Efficient Concurrent Zero-Knowledge in the Auxiliary String Model

Ivan Damgård

Aarhus University, BRICS*
ivan@daimi.au.dk
Dept. of Computer Sience
Ny Munkegade
DK 8000 Aarhus C, Denmark

Abstract. We show that if any one-way function exists, then 3-round concurrent zero-knowledge arguments for all NP problems can be built in a model where a short auxiliary string with a prescribed distribution is available to the players. We also show that a wide range of known efficient proofs of knowledge using specialized assumptions can be modified to work in this model with no essential loss of efficiency. We argue that the assumptions of the model will be satisfied in many practical scenarios where public key cryptography is used, in particular our construction works given any secure public key infrastructure. Finally, we point out that in a model with preprocessing (and no auxiliary string) proposed earlier, concurrent zero-knowledge for NP can be based on any one-way function.

1 Introduction

In a zero-knowledge protocol [23], a prover convinces a verifier that some statement is true, while the verifier learns nothing except the validity of the assertion. Apart from being interesting as theoretical objects, it is well-known that zero-knowledge protocols are extremely useful tools for practical problems. For instance as stand-alone for identification schemes[1], but probably a more important application is as subprotocols in schemes for more complex tasks such as voting, electronic cash and distributed key generation.

Hence the applicability of the theory of zero-knowledge in real life is of extreme importance. One important aspect of this is composition of protocols, and the extent to which such composition preserves zero-knowledge. While sequential composition does preserve zero-knowledge, this is not always the case for parallel composition [22].

* Basic Research in Computer Science, Center of the Danish National Research Foundation

[1] However, the identification problem can also be solved without using zero-knowledge [2]

In [12] Dwork, Naor and Sahai pointed out that the strict synchronization usually assumed when composing zero-knowledge protocols is unrealistic in scenarios such as Internet based communication. Here, many instances of the same or different protocols may start at different times and may run with no fixed timing of messages. What is needed here is a stronger property known as *concurrent zero-knowledge*, i.e., even an arbitrary interleaving of several instances of zero-knowledge protocols is again zero-knowledge, even when the verifiers are all controlled by a single adversary, who may use information obtained from one protocol to determine its behavior in another instance.

Unfortunately, standard constructions for zero-knowledge protocols fail to provide this property. This is because they are based on simulation by rewinding the verifier. In a concurrent setting, the simulator may be forced to rewind an exponential number of times. In fact, it seems that concurrent zero-knowledge cannot be provided at all in the usual model with as few rounds as ordinary zero-knowledge. Kilian, Petrank and Rackoff [19] show that only BPP languages have concurrent zero-knowledge proofs or arguments with 4 rounds or less, if black-box simulation is assumed[2].

Thus, a lot of research has gone into finding ways of getting around this problem. In [12], it was shown that given constraints on the timing of messages[3], concurrent zero-knowledge can be achieved for all of NP in a constant number of rounds. Subsequently it was shown that the need for timing constraints could be pushed into a preprocessing phase[13]. In [10] it was shown that the timing constraints in the preprocessing can be reduced to merely ensuring th at all preprocessings are finished before the main proofs start. This comes at the price that the work needed in the preprocessing depends on the size and number of statements to be proved later. Finally, Richardson and Kilian [26] show that it is possible to do without timing constraints, at the expense of a non-constant number of rounds.

We note that a completely different approach is possible: one could go for a weaker property than zero-knowledge, one that would be preserved in a concurrent setting. One such possibility is the Witness-Hiding (WH) protocols of Feige and Shamir [15]. Most WH protocols are based on the standard paradigm of the prover proving knowledge of one of two "computationally independent" witnesses without revealing which one he knows. Such protocols are also WH when used concurrently, and can be used to construct secure identification systems. In [8], very efficient methods for building such protocols are developed. However, for more general use, e.g., as subrutines in multiparty computation or verifiable secret sharing protocols, WH is not always sufficient, one needs simulatability to prove the overall protocol secure.

