# A Note on Zero-Knowledge Proofs of Knowledge and the ZKPOK Ideal Functionality[*]

Carmit Hazay[†] Yehuda Lindell[‡]

October 28, 2010

**Abstract**

In this note, we provide a formal proof of the fact that any protocol that is a zero-knowledge proof of knowledge for a relation $R$ is also a secure protocol for the zero-knowledge proof of knowledge functionality, where the latter is defined according to the standard framework of stand-alone secure computation. Although this is a well-known fact, to the best of our knowledge, no full proof of this has been published.

## 1 Introduction

In order to keep this note brief, we assume familiarity with the definitions of zero-knowledge, zero-knowledge proofs of knowledge, and secure computation in the presence of malicious adversaries. See [3, Chapter 4] and [4, Chapter 7] for definitions of zero-knowledge and secure computation, respectively.

**The goal.** In this note, we show that any protocol that is a zero-knowledge proof of knowledge fulfills the ideal zero-knowledge proof of knowledge functionality under the definition of secure computation with malicious adversaries. This is very useful when using zero-knowledge proofs of knowledge as subprotocols since it enables the use of the modular composition theorem of [2], and so significantly simplifies proofs of security.

**The problem.** The problem that arises when attempting to prove the above is in the case that the prover is corrupted. This is due to the fact that simulation of this case (in the setting of secure computation) works by first verifying the proof from the prover. Then, if the proof is convincing, the knowledge extractor is run, and it runs in time that is close to the inverse of the probability that the proof was convincing. The problem that arises is that "close" here means a negligible difference, and this does *not* guarantee that the expected running time of the simulator is polynomial. This is explained and demonstrated in detail in the proof below. This problem was first dealt with by [5] in the context of constant-round zero-knowledge proofs. We use their exact technique, and remark that this problem actually arises quite often in secure computation. Thus, it is important to be familiar with this technique, beyond its specific application to zero-knowledge and zero-knowledge proofs of knowledge.

[*]This note is an excerpt from [7].

[†]Dept. of Computer Science, Aarhus University, Denmark. email: `carmit@cs.au.dk`.

[‡]Dept. of Computer Science, Bar-Ilan University, Israel. email: `lindell@cs.biu.ac.il`.

**Definitions.** In order to be precise, we present the definition of proofs of knowledge that we use. This is the *original definition* as it appeared in [1] and not the version of the definition as it appears in [3].

**Definition 1** *Let $\kappa : \{0,1\}^* \to [0,1]$ be a function. A protocol $(P, V)$ is a* proof of knowledge *for the relation $R$ with* knowledge error *$\kappa$, if the following properties are satisfied:*

**Completeness:** *If $P$ and $V$ follow the protocol on input $x$ and private input $w$ to $P$ where $(x, w) \in R$, then $V$ always accepts.*

**Knowledge soundness (or validity):** *There exists a constant $c > 0$ and a probabilistic oracle machine $K$, called the* knowledge extractor*, such that for every interactive prover function $P^*$ and every $x \in L_R$, the machine $K$ satisfies the following condition. Let $\epsilon(x)$ be the probability that $V$ accepts on input $x$ after interacting with $P^*$. If $\epsilon(x) > \kappa(x)$, then upon input $x$ and oracle access to $P^*$, the machine $K$ outputs a string $w$ such that $(x, w) \in R$ within an expected number of steps bounded by*

$$\frac{|x|^c}{\epsilon(x) - \kappa(x)}$$

In addition, we use the zero-knowledge proof of knowledge functionality, defined as follows. Let $R$ be a relation and define the zero-knowledge functionality $\mathcal{F}_{zk}^R$ by

$$\mathcal{F}_{zk}^R((x, w), w) = (\lambda, R(x, w)).$$

where $\lambda$ is the empty output, signifying that the first party receives no output.

## 2 The Proof of Equivalence

Intuitively, any zero-knowledge proof of knowledge for $R$ securely realizes $\mathcal{F}_{zk}^R$ because simulation takes care of the case that $V$ is corrupted and witness extraction takes care of the case that $P$ is corrupted. However, for technical reasons that will become apparent in the proof below, this is not so simple.

