding any errors does not guarantee that the protocol is correct. Therefore, we say that proving a protocol correct using one of these logics, only proves that the protocol has passed a first test of some not-so-obvious flaws. Nevertheless, we deem a proof in a BAN-style logic an important step in defending correctness of protocols. For ease of speech, from here on we use "correctness" as an abbreviation for "correct regarding to BAN-style logic".

3.2. *Our Use of GNY Logic*

In this paper, we will use and extend GNY logic to prove the secret prover protocols correct. The secret prover protocols are different from normal authentication protocols in that the outcome of the protocol should depend on the knowledge precondition of the principals: if one of the principals does not know the secret before the protocol run, he will not learn (any part of) the secret during the protocol. Thus, the correctness of the protocol should critically depend on the truth value of the knowledge precondition. This kind of proof assumes the list of inference rules in the logic is not only sound, but also more or less complete. To be more precise, the list of rules should be so complete that whatever "undiscovered" rules exist, they do not change the knowledge precondition dependency.

We argue that the list of inference rules of GNY logic is however not complete enough to prove any of the secret prover protocols correct. Due to the rather unprecedented use of hash functions, we need an inference rule to reflect this use.

The main issue of the protocols is whether the prover can cheat by asking someone else to compute the proof in name of the prover, and just forward this proof. Making someone different from the prover compute the actual proof can be achieved by either a successful man-in-the-middle attack by the prover, or by a willing assistant of the prover which *does* have the knowledge referred to in the knowledge precondition. We should design protocols in such a way that successful man-in-the-middle attacks do not exist. BAN-like logics help in analysing the existence of such attacks. However,

we cannot fight willing assistants of provers. In some sense, this is also unnecessary, since the goal of the secret prover protocols is to test whether the prover has effective access to the secret, and one may reasonably claim the prover indeed has such access if she has an assistant who will perform computations on the secrets on behalf of the prover.

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

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

3.3. *Summary of GNY Logic*

In this section, we summarize the logic as presented in Gong et al. (1990). Parts of the logic we do not use are omitted. When describing a protocol, we adhere to the notation $P \rightarrow Q$: message, denoting that the principal P sends the message and that principal Q receives it. The message consists of a formula in GNY logic.

3.3.1. *Formulae in GNY Logic*

A formula is a name used to refer to a bit string, which would have a particular value in a run. Let X and Y range over formulae, and $+K$ and $-K$ over public keys and private keys respectively. The following are also formulae:

(X, Y): conjunction of two formulae. We treat conjunctions as sets with properties such as associativity and commutativity.

$\{X\}_{+K}$ and $\{X\}_{-K}$: public-key encryption and public-key decryption. We assume a cryptosystem for which $\{\{X\}_{+K}\}_{-K} = X$ holds (i.e. encryption), and for which also $\{\{X\}_{-K}\}_{+K} = X$ holds (i.e. signatures, e.g. RSA).
$H(X)$: a one-way cryptograpic hash function of X.
$*X$: a not-originated-here formula. A formula X is a not-originated-here formula if a principal receives X without having sent X itself before. Thus, a formula is a not-originated-here formula for a principal P, if it is not a replay of one of P's previously sent messages.

3.3.2. *Statements in GNY Logic*
Statements reflect properties of formulae. Let P and Q be principals. The following are basic statements:
$P \lhd X$: P *is told* formula X. P receives X, possibly after performing some computation such as decryption.
$P \ni X$: P *possesses,* or is capable of possessing, formula X. At a particular state of a run, this includes all the formulae P has been told, all the formulae he started the session with, and all the ones he has generated in that run. In addition P possesses, or is capable of possessing, everything that is computable from the formulae he already possesses.
$P \mid\sim X$: P *once conveyed* formula X. X can be a message itself or some content computable from such a message, i.e. a formula can be conveyed implicitly.
$P \models \sharp(X)$: P *believes*, or is entitled to believe, that formula X is *fresh*. That is, X has not been used for the same purpose at any time before the current run of the protocol.
$P \models \phi(X)$: P *believes*, or is entitled to believe, that formula X is *recognizable*. That is, P would recognize X if P has certain expectations about the contents of X before actually receiving X. P may recognize a particular value or a particular structure.
$P \models P \overset{S}{\leftrightarrow} Q$: P *believes,* or is entitled to believe, that S is a suitable *secret* for P and Q. They may properly use it to mutually prove identity. They may also use it as, or derive from it, a key to communicate. S will never be discovered by any principal except P and Q.
$P \models \overset{+K}{\mapsto} Q$: P *believes,* or is entitled to believe, that $+K$ is a suitable *public key* for Q, i.e. the matching secret key $-K$ will never be discovered by any principal except Q.