[2] Virtually all known zero-knowledge protocols are black-box simulatable

[3] These constraints are much milder than strict synchronization, please refer to [12] for details

2 Our Work

Our main objective is to show that concurrent zero-knowledge can sometimes be obtained in a simple way using standard tools. We do not claim any major new techniques, in fact our solution is quite straightforward. Nevertheless, we believe it is useful to realize that in many real life scenarios, resources are available that allow achieving concurrent zero-knowledge easily.

We do not mean to suggest that our solution is always more practical than previous methods for achieving concurrent zero-knowledge in constant round, such as the timing based one from [12]. In fact the solutions are based on assumptions of very different nature. Which solution is preferable will often depend on the actual scenario in which the you want the solution to work.

Also, nothing we say here makes the theoretical work on the subject less interesting or important – a major problem, namely whether concurrent zero-knowledge for NP can be achieved in constant round *without* extra assumptions, remains open.

Independently, Kilian and Petrank [18] and Canetti, Goldreich, Goldwasser and Micali [9] have made observations similar to ours, in the case of [9] as a result of introducing a new general concept called Resettable Zero-Knowledge.

2.1 The Model

Our work starts from the following assumption: an auxiliary string with a prescribed distribution is available to the prover and verifier. Given this assumption we will see that concurrent zero-knowledge can be achieved in constant round with no timing constraints or preprocessing. Informally, zero-knowledge in such a setting means as usual that the verifiers entire view can be simulated efficiently, which here means its view of the interaction with the prover, as well as the auxiliary string. Soundness means that no polynomial time prover can cheat the verifier with non-negligible probability where the probability is taken over the choice of the auxiliary string as well as the coin tosses of the players.

More formally, an interactive argument for a language L in this model consists of a probabilistic polynomial time algorithm G, and polynomial time interactive Turing Machines P, V. The algorithm G gets as as input 1^k and outputs an auxiliary string σ. P, V then get σ and a word x of length k as input, and P gets a private input w. At the end of the interaction, V halts and outputs accept or reject. When we talk about the probability of acceptance or rejection in the following, these probabilities are taken over the coin tosses of G, P and V. Note that even when we consider cheating provers, we still assume that σ is correctly generated (by G).

As usual, a negligible function from natural to real numbers $\delta()$ is a function such that $\delta(k) \leq 1/p(k)$ for all polynomials $p(\cdot)$ and all large enough k)

Definition 1. *We say that* (G, P, V) *is an interactive argument in the auxiliary string model for language* L*, if*

- *For every $x \in L$, there is a w such that if P gets w as private input, V will accept on input x.*
- *For words $x \notin L$, and for every probabilistic polynomial time prover, the probability that V accepts input x is negligible in $|x|$.*

We have defined here for simplicity only the case where an auxiliary string is only used to prove a single statement, and where the input parameter to G is set equal to the length of the common input x. None of these restrictions are essential, and they can be ignored in a practical application.

To define zero-knowledge, we consider as usual an arbitrary probabilistic polynomial time verifier V^* that gets private auxiliary input y of length polynomial in $|x|$. We then have:

Definition 2. *For any verifier V^*, there exists a simulator M_{V^*}, such that for words $x \in L$ and arbitrary auxiliary inputs y, such that M_{V^*} runs in expected polynomial time, and the distribution of $M_{V^*}(x, y)$ is polynomially indistinguishable from the view of V^* produced from the same input (namely σ, the random coins of V^*, and the conversation with P).*

Note that the standard non-interactive zero-knowledge model (where the auxiliary string is a uniformly chosen random string) [3] is a special case, and indeed by their very nature non-interactive zero-knowledge proofs do not require rewinding to simulate, and so are robust in a concurrent setting. It is even possible to do any polynomial number of non-interactive proofs based on the same globally shared random string [14].