Before proving the theorem we remark on one technical change that must be made to the zero-knowledge proof of knowledge protocol. Specifically, in the setting of secure computation, the prover may be invoked with input $(x, w)$ such that $(x, w) \notin R$. In this case, the prover should just send 0 to the verifier and do nothing else. Thus, we add an instruction to the protocol to have the prover first verify that $(x, w) \in R$. If yes, it proceeds with the protocol, and if not it sends 0 to the verifier and halts. Observe that in order for this to be possible, the relation $R$ must be in $\mathcal{NP}$. (We also need the following property: for every $x$ it is possible to efficiently find a value $w$ such that $(x, w) \notin R$. This property holds for all "interesting" relations that we know of. Note that if it does not hold, then a random witness is almost always a valid one, and so running such a proof is meaningless.)

**Theorem 2** *Let $\pi$ be a zero-knowledge proof of knowledge with negligible knowledge error for an $\mathcal{NP}$-relation $R$. Then, $\pi$ securely computes the zero-knowledge functionality $\mathcal{F}_{zk}^R$ in the presence of malicious adversaries.*

**Proof:** Let $\mathcal{A}$ be an adversary. We separately consider the case that $\mathcal{A}$ corrupts the prover $P$ and the case that $\mathcal{A}$ corrupts the verifier $V$. In the case that the verifier is corrupted, the simulator $\mathcal{S}$ for $\mathcal{F}_{zk}^R$ receives a bit $b \in \{0,1\}$ from the trusted party. If $b = 0$, then $\mathcal{S}$ hands 0 to $\mathcal{A}$ as if coming from $P$ and halts. Otherwise, if $b = 1$, then $\mathcal{S}$ runs the zero-knowledge simulator that is guaranteed to exist for $\pi$ with $\mathcal{A}$ as the verifier. In the case that $b = 0$, the adversary $\mathcal{A}$ sees exactly what an honest prover would send. In the case that $b = 1$, by the security properties of the zero-knowledge simulator, the output generated by $\mathcal{S}$ is computationally indistinguishable from the output of $\mathcal{A}$ in a real execution with $P$. This therefore completes this corruption case.

We now consider the case that $P$ is corrupted by $\mathcal{A}$. Intuitively, the simulator $\mathcal{S}$ works as follows:

1. $\mathcal{S}$ plays the honest verifier with $\mathcal{A}$ as the prover.
   (a) If $\mathcal{A}$ sends 0 to the verifier in this execution, then $\mathcal{S}$ sends the trusted party computing $\mathcal{F}_{zk}^R$ an invalid witness $w$ such that $(x,w) \notin R$. Then, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.
   (b) If $\mathcal{S}$, playing the honest verifier, is not convinced by the proof with $\mathcal{A}$, then it sends $\mathsf{abort}_P$ to the trusted party computing $\mathcal{F}_{zk}^R$, outputs whatever $\mathcal{A}$ outputs and halts.
   (c) If $\mathcal{S}$ is convinced by the proof with $\mathcal{A}$, then it records the output that $\mathcal{A}$ outputs and proceeds to the next step.
2. $\mathcal{S}$ runs the knowledge extractor $K$ that is guaranteed to exist for $\pi$ on the prover $\mathcal{A}$, and receives back a witness $w$ such that $(x,w) \in R$. $\mathcal{S}$ then sends $w$ to the trusted party computing $\mathcal{F}_{zk}^R$ and outputs the output of $\mathcal{A}$ recorded above.

The intuition behind this simulation is clear. In the case that $\mathcal{A}$ would not convince the verifier in a real execution, the same behavior (output of the value 0 or $\mathsf{abort}_P$) is achieved in the simulation. However, in the case that $\mathcal{A}$ would convince the verifier, the simulator $\mathcal{S}$ has to send a valid witness $w$ to the trusted party computing $\mathcal{F}_{zk}^R$. It therefore runs the knowledge extractor $K$ to do this. However, $K$ runs in expected time

$$\frac{|x|^c}{\epsilon(x) - \kappa(x)}$$

where $\epsilon(x)$ is the probability that $\mathcal{A}$ would convince the verifier and $\kappa(x)$ is the (negligible) knowledge error. Now, since $\mathcal{S}$ only runs the extractor in the case that $\mathcal{A}$ convinces it in the proof while $\mathcal{S}$ plays the honest verifier, we have that $\mathcal{S}$ only runs the extractor with probability $\epsilon(x)$. Thus, the expected running-time of $\mathcal{S}$ is

$$\epsilon(x) \cdot \frac{|x|^c}{\epsilon(x) - \kappa(x)} = |x|^c \cdot \frac{\epsilon(x)}{\epsilon(x) - \kappa(x)}$$