Let C range over statements. The following are also statements:

$P \models C$: *P believes,* or is entitled to believe, that formula *C holds*.
C_1, C_2: conjunctions. We treat conjunctions as sets with properties such as associativity and commutativity.

3.4. *New Inference Rules*

In this section we will discuss several methods for *knowledge authentication*, proving that you know something. For these methods, we show and discuss the corresponding inference rules in GNY logic, and where necessary introduce new inference rules that we add to GNY logic. A well known method for knowledge authentication is to sign the message you want to prove knowledge of, and then to show this signed message. Obviously this method cannot be used in a setting where the message itself should be kept secret, because the message will be disclosed when showing the signed message. Nevertheless it is interesting to look at the inference rule corresponding to interpreting signed messages, shown below (from Gong et al. 1990).

$$\textbf{I4} \qquad \frac{P \lhd \{X\}_{-K},\ P \ni +K,\ P \models \overset{+K}{\mapsto} Q,\ P \models \phi(X)}{P \models Q \mid\sim X,\ P \models Q \mid\sim \{X\}_{-K}}$$

What the rule says is this: If P sees a signed message $\{X\}_{-K}$, knows the public key $+K$, knows the corresponding private key $-K$ belongs to Q, and recognises X to be a message, then P is entitled to believe that Q once conveyed the signed message $\{X\}_{-K}$, and thus also once conveyed the message X itself (see Gong et al. 1990).

Important to note are two silent assumptions of this inference rule:

1. For any X, no principal Q will convey $\{X\}_{-K}$ where $Q \neq P$. Thus, Q's private key $-K$ is only known to Q and Q will never convey $-K$.[7]
2. Q will, in a sense, be conservative in what he signs: Q will only sign intentionally and with consent.[8] Q will never sign unseen messages.

The reason for assumption 2 is that a signature counts as an irrefutable commitment.

Making similar assumptions, we could introduce a new identity-related inference rule:[9]

$$\textbf{H1} \qquad \frac{V \lhd *H(X, P),\ V \ni (X, P)}{V \models P \mid\sim (X, P),\ V \models P \mid\sim H(X, P)}$$

If V sees a message $H(X, P)$, and also possesses (X, P), he is entitled to believe that P once conveyed (X, P) and $H(X, P)$.

The assumptions under which this rule is justified are these:

1. For any X, no principal Q will convey $H(X, P)$ where $Q \neq P$.
2. P will, in a sense, be conservative in the set of X's for he conveys $H(X, P)$. More specifically, he will only convey $H(X, P)$ for X's of which he wants to show other principals he possesses X.

These two assumptions tie together just like the two assumptions of rule **I4**: the first assumption states that only one principal is capable of sending certain messages, and the second states that this principal will only do so with informed consent.

However, assumption 1 of inference rule **H1** is not justifiable. Using rule **H1**, a malicious principal, knowing any secret X, can "commit" any principal P to conveying the secret X by broadcasting $H(X, P)$. To prevent malicious principals from creating havoc in this way, we should require the message sent to be authentic, i.e. that it is known who sent the message $H(X, P)$. Using such authentication, a verifier can distinguish proofs of knowledge by malicious principals from proofs by intended principals.

Assuming authentication, we would be able to introduce a more moderate rule like this one:

$$\textbf{H2} \qquad \frac{V \models P \mid\sim *H(X, P), V \ni (X, P)}{V \models P \mid\sim (X, P)}$$

If V sees a message $H(X, P)$ from P, and also possesses (X, P), he is entitled to believe that P once conveyed (X, P). This effectively eliminates assumption 1 from rule **H1**.

Rule **H2** is justified under the following assumption:

1. P will, in a sense, be conservative in the set of X's for which he conveys $H(X, P)$ in an authenticated manner. More specifically, he will only convey $H(X, P)$ for X's of which he wants to show other principals that he possesses X.

This assumption is very reasonable, when one takes into account that it is a minor variation of assumption 2 of rule **I4**: P will not sign just any message, but only if she really intends to sign a message.