However, there are still several reasons why non-interactive zero-knowledge proofs are not the answer to all our problems: they are in general much less efficient than interactive ones and - as far as we know - require stronger cryptographic assumptions (trapdoor one-way permutations as opposed to arbitrary one-way functions). We would like a solution allowing us to use standard efficient constructions of protocols securely in a concurrent setting, without significant loss of efficiency.

We also need to consider proofs of knowledge in our model. For this, we use a straightforward adaptation of the definition of Bellare and Goldreich, in the version modified for computationally convincing proofs of knowledge[4]. The scenario, consisting of G, P, V is the same as before. Now, however, the language L is replaced by a binary relation R, and the prover's claim for the given common input x is that he knows w such that $(x, w) \in R$.

Definition 3. *We say that (G, P, V) is a proof of knowledge for R in the auxiliary string model, with knowledge error $\kappa()$, if*

- *If P is given w, such that $(x, w) \in R$, then V accepts.*
- *There exists an* extractor*, a machine with oracle access to the prover which on input x outputs w with $(x, w) \in R$. This extractor must satisfy the following for any probabilistic polynomial time prover and any long enough x: if the provers probability $\epsilon(x)$ of making the verifier accept is larger than $\kappa(x)$, then the extractor runs in expected time $p(|x|)/(\epsilon(x) - \kappa(x))$.*

The model we use (with a general auxiliary string) was also used in [5] (for a different purpose). The rationale for allowing a general distribution of the reference string is of course that one may hope that this allows for more efficient protocols, for examplea much shorter auxiliary string. The problem, on the other hand, may be that requiring a more powerful resource makes the model less realistic.

However, as we shall see, our protocols do in fact apply to a realistic situation, namely a public-key cryptography setting where users have public/private key pairs. In fact our prover and verifier do not need to have key pairs themselves, nevertheless, they will be able to prove and verify general NP statements in concurrent zero-knowledge by using the public key P_A of a third party A as auxiliary string. This will work, provided that

- The verifier believes that A's secret key is not known to the prover.
- The prover believes that P_A was generated using the proper key generation algorithm for the public-key system in use.

We stress that A does not need to take part in the protocols at all, nor does he need to be aware that his public key is being used this way, in particular keys for standard public key systems like RSA, El Gamal or DSS can be used directly.

Note that if we have a secure public key infrastructure where public keys are being certified by a certification authority (CA), then all our demands are already automatically satisfied because the CA can serve as player A in the above: in order for the infrastructure to be secure in the first place, each user needs to have an authentic copy of the CA's public key available, and one must of course trust that the CA generated its public key in the proper way and does not reveal its private key to anyone else.

So although our model does make stronger assumptions on the environment than the standard one, we believe that this can be reasonable: The problem of concurrent zero-knowledge arises from the need to apply zero-knowledge protocols in real situations. But then solutions to this problem should be also allowed to take advantage of resources that may exist in such scenarios.

It is important to realize one way in which our model can behave differently from the standard one: suppose a verifier shows to a third party a transcript of his interaction with the prover as evidence that the protocol really took place. Then, in our model, there are scenarios where this will be convincing to the third party (contrary to what is case with the standard model). This may in some applications be a problem because it can harm the privacy of users. We stress, however, that in the case where a public-key infrastructure exists, there are ways around this problem. We discuss this issue in more detail in Section 4.