It may be tempting at this point to conclude that the above is polynomial because $\kappa(x)$ is negligible, and so $\epsilon(x) - \kappa(x)$ is almost the same as $\epsilon(x)$. This is true for "large" values of $\epsilon(x)$. For example, if $\epsilon(x) > 2\kappa(x)$ then $\epsilon(x) - \kappa(x) > \epsilon(x)/2$. This then implies that $\epsilon(x)/(\epsilon(x) - \kappa(x)) < 2$. Unfortunately, however, this is *not* true in general. For example, consider the case that $\kappa(x) = 2^{-|x|}$ and $\epsilon(x) = \kappa(x) + 2^{-|x|/2} = 2^{-|x|} + 2^{-|x|/2}$. Then,

$$\frac{\epsilon(x)}{\epsilon(x) - \kappa(x)} = \frac{2^{-|x|} + 2^{-|x|/2}}{2^{-|x|/2}} = 2^{|x|/2} + 1,$$

which is exponential in $|x|$. In addition to the above problem, the guarantee regarding the running-time of $K$ and its success only holds if $\epsilon(x) > \kappa(x)$. Thus, if $K$ runs for time $\epsilon(x)^{-2}$ whenever $\epsilon(x) \leq \kappa(x)$, we once again have a similar problem. For example, consider the case that $\kappa(x) = \epsilon(x) = 2^{-|x|}$. Then, the expected running time of $\mathcal{S}$ for such an $\mathcal{A}$ is

$$(1 - \epsilon(x)) \cdot \mathrm{poly}(|x|) + \epsilon(x) \cdot \frac{1}{\epsilon(x)^2} > \frac{1}{\epsilon(x)} = 2^{|x|}.$$

This technical problem was observed and solved by [5] in the context of zero-knowledge. We now show how to use their technique here.

Both of the problems described above are solved by ensuring that the extractor never runs "too long". Specifically, if $\mathcal{S}$ is convinced of the proof by $\mathcal{A}$, and so proceeds to the second step of the simulation, then it first estimates the value of $\epsilon(x)$, where $\epsilon(x)$ denotes the probability that $\mathcal{A}$ successfully proves that it knows a witness $w$ such that $(x, w) \in R$. This is done by repeating the verification until $m(x)$ successful verifications occur, for a large enough polynomial $m(\cdot)$. Then, an estimate $\tilde{\epsilon}$ of $\epsilon$ is taken to be $m/T$, where $T$ is the overall number of attempts until $m$ successful verifications occurred. As we will see, this suffices to ensure that the probability that $\tilde{\epsilon}$ is not within a constant factor of $\epsilon(x)$ is at most $2^{-|x|}$. We show this using the following bound:

**Lemma 2.1** (Tail inequality for geometric variables [6]): *Let $X_1, \ldots, X_m$ be $m$ independent random variables with geometric distribution with probability $\epsilon$ (i.e., for every $i$, $Pr[X_i = j] = (1-\epsilon)^{j-1} \cdot \epsilon$). Let $X = \sum_{i=1}^{m} X_i$ and let $\mu = E[X] = m/\epsilon$. Then, for every $\delta$,*

