Lecture Notes for CS-578 Cryptography (SS2006) Prof. Michael Backes

15. Zero-Knowledge Proofs

Lecture 20 Saarland University

Zero-knowledge proof systems are intensely studied in modern theoretical cryptography. Similar to commitment schemes, they mostly occur as subprotocols in other protocols, but there also exist direct applications. Examples for direct applications are as identification schemes and constructions of signature schemes from these identification schemes.

15.1 Overview of Types of Proof Systems and Zero-Knowledge

We first give an overview of possible goals with proof systems. The notions of a proof system and zero-knowledge are actually independent: A *proof system* is a 2-party protocol between a so-called *prover* and a so-called *verifier*. The goals are roughly:

- *Completeness*: The prover can convince the verifier of correct statements.
- *Soundness*: Not even a cheating prover can convince an honest verifier of wrong statements. Note that "completeness" in the terminology of proof systems is simply the property that everything works if nobody cheats, whereas "soundness" is the main integrity goal.

Being *zero-knowledge* is a property of one interactive machine, here the prover. It means that this machine, no matter with what other machines it interacts, does not give the other machines any new knowledge. The definition of "any new knowledge" is not trivial and takes some getting used to when one first sees it. For example, one might think that being convinced after a proof seems to be "knowledge" and thus the notion of a zero-knowledge proof would be contradictory. There are two major classes of proof systems with different goals:

- Proofs of *language membership*: Here a language L is given, i.e., a subset of the strings over some alphabet, typically $\{0, 1\}$. The statements to be proved are that a certain string x, known to both the prover and the verifier, lies in L.
- Proofs of *knowledge*: Here a binary relation R is given. The statements to be proved are that for a certain string x, known to both parties, the prover "knows" a string w with $(w, x) \in R$. The string w is often called a *witness* for x.

Examples Proving that a number x is a quadratic residue is an example of a proof of language membership: The language L in this case is actually the set of pairs (N, x) with $x \in QR_N$. Proving that one knows a square root of a number x modulo N is an example of a proof of knowledge: The relation in this case is $R = \{(w, (x, N)) \mid w^2 = x \bmod N\}$.

It should be fairly clear at this point how the security of proofs of language membership will be formalized, whereas defining "knowledge" is non-trivial, just like the other extreme, "zero-knowledge". The goals can hold with information-theoretical or computational security. Proofs whose soundness only hold with computational security are often called arguments. Moreover, one can construct proof systems either for one particular language L or for a large class of languages, and similarly for relations. Hence there is an even greater variety of constructions of zero-knowledge proof systems than of commitment schemes.

15.2 Defining Interactive Proofs

In this section, we do not bother about zero-knowledge yet, i.e., about the secrecy aspect of our protocols, but only about the proof properties, i.e., about integrity aspects. These aspects are in the interest of the verifier. Thus we assume that the verifier is honest, whereas the prover may try to cheat. We first define proofs of language membership and then proofs of knowledge.

15.2.1 Proofs of Language Membership

Before we can formalize the definition, we have to consider for what reason it makes sense to have an *interactive* proof at all: Why doesn't the verifier sit down at home and prove the statement for himself? The reasons are different in complexity theory and in practical cryptography, and this leads to two different definitions.

In *complexity theory*, interactive proofs are seen as an extension of the concept of NP: One can regard a language $L \in NP$ as a language with an interactive proof system where the prover is computationally unrestricted and sends only one message. This message is a witness for the NP statement, i.e., the string that the verifier would otherwise have to guess non-deterministically, e.g., a satisfying assignment for a Boolean formula if $L = SAT$. Given such a string, the verifier can decide in polynomial time whether it is correct or not. We will not follow this further, because honest provers in practical protocols cannot be computationally unrestricted. Let us just mention that a famous result says that $IP = PSPACE$, where IP is the set of all languages with an interactive proof system, i.e., the class is presumably larger than NP.

In *practical cryptography*, the reason why the prover can do more than the verifier is that he has originally chosen some secrets, i.e., he has auxiliary information at the start of the proof protocol. We will have to mention this auxiliary information in the definition, and the proofs will only work if this information is somehow correct.[1]

Example In an interactive proof that a number x_0 is a quadratic residue modulo some N, we will typically exploit that the prover knows the prime factors of N.[2] Most definitions of interactive proofs in the literature are in the first scenario, which makes them a bit simpler than the following one. However, the following definition is still a compromise: A definition that can really be used for most things one usually wants to do with it contains even more parameters.

Definition 15.1 (Interactive Proof System for Language Membership) *An* interactive proof system for language membership *with generation algorithm has the following components:*

[1] In these cases, if we would not bother about zero-knowledge, an interactive proof could always be transformed into one where only one message is sent: The prover would send his entire auxiliary information to the verifier, and the verifier could carry out the rest of the protocol alone (i.e., run both interactive algorithms).