2.2 The Results

Our first result is a construction for protocols of a particular form. Assume we have a binary relation R, and a 3-move proof of knowledge for R, where the verifier sends a random challenge as the second message. Thus this protocol gets a string x as common input for prover and verifier, whereas the prover gets as private input a witness for x, i.e. w such that $(x, w) \in R$. Conversations in the protocol are of form (a, e, z), where the prover chooses a, z. We will assume that the protocol is honest verifier zero-knowledge in the sense that given e, one can efficiently compute a correctly distributed conversation where e is the challenge. Finally we assume that a cheating prover can answer only one of the possible challenges, or more precisely, from the common input x and any pair of accepting conversations $(a, e, z), (a, e', z')$ where $e \neq e'$, one can compute a witness for x. We call this a Σ-protocol in the following. We have

Theorem 1. *Given any binary relation R and a Σ-protocol for R. If one-way functions exist, then there exists a computationally convincing and concurrent zero-knowledge 3-move proof of knowledge (with negligible knowledge error and no timing constraints) for R in the auxiliary string model.*

The construction behind this result can be applied in practice to the well known proofs of knowledge of Schnorr [21] and Guillou-Quisquater [16] to yield concurrent zero-knowledge proofs of knowledge in the auxiliary string model with negligible loss of efficiency compared to the original protocols (which were not even zero-knowledge in the usual sense!). The idea behind this result also immediately gives:

Theorem 2. *If one-way functions exist, there exist 3-move concurrent zero-knowledge interactive arguments in the auxiliary string model (with no timing constraints) for any NP problem.*

In both these results, the length of the auxiliary string is essentially the size of the computational problem the prover must solve in order to cheat. The length does not depend on the size or the number of statements proved.

Our final result is an observation concerning the preprocessing model of Dwork and Sahai [13] (where there is no auxiliary string). It was shown in [13] that prover and verifier can do a once-and-for-all preprocessing (where timing constraints are applied), and then do any number of interactive arguments for any NP problem in concurrent zero-knowledge (with no timing constraints) in 4 rounds. This was shown under the assumption that one-way trapdoor permutations exist. Below, we observe the following:

Theorem 3. *If any one-way functions exists, then any NP problem has a 3-round concurrent zero-knowledge argument in the preprocessing model of Dwork and Sahai.*

We note that our preprocessing is once-and-for-all, like the one in [13]: once the preprocessing is done, the prover and verifier can execute any polynomial number of proofs securely, and the complexity of the preprocessing does not depend on the number or size of the statements proved.

3 The Protocols

3.1 Trapdoor Commitments Schemes

In a commitment scheme, a committer C can commit himself to a secret s chosen from some finite set by sending a *commitment* to a reciever R. The receiver should be unable to find s from the commitment, yet C chould be able to later open the commitment and convince R about the original choice of s.

A *trapdoor commitment scheme* is a special case that can be loosely described as follows: first a public key pk is chosen based on a security parameter value k, usually by R by running a probabilistic polynomial time *generator* G. Then pk is sent to C. There is a fixed function *commit* that C can use to compute a commitment c to s by choosing some random input r, and setting $c = commit(s, r, pk)$. Opening takes place by revealing s, r to R, who can then check that $commit(r, s, pk)$ is the value he received originally.

We then require the following:

Hiding: For a pk correctly generated by G, uniform r, r' and any s, s', the distributions of $commit(s, r, pk)$ and $commit(s', r', pk)$ are polynomially indistinguishable (as defined in [23]).

Binding: There is a negligible function $\delta()$ such that for any C running in expected polynomial time (in k) the probability that C on input pk computes s, r, s', r' such that $commit(s, r, pk) = commit(s', r', pk)$ and $s \neq s'$ is at most $\delta(k)$.

Trapdoor Property: The algorithm for generating pk also outputs a string t, the trapdoor. There is an efficient algorithm which on input t, pk outputs a commitment c, and then on input any s produces r such that $c = commit(s, r, pk)$. The distribution of c is poynomially indistinguishable from that of commitments computed in the usual way.

In other words, the commitment scheme is binding if you know only pk, but given the trapdoor, you can cheat arbitrarily.

From the results in Shamir et al.[20], it follows that existence of any one-way function f implies the existence of a trapdoor commitment scheme, where the public key is simply $f(y)$, where y is chosen uniformly in the input domain of f, and y is the trapdoor. Based on standard intractability assumptions such as hardness of discrete log or RSA root extraction, very efficient trapdoor commitment schemes can be built, see e.g. [6].