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{m\delta^2}{2(1+\delta)}}$$

**Proof:** In order to prove this lemma, we define a new random variable $Y_\alpha$ for any $\alpha \in \mathbb{N}$ as follows. Consider an infinite series of independent Bernoulli trials with probability $\epsilon$ (i.e., the probability of any given trial being 1 is $\epsilon$). Then, write the results of these trials as a binary string and let $Y_\alpha$ be the number of *ones* appearing in the prefix of length $\alpha$. It is clear that

$$\mu_\alpha = E[Y_\alpha] = \alpha \cdot \epsilon \ .$$

Furthermore,

$$\Pr[X \geq (1+\delta)\mu] = \Pr[Y_\alpha < m]$$

for $\alpha = (1+\delta)\mu$. In order to see why this holds, observe that one can describe the random variable $X = \sum_{i=1}^{m} X_i$ by writing an infinite series of Bernoulli trials with probability $\epsilon$ (as above), and then taking $X$ to be the index of the $m$th one to appear in the string. Looking at it in this way, we have that $X$ is greater than or equal to $(1+\delta)\mu$ if and only if $Y_{(1+\delta)\mu} < m$ (because if $Y_{(1+\delta)\mu} < m$ then this means that $m$ successful trials have not yet been obtained). Observe now that $\mu_\alpha = \alpha \cdot \epsilon$, $\alpha = (1+\delta)\mu$, and $\mu = m/\epsilon$. Thus, $\mu_\alpha = (1+\delta) \cdot m$. This implies that

$$\left(1 - \frac{\delta}{1+\delta}\right) \cdot \mu_\alpha = \left(1 - \frac{\delta}{1+\delta}\right) \cdot (1+\delta) \cdot m = (1+\delta) \cdot m - \delta \cdot m = m \ ,$$

and so

$$\Pr[Y_\alpha < m] = \Pr\left[Y_\alpha < \left(1 - \frac{\delta}{1+\delta}\right) \cdot \mu_\alpha\right].$$

Applying the Chernoff bound[1], we have that

$$\Pr\Big[Y_\alpha < m\Big] = \Pr\left[Y_\alpha < \left(1 - \frac{\delta}{1+\delta}\right)\mu_\alpha\right] < e^{-\frac{\mu_\alpha}{2}\cdot\left(\frac{\delta}{1+\delta}\right)^2}$$

Once again using the fact that $\mu_\alpha = (1+\delta)\cdot m$ we conclude that

$$\Pr[X \geq (1+\delta)\mu] = \Pr\Big[Y_\alpha < m\Big] < e^{-\frac{(1+\delta)m}{2}\cdot\left(\frac{\delta}{1+\delta}\right)^2} = e^{-\frac{m\delta^2}{2(1+\delta)}}$$

as required. ■

Define $X_i$ to be the random variable that equals the number of attempts needed to obtain the $i$th successful verification (not including the attempts up until the $i-1$th verification), and let $\delta = \pm 1/2$. Clearly, each $X_i$ has a geometric distribution with probability $\epsilon$. It therefore follows that

$$\Pr\left[X \leq \frac{m}{2\epsilon} \vee X \geq \frac{3m}{2\epsilon}\right] \leq 2\cdot\Pr\left[X \geq \frac{3}{2}\cdot\frac{m}{\epsilon}\right] \leq 2\cdot e^{-\frac{m}{12}}$$

Stated in words, the probability that the *estimate* $\tilde{\epsilon} = m/X$ is not between $2\epsilon/3$ and $2\epsilon$ is at most $2e^{-m/12}$. Thus, if $m(x) = 12|x|$ it follows that the probability that $\tilde{\epsilon}$ is not within the above bounds is at most $2^{-|x|}$, as required.

Next, $\mathcal{S}$ repeats the following up to $|x|$ times: $\mathcal{S}$ runs the extractor $K$ and answers all of $K$'s oracle queries with the $\mathcal{A}$ as the prover. However, $\mathcal{S}$ limits the number of steps taken by $K$ to $|x|^{c+1}/\tilde{\epsilon}$ steps, where $c$ is the constant from the *knowledge soundness* (or *validity*) condition in the definition of proofs of knowledge (every extractor $K$ has a single constant $c$ associated with it and so $\mathcal{S}$ can use the appropriate $c$). Note that a "call" to $\mathcal{A}$ as the prover is counted by $\mathcal{S}$ as a single step. Now, if within this time $K$ outputs a witness $w$, then $\mathcal{S}$ sends $w$ to the trusted party computing $\mathcal{F}_{zk}^R$ and outputs the output of $\mathcal{A}$ that it first recorded. (We note that $\mathcal{S}$ does not need to check if $w$ is a valid witness because by the definition of $K$, it only outputs valid witnesses.) If $K$ does not output a witness within this time, then $\mathcal{S}$ aborts this attempt and tries again. As mentioned above, this is repeated up to $|x|$ times; we stress that in each attempt, $K$ is given independent coins by $\mathcal{S}$. If the extractor $K$ did not output a witness in any of the $|x|$ attempts, then $\mathcal{S}$ halts and outputs $\mathsf{fail}$. We will show that this strategy ensures that the probability that $\mathcal{S}$ outputs $\mathsf{fail}$ is negligible. Therefore, the probability that the initial verification of the proof succeeded, yet $\mathcal{S}$ does not output a valid witness, is negligible.