There is a slight technical issue with rule **H2**, which also applies to rule **H1**: How can a principal P make sure that he is not sending $H(X, P)$ in an authenticated way without actually knowing that he

is sending $H(X, P)$? For example, in a concurrently running protocol of a different kind, P might be required to sign a challenge. P cannot verify whether this challenge in fact is equal to a $H(X, P)$ if he does not possess X. This problem can be solved by adding something like a recognizable *speech act token* to the hash value that has to be signed: P would have to sign ("I know", $H(X, P)$) instead of just $H(X, P)$. The inference rule needs to be adjusted to reflect this, giving rule **H3** shown below. In such a way, P can make sure that he never accidentally signs a value that may be interpreted using inference rule **H3**.

$$\textbf{H3} \qquad \frac{V \models P \mid\sim (\text{"I know"}, *H(X, P)), V \ni (X, P)}{V \models P \mid\sim (X, P)}$$

This rule has the same assumption as **H2**, except that for this rule, P can really make sure the assumption is true. This rule requires that the language of the GNY logic be extended with *tokens*. We decide not to do this (yet), and use rule **H2**, knowing that we can easily modify protocols to reflect rule **H3** instead of rule **H2**.

3.5. *Proving that Principals Do not Learn too much*

BAN-style logics focus on establishing whether the principals interacting in a protocol draw correct conclusions in a security protocol. However, for security protocols, it is also crucial to prove that certain principals do *not* draw some specific conclusions. This should be proven for both active and passive attackers. The literature about BAN-style logics generally addresses the case of active attackers (like in the NSPK protocol, Lowe 1996), but not the case of passive attackers. A passive attack is an attack where a principal learns something that should be kept secret, while the only capabilities available to the attacker are eavesdropping and reasoning.

In this work we demonstrate a novel approach to proving that principals do not learn specific facts in the course of a protocol run. We believe that this is a significant contribution to establishing a proof of property 3 as mentioned in Section 3.1. Our approach is a new way of using BAN-style logics.

Normally, when proving a protocol using BAN-style, assumptions about the principals are stated. We introduce an extra principal and show that the principal cannot infer what should be kept secret. We call this principal Eve the *e*avesdropper. Just as with any other principal, we list assumptions about what Eve possesses

and believes at the beginning of the protocol. The meaning of the assumptions is somewhat different, however. When we state an assumption for a principal participating in the protocol, this is in some sense a weakness of the protocol: it has to be met in order for the protocol to be correct. When we state an assumption about Eve, this is a strength of the protocol: even if Eve knows or possesses this *a priori*, the protocol is still correct in the sense that Eve cannot deduce the secret. Thus, we establish the maximum amount of *a priori* beliefs and possessions Eve may have under which it is still impossible for Eve to infer the secret facts.[10] Just as with normal BAN-style proofs, the list of assumptions allows to reason about subtleties concerning the quality and applicability of a protocol.

In fact we model two properties of a passive attacker, namely (1) its beliefs and possessions, and (2) its reasoning capabilities. The reasoning capabilities are modeled by the inference rules of the logic under the assumption that this list is complete.

We make no assumptions on what role Eve takes in the protool: Eve may either be one of the participants or an external observer.

4. PROTOCOL PROOFS

In the protocols described in Section 2.3 we have two participating principals, V the *v*erifier and P the *p*rover. We assert that our protocols satisfy the following properties:

1. "The verifier learns whether P knows I, iff the verifier knows I and the prover wants to prove her knowledge of I":
   $V \models P \ni I$ holds after the protocol run, iff $P \ni I$, $V \ni I$ holds and P wants to prove possession of I before the protocol run.
2. "Only the verifier learns whether anybody knows I in course of the protocol":
   For any principal Q, except V:
   (a) $Q \models P \ni I$ holds after the protocol run, iff $Q \models P \ni I$ holds before the protocol run.
   (b) $Q \models V \ni I$ holds after the protocol run, iff $Q \models V \ni I$ holds before the protocol run.
3. "Nobody learns I in the course of the protocol":
   For any principal Q, $Q \ni I$ holds after the protocol run, iff $Q \ni I$ holds before the protocol run.

The $\Leftarrow$ part of property 1 will be proven using a formal proof, which will span Sections 4.2 and 4.3. The $\Rightarrow$ part of property 1 and property 3 will be proven in Section 4.4. Proving property 2 requires us to make some assumptions on the beliefs and possessions of an attacker. This will be done in Section 4.5.