3.2 A construction for Σ-protocols

In what follows, we will assume that we have a relation R and a Σ-protocol $\mathcal{P}$ for R. The prover and verifier get as common input x, while the prover gets as private input w, such that $(x, w) \in R$.

We will be in the auxiliary string model, where the auxiliary string will be the public key pk of a trapdoor commitment scheme, generated from security parameter value $k = |x|$. For simplicity, we assume that the commitment scheme allows to commit in one commitment to any string a, that may occur as the first message in $\mathcal{P}$ (in case of a bit commitment scheme, we could just commit bit by bit). Finally, note that since the properties of a Σ-protocol are preserved under parallel composition, we may assume without loss of generality that the length of a challenge e in the protocol is at least k.

The protocol then proceeds as follows:

1. On input x, w, the prover computes a using the prover's algorithm from $\mathcal{P}$, chooses r at random and sends $c = commit(a, r, pk)$ to the verifier.
2. The verifier chooses e at random and sends it to the prover.
3. The prover computes z, the answer to challenge e in $\mathcal{P}$ and sends z, a, r to the verifier.
4. The verifier accepts iff it would have accepted on x, a, e, z in $\mathcal{P}$, and if $c = commit(a, r, pk)$.

It is straightforward to show that this protocol has the desired properties. First, a simulator for the protocol given an arbitrary verifier V^*:

1. Generate pk with known trapdoor t and give x, pk to V^*.
2. Send a commitment c computed according to the trapdoor property to V^* and get e back.
3. Run the honest verifier simulator on input e to get an accepting conversation (a, e, z) in the original protocol. Use the trapdoor to compute r such that $c = commit(a, r, pk)$. Send z, a, r to V^*.

This simulation works based on the hiding and trapdoor properties of the commitment scheme, and does not require rewinding of V^*, hence the protocol is also concurrent zero-knowledge.

To show it is a proof of knowledge with knowledge error $\kappa()$, we will show that the protocol satisfies the definition when we choose $\kappa(x) = 1/q(|x|)$ for *any* polynomial $q()$, thus the "true" knowledge error is smaller than any polynomial and so is negligible. This analysis is rather loose because we are dealing with a general type of intractability assumption. A much better analysis can be obtained from making a concrete assumption on a particular commitment scheme.

Our algorithm for extracting a witness will based on the following

Claim. From any prover convincing the verifier with probability $\epsilon(x) > 1/q(k)$, we can extract, using rewinding, convincing answers to two different challenges (on the same intial message) e, e', in time proportional to $1/\epsilon(x)$ for all large enough k (Recall that we have set $k = |x|$).

Intuitively, this is just because $1/q(k) > 2^{-k}$ for all large enough k, and a success probability larger than 2^{-k} must mean that you can answer more than one challenge, since the number of challenges is at least 2^k. However, the proof is a bit less obvious than it may seem: the prover may be probabilistic, but we still have to fix his random tape once we start rewinding. And there is no gurantee that the prover has success probability $\epsilon(x)$ for all choices of random tapes, indeed $\epsilon(x)$ is the average over all such choices. However, a strategy for probing the prover can be devised that circumvents this problem:

Using a line of reasoning devised by Shamir, let H a matrix with one row for each possible set of random choices by the prover, and 2^k columns index by the possible challenges (assume for simplicity that there are precisely 2^k). In the matrix we write 1 if the verifier accepts with this random choice and challenge, and 0 otherwise. Say a row is *heavy* if it contains more that $\epsilon(x)/2$ 1's. Since the total fraction of 1's in H is $\epsilon(x)$, at least 1/2 of the 1's are located in heavy rows. By using the prover as a black-box, we can probe random entries in H, or random entries in a given row, and the goal is of course to find two 1's in the same row. Consider the following algorithm:

1. Probe random entries in H until a 1 is found.
2. Start running the following two processes in parallel. Stop when at least one of them stops.
   A. Probe random entries in the row where we already found a 1, stop when a new 1 is found.
   B. Repeatedly flip a random coin that comes out heads with probability $\epsilon(x)/c$ (where c is an appropriately chosen constant, see below), and stop as soon as heads comes out. This can be done, e.g., by probing a random entry in H, choosing a random number among $1, ..., c$, and outputting heads iff both the entry and the number was 1.
3. If process A finished first, output the position of the two 1-entries found.