In addition to the above, $\mathcal{S}$ keeps a count of the overall running time of $K$ and if it reaches $2^{|x|}$ steps, then it halts, outputting $\mathsf{fail}$. (This additional time-out is needed to ensure that $\mathcal{S}$ does not run too long in the case that the estimate $\tilde{\epsilon}$ is not within a constant factor of $\epsilon(x)$. Recall that this "bad event" can only happen with probability $2^{-|x|}$.)

We first claim that $\mathcal{S}$ runs in expected polynomial-time.

**Claim 2.2** *Simulator $\mathcal{S}$ runs in expected-time that is polynomial in $|x|$.*

**Proof:** Recall that $\mathcal{S}$ initially verifies the proof provided by $\mathcal{A}$. Since $\mathcal{S}$ merely plays an honest verifier, this takes a strict polynomial number of steps. Next, $\mathcal{S}$ obtains an estimate $\tilde{\epsilon}$ of $\epsilon(x)$. This

[1]We use the following version of the Chernoff bound. Let $X_1,\ldots,X_m$ be independent Bernoulli trials where $\Pr[X_i = 1] = \epsilon$ for every $i$, and let $X = \sum_{i=1}^m X_i$ and $\mu = E[X] = m\epsilon$. Then, for every $\delta$ it holds that $\Pr[X < (1-\beta)\mu] < e^{-\frac{\mu}{2}\cdot\beta^2}$.

involves repeating the verification until $m(|x|)$ successes are obtained. Therefore, the expected number of repetitions in order to obtain $\tilde{\epsilon}$ equals exactly $m(|x|)/\epsilon(x)$ (since the expected number of trials for a single success is $1/\epsilon(x)$). After the estimation $\tilde{\epsilon}$ has been obtained, $\mathcal{S}$ runs the extractor $K$ for a maximum of $|x|$ times, each time for at most $|x|^{c+1}/\tilde{\epsilon}$ steps.

Given the above, we are ready to compute the expected running-time of $\mathcal{S}$. In order to do this, we differentiate between two cases. In the first case, we consider what happens if $\tilde{\epsilon}$ is *not* within a constant factor of $\epsilon(x)$. The only thing we can say about $\mathcal{S}$'s running-time in this case is that it is bound by $2^{|x|}$ (since this is an overall bound on its running-time). However, since this event happens with probability at most $2^{-|x|}$, this case adds only a polynomial number of steps to the overall expected running-time. We now consider the second case, where $\tilde{\epsilon}$ *is* within a constant factor of $\epsilon(x)$. In this case, we can bound the expected running-time of $\mathcal{S}$ by

$$\text{poly}(|x|) \cdot \epsilon(x) \cdot \left( \frac{m(|x|)}{\epsilon(x)} + \frac{|x| \cdot |x|^{c+1}}{\tilde{\epsilon}} \right) = \text{poly}(|x|) \cdot \frac{\epsilon(x)}{\tilde{\epsilon}} = \text{poly}(|x|)$$

and this concludes the analysis. ■

It is clear that the output of $\mathcal{S}$ is distributed exactly like the output of $\mathcal{A}$ in a real execution. This is because $\mathcal{S}$ just plays the honest verifier with $\mathcal{A}$ as the prover, and so the view of $\mathcal{A}$ in this simulation is identical to a real execution. Thus, the only problem that arises is if $\mathcal{S}$ accepts $\mathcal{A}$'s proof, but fails to obtain a valid witness. Notice that whenever $\mathcal{A}$'s proof is accepting, $\mathcal{S}$ runs the extractor $K$ and either obtains a proper witness $w$ or it outputs $\mathsf{fail}$. That is, in the case of accepting proofs, if $\mathcal{S}$ does not output $\mathsf{fail}$, then it outputs a proper witness. Therefore, it suffices to show that the probability that $\mathcal{S}$ outputs $\mathsf{fail}$ is negligible.