We will give the idealised versions of the secret prover protocols. However, due to limited space we only concentrate on one configuration of the protocol: the one where the verifier asks the prover to prove possession of a specific secret. The configurations in which the prover initiates the protocol and where a mutual proof is exercised do not shed new light on the security analysis of the protocols. For the chosen configuration, two protocols exist: one which uses no encryption but depends on a shared nonce between the prover and the verifier, and one which depends on encryption from the verifier to the prover. (For an overview of what configurations exist, see Sections 2.1 and 2.4.)

In Section 4.2 we give an analysis of the protocol without encryption up to the point where the newly introduced inference rules of Section 3.4 are needed. A comparision of how these rules help to complete the proof, including the proof completion itself, is shown in Section 4.3.

In Section 2.2, it has been noted that the communication channel should be authenticated and cannot be modified by an adversary. The most important is that the last message of the protocol is clearly bound to its sender, more precisely that the receiver can verify who is the sender. For our protocols, it is not really relevant in what way this authentication is established. To keep the proofs of the protocols as simple as possible, we simply assume a specific way of authentication of the sender. We choose a public-key signature for this. This choice is not essential and if we change this authentication method, the protocol proofs can easily be adjusted to reflect this.

It may seem counter-intuitive to prove a protocol that does not use encryption by assuming signatures, which essentially is a special case of encryption. However we would like to stress that this is just the easiest way to prove the protocol. The issue is that we do not have to assume encryption for our protocols to work, but only sender authentication.

4.1. *Notation*

When referring to inference rules, we use as rule identifier the letter-cipher combination of the rule as used in the appendix of Gong

et al. (1990). These rules are repeated in this paper in Appendix A. When performing inferences using rules, we state the rule identifier, and refer to its satisfied preconditions by mentioning the line numbers of the preconditions between parentheses.

We also test whether the principal is indeed able to perform the actions prescribed in the protocol. Obviously we need this, because we want to show that the prover cannot convey a valid proof if he does not know the secret. Therefore, we also annotate the protocol communication steps with (1) the communication step number between square brackets, and (2) the line number in which it was shown that the performing principal indeed possesses the information he utters, between parentheses.

4.2. *Protocol Idealisation of the Secret Prover Protocol without Encryption*

In this and the next section, we will give a constructive correctness proof of the protocol as described in Section 2.3 and Figure 1. Due to space considerations, the proof for the protocol that uses encryption, as described in Section 2.4 and Figure 5, is omitted. That proof is very similar to the proof we will give here. The only essential differences are that all references to the nonce N are removed, and that a few steps are inserted which take care of encryption and decryption of the first message of the protocol.

When we idealise the protocol as described in Section 2.3 and Figure 1, we get to the following description and output of the protocol parser (Gong et al. 1990, 237).

	idealised protocol	output of the protocol parser
1	$V \rightarrow P : H(I, N), C$	$P \lhd (*H(I, N), *C)$
2	$P \rightarrow V : \{H(I, N, P, C)\}_{-K}$	$V \lhd *\{*H(I, N, P, C)\}_{-K}$

The protocol assumptions are these:

A.1	$P \ni I$	A.5	$P \ni -K$	A.8	$V \ni C$
A.2	$V \ni I$	A.6	$V \ni +K$	A.9	$V \models \sharp(C)$
A.3	$P \ni P$	A.7	$V \models \overset{+K}{\mapsto} P$	A.10	$P \ni N$
A.4	$V \ni P$			A.11	$V \ni N$

Assumptions A.1 and A.2 express that the principals do indeed know the secret. Thus, these are the knowledge preconditions. Assumptions A.3 and A.4 reflect that both principals know the identity of P. Assumption A.5 expresses that the prover knows her

private key, and assumption A.6 expresses that the verifier knows the corresponding public key. Assumption A.7 reflects that the verifier believes this public key indeed corresponds to the prover's private key. Assumptions A.8 and A.9 reflect respecively that the verifier knows his own challenge and that the verifier believes its freshness. Assumptions A.10 and A.11 reflect that both principals know the nonce.

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

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

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

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

4.3. *Different Options to Complete the Proof*

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

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

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

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

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

In order to prove correctness without relying on assumption A.12, we need new inference rules. In Section 3.4 we have discussed various rules, and we will apply them here. Using rule **H1**, we can finish the protocol proofs. The proof is as follows:

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

Note that in this proof, P's identity is not used. As discussed in Section 3.4, rule **H1** is dubious. Using the better-justified rule **H2**, we can also finish the protocol proofs. The proof is as follows:

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

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

4.4. *Proving Principals Do not Learn too much*

So far, of the properties stated in Section 4, we have only proven the $\Leftarrow$ part of property 1. In this section, we will prove the the $\Rightarrow$ part of property 1, and we will prove property 2.