This algorithm obviously runs in expected time a polynomial in k times $O(1/\epsilon(x))$.

We then look at the success probability: assume the row we find is heavy. Then the expected number of trials to find a new 1 is $T(\epsilon(x)) = 2^k/(\epsilon(x)2^{k-1}-1)$ which is $O(1/\epsilon(x))$ if $\epsilon(x) > 2^{-k+2}$; and this last condition certainly holds for all large enough k. The probability that A uses less than $2T(\epsilon(x))$ trials is at least 1/2. By choosing c large enough, we can ensure that the probability that B uses more trials than $2T(\epsilon(x))$ is constant. Since the probability that we find a heavy row to begin with is constant (1/2), the total success probability is also constant. Hence repeating this entire procedure until we have success takes expected time a polynomial in k times $O(1/\epsilon(x))$, as required. This finishes the proof of the above claim.

Once we are successful, we get commitment c, conversations (a, e, z), (a', e', z') that are accepting in the original protocol, and finally values r, r' such that $c = \mathit{commit}(a, r, pk) = \mathit{commit}(a', r', pk)$. If $a = a'$, a witness for the common input x can be computed by assumption on the original protocol. Our extractor simply repeats the whole extraction process until $a = a'$.

Since one repeat of the extraction process takes expected polynomial time, it follows from the binding condition of the commitment scheme that the case $a \neq a'$ occurs with negligible probability, at most $\delta(k)$. Hence the entire extractor takes expected time $1/(1 - \delta(k))$ times the time need for one attempt. This is certainly at most $p(|x|)/(\epsilon(x) - q(|x|))$ for some polynomial $p()$.

This and the result from [20] above on existence of trapdoor commitments now implies Theorem 1. As for Theorem 2, we just need to observe that the standard zero-knowledge interactive protocols for NP complete problems [24, 1] can in fact be based on any commitment scheme. They are usually described as sequential iterations of a basic 3-move protocol. However, in our model we will use a trapdoor commitment scheme, and do the iterations in parallel: it is then trivial that the protocols can be straight line simulated if the simulator knows the trapdoor. And soundness for a poly-time bounded prover follows by a standard rewinding argument. A more careful analysis of the error probability and the way it depends on the intractability assumption we make can be obtained using the definitions from [11].

This same idea applies easily to the preprocessing model (with no auxiliary string) of Dwork and Sahai [13]: here, the prover and verifier are allowed to do a preprocessing, where timing constraints are used in order to ensure concurrent zero-knowledge. After this, the goal is to be able to do any number of interactive arguments in concurrent zero-knowledge, without timing constraints. In [13], it is shown how to achieve this based on existence of one-way trapdoor permutations. However, an idea similar to the above will allow us to do it based on any one-way function (and a smaller number of rounds): In the preprocessing, the verifier chooses an instance of the trapdoor commitment scheme from [20] and sends the public key to the prover. The verifier then proves knowledge of the trapdoor. After this, any number of interactive arguments for NP problems can be carried out in constant round and concurrent zero-knowledge. We will use the parallel version of [24] or [1] based on the commitment scheme we established in the preprocessing. Simulation can be done by extracting the trapdoor from the verifier's proof of knowledge (here, rewinding is allowed because of the timing constraints) and then simulate the main proofs straight-line.