**Claim 2.3** *The probability that $\mathcal{S}$ outputs* $\mathsf{fail}$ *is negligible in $|x|$.*

**Proof:** Notice that the probability that $\mathcal{S}$ outputs $\mathsf{fail}$ is less than or equal to the probability that the extractor $K$ does not succeed in outputting a witness $w$ in any of the $|x|$ extraction attempts *plus* the probability that $K$ runs for $2^{|x|}$ steps.

We first claim that the probability that $K$ runs for $2^{|x|}$ steps is negligible. We have already shown in Claim 2.2, that $\mathcal{S}$ (and thus $K$) runs in expected polynomial-time. Therefore, the probability that an execution will deviate so far from its expectation and run for $2^{|x|}$ steps is negligible. (It is enough to use Markov's inequality to establish this fact.)

We now continue by considering the probability that in all $|x|$ extraction attempts, the extractor $K$ does not output a witness within $|x|^{c+1}/\tilde{\epsilon}$ steps. Consider the following two possible cases (recall that $\epsilon(x)$ equals the probability that $\mathcal{A}$ succeeds in proving the proof, and that $\kappa$ is the negligible knowledge-error function of the proof system):

1. *Case 1:* $\epsilon(x) \leq 2\kappa(x)$: In this case, $\mathcal{A}$ succeeds in proving the proof with only negligible probability. This means that the probability that $\mathcal{S}$ even reaches the stage that it runs $K$ is negligible (and thus $\mathcal{S}$ outputs $\mathsf{fail}$ with negligible probability only).

2. *Case 2:* $\epsilon(x) > 2\kappa(x)$: Recall that by the definition of proofs of knowledge, the constant $c$ is such that the expected number of steps taken by $K$ to output a witness is at most $|x|^c/(\epsilon(x) - \kappa(x))$. Now, since in this case $\epsilon(x) > 2\kappa(x)$, it holds that the expected number of steps required by $K$ is less than $2|x|^c/\epsilon(x)$. Assuming that $\tilde{\epsilon}$ is within a constant factor of $\epsilon(x)$, we have that the expected number of steps is bound by $O(|x|^c/\tilde{\epsilon})$. Therefore, by Markov's

inequality, the probability that $K$ runs longer than $|x|^{c+1}/\tilde{\epsilon}$ steps in any given extraction attempt is at most $O(1/|x|)$. It follows that the probability that $K$ runs longer than this time in $|x|$ independent attempts is negligible in $|x|$ (specifically, it is bound by $O(1/|x|)^{|x|}$). This covers the case that $\tilde{\epsilon}$ is within a constant factor of $\epsilon(x)$. However, the probability that $\tilde{\epsilon}$ is *not* within a constant factor of $\epsilon(x)$ is also negligible. Putting this together, we have that $\mathcal{S}$ outputs fail with negligible probability only.

■

Combining the above two claims, together with the fact that the simulation by $\mathcal{S}$ is perfect when it does not output fail, we conclude that $\mathcal{S}$ is a valid simulator for the case that $P$ is corrupted. Thus, $\pi$ securely computes the $\mathcal{F}_{zk}^{R}$ functionality. ■

## References

[1] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In *CRYPTO'92*, Springer-Verlag (LNCS 740), pages 390–420, 1992.

[2] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[3] O. Goldreich. *Foundations of Cryptography: Volume 1 – Basic Tools.* Cambridge University Press, 2001.

[4] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge University Press, 2004.

[5] O. Goldreich and A. Kahan. How To Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.

[6] S. Har-Peled. Lecture Notes on Approximation Algorithms in Geometry, Chapter 27, Excercise 27.5.3, 2010. Currently found at http://valis.cs.uiuc.edu/∼sariel/teach/notes/aprx/.

[7] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols – Techniques and Constructions.* Springer, 2010.