1. "The verifier learns whether P knows I, iff the verifier knows I and the prover wants to prove her knowledge of I'', $\Rightarrow$ part: We assume $V \models P \ni I$ holds after the protocol run (the verifier has been convinced of prover's possession of I).
   For the prover to be able to actually prove possession of I, she has to use it while constructing message 2. If she does not possess I, she cannot perform step C.6, which is necessary for C.8, which states message 2. Thus, $P \ni I$ holds before the protocol run.
   If the prover would not want the verifier to possibly learn that the prover knows I, the prover would not have sent message 2. Thus, the prover wants to prove her knowledge of I.
   For the verifier to be able to verify the proof, he has to possess I as well. More specifically, the verifier has to verify whether the message he sees in line D″.3 (or equivalently, in D.3 or D′.3) equals the value the verifier computed at line B.6.[11] Thus, $V \ni I$ holds before the protocol run.
2. "Nobody learns I in course of the protocol": We prove this by contradiction. Let us that assume principal Q does learn I by means of the protocol, and that by analysing messages Q managed to reconstruct I. I itself is never conveyed except as an argument to a one-way hash function. Thus, Q managed to invert a one-way hash function. Obviously, this is impossible. This is reflected by the fact that no inference rules exist that allow to infer that someone conveyed the arguments of a one-way hash function without knowing the arguments beforehand. Any inference rule which has $P \lhd *H(I)$ as a precondition and $V \models Q \mid\sim I$ as a conclusion, always also has $P \ni I$ as a precondition.

Except that the protocol works, it is also very efficient. Both the verifier and the prover only need to perform a bounded number of steps. The prover will, upon seeing $*H(I, N)$, look whether she

has a matching secret I. Only after establishing that she actually does, she will start further computations. The bottleneck of course is recognizing an I which matches the sent $H(I, N)$. A principal can in fact generate a look-up table in advance, which stores for each possible I the corresponding $H(I, N)$ value. This is a one-time operation whose cost is proportional to the total size of all secrets that a player wants to be able to look up. This has to be done for each value of N the principal is interested in. If however the protocol which uses encryption is used, this dependency on N disappears.

4.5. *Modeling the Beliefs and Possessions of an Attacker*

In the previous section we have shown that no principal can learn I itself from observing the protocol. However, we are also interested in anything that an eavesdropper *could* learn. Could an eavesdropper become convinced that the prover or the verifier knows I? This is what property 2 of Section 4 is about. Or, less bad but still undesirable: could an eavesdropper learn about what secret I the protocol is run?

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

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

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

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

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

5. DISCUSSION

We have built an implementation of our new protocols. It allows users to test one another's possession of files over the internet (Teepe 2004b).

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 Schneier (1996):

> "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."

An equivalent claim is made in Agray et al. (2001). Unfortunately the procedure described above 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, by presenting the hash values as "proofs". 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 $H(M, C, \dots)$. Maintaining hash value look-up tables for all possible challenges is not feasible, and therefore presenting a value equal to $H(M, C, \dots)$ proves the possession of a file by 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 (1992). 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 message authentication code (MAC), though the functions he proposes have been shown insufficient in Preneel and van Oorschot (1995). In Bellare et al. (1995) this problem is more or less solved. It should nevertheless be stressed that 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 Naor and Pinkas (1999), Naor and Pinkas present a way to solve the list intersection problem, and a way to solve a 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 has 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 $O(|KB_a|^2 size(KB_a))$, which is dramatically worse than our solution. It effectively prohibits $|KB_a|$ to be large.

Freedman, Nissim and Pinkas also present a way to solve the list intersection problem (Freedman et al. 2004). The cryptographical primitive their solution uses is the homomorphic asymmetric encryption scheme. Their solution requires the expensive computation of many polynomials, and becomes infeasible if the secrets to be compared are large, say thousands or millions of bytes long.

6. 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.

The protocols require a precomputation by one of the two participants of the protocol. The precomputation has a time complexity

equal to the space required to store all that player's secrets. The protocol which only uses one-way hashes requires one precomputation for each different communication partner. The protocol which also uses encryption requires only one single precomputation.