4 Implementation in Practice

In our arguments for practicality of our model, we claimed that the public key of a third party can be used as auxiliary string. Given the construction above, this amounts to claiming that the public key of any public-key crypto-system or signature scheme can also be used without modification as the public key of a trapdoor commitment scheme.

We can assume that the public key was generated using some known key generation algorithm (recall that we originally assumed about the third party that he generates his keys properly and does not give away the private key). Clearly, the function mapping the random bits consumed by this algorithm to the resulting public key must be one-way. Otherwise, the system could be broken by reconstructing the random input and running the algorithm to obtain the private key. Thus, from a theoretical point of view, we can always think of the public key as the image of a random input under a one-way function and apply the commitment scheme from [20].

This will not be a practical solution. But fortunately, standard public key systems used in real life allow much more efficient implementations. Any system based on discrete logarithms in a prime order group, such as DSS, many El

Gamal variants, and Cramer-Shoup has as part of the public key some group element of form g^x where x is private and g is public, and where g has prime order q . This is precisely the public key needed for the trapdoor commitment scheme of Pedersen [25], which allows commitment to a string of length $\log q$ in one commitment.

If we have an RSA public key with modulus n, we can always construct from this a public key for the RSA based trapdoor commitment scheme described in [6]. We define q to be the least prime such that $q > n$ (this can easily be computed by both prover and verifier). We then fix some number b in Z_n^*, this could be for instance be a string representing the name of the verifier. The intractability assumption for the commitment scheme then is that the prover will not be able to extract a q'th root mod n of b (such a root always exists by choice of q). Also this scheme allows comitment to $\log q$ bits in one commitment.

Note that when executing a proof of the kind we constructed, it is always enough in practice for the prover to make only one commitment: he can always hash the string the wants to commit to using a standard collision intractable hash function and commit to the hash value. This means that well known efficient protocols can be executed in this model with no significant loss of efficiency.

Finally, we discuss the issue of whether a verifier can prove to a third party that he interacted with the prover. We give an example where this is possible in our model:

Suppose a public key pk is used as auxiliary string as we have described, to do proofs of knowledge for a hard relation. And suppose the verifier V interacts with a prover and then shows a transcript of the interaction to a third party C as evidence that the protocol actually took place.

Note that V, if he wanted to create the transcript completely on his own, would have to simulate the protocol *given the fixed key pk*. Now, if V computes his challenge string for instance by applying a one-way function to the first message sent by the prover in the protocol, this simulation appears to be a hard problem, unless one knows either the private key corresponding to pk or the prover's secret. Of course, this is different from simulating the verifier's *entire view*, which includes the random choice of pk - this can indeed be done efficiently since the protocol is zero-knowledge in our model.

So in this scenario, C would have to conclude that V could only have obtained the transcript by either interacting with the prover or cooperating with the party who generated pk in the first place. And if for instance this party is a CA that C trusts then he can exclude the latter possibility.

The implications of this property depend entirely on the scenario we are in. In some cases it can be an advantage to be able to prove that a protocol really took place, in other cases such tracing would harm the privacy of users.

However, in a case where a public-key infrastructure is available, in particular when V has a public key pk_V known to P, one can change the protocol slightly making it harder for V to convince C. The idea is to redefine the way in which P commits to bits, such that a commitment to bit b will have the form $commit(pk, b_1), commit(pk_V, b_2)$, where P chooses b_1, b_2 randomly such that $b = b_1 \oplus b_2$. This preserves binding and hence soundness because P does not know an honest V's private key. Also hiding and hence zero-knowledge is preserved because we can still assume that pk is correctly generated and so no information on b_1 is leaked. However, assuming that V actually knows his own private key, he can clearly use it as trapdoor for the commitment in order to simulate the protocol without interacting with P, and so seeing a transcript will not convince C in this case. This idea is closely related to the concept of verifier designated proofs (see e.g. [7, 17]).