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

We have proven these protocols correct using an extended version of GNY logic (Gong et al. 1990), for which we had to introduce new inference rules **H2** and **H3**. The latter rule is a slight modification of the former, but requires a small extension of the GNY language. **H3** has a sound justification to add it to the list of inference rules of GNY logic. Protocols and proofs using **H2** can however easily be modified to use **H3** instead of **H2**. Moreover, we have shown that the protocol does not leak any valuable information to observers, assuming completeness of the list of available inference rules in the logic.

The relevance of this paper is fourfold. First, we have presented protocols that solve the problem of comparing information without leaking it. Secondly, we have extended the list of allowable inference rules in GNY logic. Third, making a few assumptions, we have introduced a way to prove that some facts cannot be learned by active and passive principals. Fourth, we have used these results to prove correctness of the secret prover protocols.

In our ongoing work, we plan to make our results presented here more generically applicable to security protocols. Moreover, we will extend the GNY language to include speech act tokens. Also, we intend to use the described protocols as a means to solve the list intersection problem (Naor and Pinkas 1999) 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.

APPENDIX A: LOGICAL POSTULATES

For completeness, we repeat all inference rules from Gong et al. (1990) used in this paper. The rule names correspond to the names

used in the original article. Some of the inference rules (**P2**, **F1**, **I3**) have more allowable conclusions than used in this paper. These unused conclusions are omitted.

A postulate that applies to formula X also applies to $*X$, though not necessarily vice versa. If $\frac{C1}{C2}$ is a postulate, then for any principal P, so is $\frac{P \models\!\equiv C1}{P \models\!\equiv C2}$.

A.1. *Being-told Rules*

T1 $\quad \dfrac{P \lhd *X}{P \lhd X}$

Being told a "not-originated-here" formula is a special case of being told a formula.

T2 $\quad \dfrac{P \lhd (X, Y)}{P \lhd X}$

Being told a formula implies being told each of its concatenated components.

T6 $\quad \dfrac{P \lhd \{X\}_{-K},\ P \ni +K}{P \lhd X}$

If a principal is told a formula encrypted with a private key and he possesses the corresponding public key then he is considered to have also been told the decrypted contents of that formula. This rule only holds for public-key systems with the property $\{\{X\}_{-K}\}_{+K} = X$ (e.g. RSA).

A.2. *Possession Rules*

P1 $\quad \dfrac{P \lhd X}{P \ni X}$

A principal is capable of possessing anything he is told.

P2 $\quad \dfrac{P \ni X,\ P \ni Y}{P \ni (X, Y)}$

If a principal possesses two formulae then he is capable of possessing the formula constructed by concatenating the two formulae. This rule used is inductively.

P3 $$\frac{P \ni (X, Y)}{P \ni X}$$

If a principal possesses a formula then he is capable of possessing any one of the concatenated components of that formula.

P4 $$\frac{P \ni X}{P \ni H(X)}$$

If a principal possesses a formula then he is capable of possessing a one-way computationally feasible hash function of that formula.

P8 $$\frac{P \ni -K, P \ni X}{P \ni \{X\}_{-K}}$$

If a principal possesses a formula and a private key then he is capable of possessing the decryption of that formula with that key.

A.3. *Other Rules*

F1 $$\frac{P \models \sharp(X)}{P \models \sharp(X, Y)}$$

(Freshness) If a principal believes a formula X is fresh, then he is entitled to believe that any formula of which X is a component is fresh.

R6 $$\frac{P \ni H(X)}{P \models \phi(X)}$$

(Recognizability) If P possesses a formula $H(X)$, then he is entitled to believe that X is recognizable.

I3 $$\frac{P \lhd *H(X, S), P \ni (X, S), \quad P \models P \overset{S}{\leftrightarrow} Q, P \models \sharp(X, S)}{P \models Q \mid\sim (X, S)}$$

(Message Interpretation) Suppose that for principal P all of the following hold: (1) P receives a formula consisting of a one-way function of X and S marked with a not-originated-here mark; (2) P possesses S and X; (3) P believes that S is a suitable secret for himself and Q; (4) P believes that either X or S is fresh. Then P is entitled to believe that Q once conveyed the formula X concatenated with S.

It should be noted that rule **I3** as given here differs slightly from the definiton in Gong et al. (1990), which uses the notation $\langle S \rangle$ in some places instead of S to denote that S is used for identification. This is only syntactic sugar. For readability of the proofs, these brackets have been omitted thoughout this article.

Rule **I4** has been explained in Section 3.4, and will not be repeated here.

$$\textbf{I6} \qquad \frac{P \models Q \mid\sim X,\ P \models \sharp(X)}{P \models Q \ni X}$$

(Message Interpretation) If P believes that Q once conveyed formula X and P believes that X is fresh, then P is entitled to believe Q possesses X.

ACKNOWLEDGEMENTS

The author would like to thank Hans van Ditmarsch, Sieuwert van Otterloo, the anonymous referees and my supervisor Rineke Verbrugge, for all the fruitful discussions and feedback on earlier versions of this paper.

NOTES

[1] This is a slight variation from Fagin et al. (1996).

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

[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.

[4] In this procedure, there are some technical caveats which should be obeyed when concatenating the individual arguments to one another. The concatenation should fully contain all arguments, and keep the individual arguments identifiable as such.

[5] If there is more than one IB in the set $I_i\star$, an "external" collision of the hash function has occurred (Preneel and van Oorschot 1995). 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 by making sure his challenge C yields a different hash $H(I_V, N, P, C)$ 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 $H(A)$ and $H(B)$ clash *and* that $H(A, C)$ and $H(B, C)$ clash as well. If this would not be extremely unlikely, this would be a very severe problem of the supposedly one-way hash function. 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 that 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".

[6] With "infer", we mean "properly guess" as described in Section 1.

[7] More precisely, we mean that no Q will send $\{X\}_{-K}$ before receiving $\{X\}_{-K}$: Thus, Q could perform replays of messages, but cannot generate messages signed with the key $-K$.

[8] For example, Q will not sign his own death penalty.

[9] Newly introduced inference rule's names will start with **H**, for *h*ash. Moreover, we will use V and P to denote principals, V to represent the *v*erifier and P to represent the *p*rover.

[10] This "maximum belief set" is not necessarily unique.

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

REFERENCES

Abadi, M. and M. Tuttle: 1991, 'A Semantics for a Logic of Authentication', in *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*. Montreal, pp. 201–216.

Agray, N., W. van der Hoek, and E. P. de Vink: 2001, 'On BAN Logics for Industrial Security Protocols', in B. Dunin-Kęplicz and E. Nawarecki (eds.), *Proceedings of the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems*, Cracow, pp. 29–36.

Bakhtiari, S., R. Safavi-Naini, and J. Pieprzyk: 1995, 'Cryptographic Hash Functions: A Survey', Technical Report 95-09, Department of Computer Science, University of Wollongong.

Bellare, M. and O. Goldreich: 1993, 'On Defining Proofs of Knowledge', in E. Brickell (ed.), *Advances in Cryptology: Crypto '93 Proceedings*, Vol. 740, Springer Verlag, Berlin, pp. 390–420.

Bellare, M., R. Guerin, and P. Rogaway: 1995, 'XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions', in D. Coppersmith (ed.), *Advances in Cryptology: Crypto '95 Proceedings*, *Lecture Notes in Computer Science*, Vol. 963, Springer Verlag, Berlin, pp. 15–28.

Blum, M., P. Feldman, and S. Micali: 1988, 'Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)', in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, Chicago, IL, pp. 103–112.

Boudot, F., B. Schoenmakers, and J. Traoré: 2001, 'A Fair and Efficient Solution to the Socialist Millionaires' Problem', *Discrete Applied Mathematics* **111**(1–2), 23–36.

Burrows, M., M. Abadi, and R. Needham: 1990, 'A Logic of Authentication', *ACM Transactions on Computer Systems* **8**(1), 18–36.

Canetti, R., D. Micciancio, and O. Reingold: 1998, 'Perfectly One-way Probabilistic Hash Functions (Preliminary Version)', in *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, Dallas, pp. 131–140.

Damgård, I.: 1988, 'Collision Free Hash Functions and Public Key Signature Schemes', in D. Chaum and W. Price (eds.), *EUROCRYPT*, *Lecture Notes in Computer Science*, Vol. 304, Springer Verlag, Berlin, pp. 203–216.

Fagin, R., M. Naor, and P. Winkler: 1996, 'Comparing Information Without Leaking It', *Communications of the ACM* **39**(5), 77–85.

Feigenbaum, J., M. Liberman, and R. Wright: 1991, 'Cryptographic Protection of Databases and Software', in J. Feigenbaum and M. Merritt (eds.), *Distributed Computing and Cryptography*, Vol. 2, pp. 161–172.

Feigenbaum, J., E. Grosse, and J. Reeds: 1992, 'Cryptographic Protection of Membership Lists', *Newsletter of the International Association for Cryptologic Research* **9**(1), 16–20.