References

1. Brassard, Chaum and Crépeau: *Minimum disclosure proofs of knowledge*, JCSS vol. 37, pp.156-189, 1988.
2. M. Bellare, R. Canetti, and H. Krawczyk: *A modular approach to the design and analysis of authentication and key exchange protocols.* STOC 98.
3. Blum, De Santis, Micali and Persiano: *Non-Interactive Zero-Knowledge*, SIAM J.Computing, vol.20, 1991.
4. Bellare and Goldreich: *Defining proofs of knowledge*, Proc. of Crypto 92, Springer Verlag LNCS series nr. 740.
5. Brandt, Landrock, Damgaård and Pedersen: *Zero-knowledge authentication scheme with secret key exchange*, J.Cryptology, vol.11, pp.147-160, 1998.
6. Cramer and Damgård: *Zero-Knowledge proofs for finite field arithmetic*, Proc. of Crypto 98, Springer Verlag LNCS series nr.1462.
7. Cramer and Damgård: *Fast and Secure Immunization against Man-in-the-Middle Impersonation*, Proc. of EuroCrypt 97.
8. Cramer, Damgaård and Schoenmakers: *Proofs of partial knowlegde*, Proc. of Crypto 94, Springer Verlag LNCS series nr. 839.
9. Canetti, Goldreich, Goldwasser and Micali: *Resettable Zero-Knowledge*, proceedings of STOC 2000.
10. Di Crescenzo and Ostrovsky: *Concurrent Zero-Knowledge: Avoiding Impossibility with Pre-Processing*, Proc. of Crypto 99, to appear.
11. Damgård and Pfitzmann: *Squential Iteration of Interactive Arguments*, proc. of ICALP 98, Springer Verlag LNCS series.
12. Dwork, Naor and Sahai: *Concurrent Zero-Knowledge*, Proc. of STOC 98.
13. Dwork and Sahai: *Concurrent Zero-Knowledge. Reducing the need for timing constraints*, Proc. of Crypto 98, Springer Verlag LNCS series, nr.1462.
14. Feige, Lapidot and Shamir: *Multiple non-interactive zero knowledge proofs based on a single random string*, Proc. of FOCS 90.
15. Feige and Shamir: *Witness indistinguishability and witness hiding protocols*, Proc. of STOC 90.
16. L. Guillou and J.-J. Quisquater: *A Practical Zero-Knowledge Protocol fitted to Security Microprocessor Minimizing both Transmission and Memory*, Proc. of EuroCrypt 88, Springer Verlag LNCS series.
17. Jacobsson, Impagliazzo and Sako: *Designated Verifier Proofs and their Applications*, Proc. of EuroCrypt 96.
18. Joe Kilian: Private communication.
19. Kilian, Petrank and Rackoff: *Lower Bounds for Zero-Knowledge on the Internet*, Proc. of FOCS 98.

20. Feige and Shamir: *Zero-knowledge proofs of knowledge in two rounds*, Proc. of Crypto 89, Springer Verlag LNCS series nr. 435.
21. C.P. Schnorr: *Efficient Signature Generation by Smart Cards*, Journal of Cryptology 4 (1991) 161–174.
22. Goldrecish and Krawczyk: *On the composition of zero-knowledge proof systems*, SIAM J.Computing, vol.25, pp.169-192, 1996.
23. Goldwasser, Micali and Rackoff: *The knowledge complexity of interactive proof systems*, SIAM J.Computing, vol. 18, pp.186-208, 1989.
24. Goldreich, Micali and Wigderson: *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge interactive proofs systems*, JACM, vol 38, pp.691-729, 1991.
25. Pedersen: *Non-interactive and information theoretic secure verifiable secret sharing*, proc. of Crypto 91, Springer Verlag LNCS series, nr. 576.
26. Richardson and Kilian: *On the Concurrent Composition of Zero-Knowledge Proofs*, to appear in Proceedings of EuroCrypt 99.