Freedman, M. J., K. Nissim, and B. Pinkas: 2004, 'Efficient Private Matching and Set Intersection', in C. Cachin and J. Camenisch (eds.), *Advances in Cryptology – EUROCRYPT 2004*, *Lecture Notes in Computer Science*, Vol. 2037, Springer Verlag, Berlin, pp. 1–19.

Goldreich, O.: 2002, 'Zero-knowledge Twenty Years after its Invention', Technical report, Department of Computer Science, Weizmann Institute of Science.

Goldreich, O., S. Micali, and A. Wigderson: 1991, 'Proofs that Yield Nothing But their Validity or All Languages in NP have Zero-Knowledge Proofs', *JACM* **38**, 691–729.

Goldwasser, S., S. Micali, and C. Rackoff: 1985, 'The Knowledge Complexity of Interactive Proof-systems', in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, Providence, RI, pp. 291–304.

Gong, L., R. Needham, and R. Yahalom: 1990, 'Reasoning About Belief in Cryptographic Protocols', in D. Cooper and T. Lunt (eds.), *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society Press, Los Angeles, pp. 234–248.

Jakobsson, M. and M. Yung: 1996, 'Proving without Knowing: On Oblivious, Agnostic and Blindfolded Provers', in N. Koblitz (ed.), *Advances in Cryptology: Crypto '96 Proceedings*, *Lecture Notes in Computer Science*, Vol. 1109, Springer Verlag, Berlin, pp. 186–200.

Lowe, G.: 1996, 'Breaking and Fixing the Needham–Schroeder Public-key Protocol Using FDR', in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Vol. 1055, Springer-Verlag, Berlin, pp. 147–166.

Naor, M. and B. Pinkas: 1999, 'Oblivious Transfer and Polynomial Evaluation', in *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, ACM Press, New York, pp. 245–254.

Naor, M. and M. Yung: 1989, '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*, ACM Press, New York, pp. 33–43.

National Institute of Standards and Technology: 1992, 'Proposed Federal Information Processing Standard for Secure Hash Standard', *Federal Register* **57**(21), 3747–3749.

Nessett, D. M.: 1990, 'A Critique of the Burrows, Abadi and Needham Logic', *ACM SIGOPS Operating Systems Review* **24**(2), 35–38.

Preneel, B. and P. van Oorschot: 1995, 'MDx-MAC and Building Fast MACs from Hash Functions', in D. Coppersmith (ed.), *Advances in Cryptology: Crypto '95*

Proceedings, *Lecture Notes in Computer Science*, Vol. 963, Springer Verlag, Berlin, pp. 1–14.
Schneier, B.: 1996, *Applied Cryptography*, John Wiley & Sons, New York.
Teepe, W.: 2004a, 'New Protocols for Proving Knowledge of Arbitrary Secrets while not Giving them away', in P. McBurney, W. van der Hoek, and M. Wooldridge (eds.), *Proceedings of the First Knowledge and Games Workshop*, University of Liverpool, Liverpool, pp. 99–116.
Teepe, W.: 2004b, 'The Secret Prover', http://www.ai.rug.nl/~woutr/provingsecrets.
Tsudik, G.: 1992, 'Message Authentication with One-Way Hash Functions', in *Proceedings of IEEE INFOCOM 1992*, IEEE Computer Society Press, Los Angeles, pp. 2055–2059.
van Ditmarsch, H.: 2003, 'The Russian Cards Problem', *Studia Logica* **75**, 31–62.
Watanabe, Y., J. Shikata, and H. Imai: 2003, 'Equivalence between Semantic Security and Indistinguishability against Chosen Ciphertext Attacks', in Y. Desmedt (ed.), *Proceedings of International Workshop on Practice and Theory in Public Key Cryptosystems (PKC 2003)*, *Lecture Notes in Computer Science*, Vol. 2567, Springer-Verlag, Berlin, pp. 71–84.
Yao, A.: 1982, 'Protocols for Secure Computations', in *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Angeles, pp. 160–164.
Yao, A.: 1986, 'How to Generate and Exchange Secrets', in *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Angeles, pp. 162–167.
Zheng, Y., T. Mashumoto, and H. Imai: 1990, 'Provably Secure One-Way Hash Functions', in *1990 Workshop on Cryptography and Information Security*, Hiroshima.

University of Groningen,
Dept. of Artificial Intelligence,
Grote Kruisstraat 2/1,
9712 TS Groningen,
The Netherlands
E-mail: wouter@teepe.com