[2] The auxiliary information here is similar to the witness in a proof of knowledge. However, here the goal of the proof is not to prove that the prover has some particular auxiliary information, in contrast to proofs of knowledge. For example, even if we construct a correct prover algorithm that needs the factors of N as auxiliary information, it is not forbidden that a cheating prover also succeeds if he only knows a root of x_0 or some other auxiliary information.

- *The parameters are the language L and certain security parameters, which we collectively call par.*
- *There are two roles, the* prover *and the* verifier.
- *A probabilistic algorithm* Gen. *It is meant for the prover to generate an element from the language together with suitable auxiliary information. The input is only the security parameters par, and the output is a pair (x, aux).*
- *The subprotocol* prove *is a 2-party protocol. The individual interactive algorithms of the prover and the verifier in it are called* P *and* V.
  - P *needs an initial input (par, x, aux) and does not make a final output.*
  - V *needs an initial input (par, x) and produces a Boolean output b, where 0 denotes that it is convinced that $x \in L$.*

  Let

$$Exp_{\mathsf{P},\mathsf{V}}((par, x, aux), (par, x)) = b$$

denote the event that V *finally outputs the bit b.*

These algorithms fulfill the following requirements:

- Correct generation: *For all valid parameters par, and all $(x, aux) \in [\mathsf{Gen}(par)]$, we have $x \in L$.*
- Completeness: *If x has been generated correctly, a correct prover can always convince a correct verifier. More precisely, for all valid parameters par:*

$$\mathsf{Pr}\left[Exp_{\mathsf{P},\mathsf{V}}((par, x, aux), (par, x)) = 0;\ (x, aux) \leftarrow \mathsf{Gen}(par)\right] \ = \ 1.$$

- (Information-theoretical) Soundness*: If $x \notin L$, no cheating prover can convince a correct verifier with not with more than an exponentially small probability. For this, we will now assume that par, the tuple of parameters, contains a parameter σ that determines how small this probability should be. Then we can write: For all efficient interactive algorithms P^*, all valid parameters par, all $x \notin L$, and all aux, the following holds:*

$$\mathsf{Pr}\left[Exp_{\mathsf{P}^*,\mathsf{V}}((par, x, aux), (par, x)) = 0\right] \ \leq \ 2^{-\sigma}.$$

$\diamond$

To define *computational soundness*, one has to be more careful where x and aux come from than in the information-theoretic version. The best way to define this is to let them be chosen by the cheating prover, i.e., in polynomial time, but not with the correct generation algorithm. This gives the following definition:

Definition 15.2 (Computational Soundness) *Assume that we have a proof system according to Definition 15.1 except for the soundness, and we have only one parameter $par = n$. Then we say that this system has* computational soundness *if for all efficient algorithms Gen^* and P^*, where P^* is interactive, the following holds:*

$$\mathsf{Pr}\left[x \notin L \ \wedge \ Exp_{\mathsf{P}^*,\mathsf{V}}((n, x, aux), (n, x)) = 0;\ (x, aux) \leftarrow \mathsf{Gen}^*(n)\right] \ \textit{is negligible in } n.$$

$\diamond$

15.2.2 Proofs of Knowledge

Before looking at the current definition of proofs of knowledge, it may be useful if you try for yourself for a while how you would define such a notion, so that you see the problems. Some approaches are discussed in the sequel, followed by the actual definition.

One approach at defining knowledge might be to say: If a machine knows a certain value w, this w should be written down somewhere in the machine's memory. Then soundness of a proof of knowledge would be something like "no machine P^* can convince V without having w written down correctly in a certain place on its tapes". This is fine for correct machines P, but difficult for cheating machines. For example, a cheating machine P^* might have w written down in reverse order, or encrypted with a simple encryption system. Then P^* will certainly still be able to convince a verifier by simulating P and always looking up or decrypting the appropriate parts of w. Thus soundness in this narrow sense would never be fulfilled.

The next approach might be to say: "Any P^* that can convince V has w written down in some encoding". But now, what is an arbitrary encoding? E.g., having primes p and q written as $N = pq$ is a valid encoding in the information-theoretic sense, but not what we want: We would not want to say that someone knows the factors p and q if in fact he only knows N.

Thus we want to add a computational aspect to the encoding, somehow like "P^* must have written down something from which w can be derived in polynomial time". Polynomial-time computability is defined for individual input-output pairs, only for an entire function. Here the inputs are what an arbitrary successful P^* has written down. This is still a rather vaguely defined set: E.g., is the program of P^* needed in the input in addition to the content of its tapes?

The following definition is slightly more restricted: It says that deriving w should be possible by using P^* as a black box. I.e., it requires that a polynomial-time algorithm K exists that finds out w from the initial state of P^*, but without examining the internal details of P^*. More precisely, K can interact with P^* in the normal way, and reset P^* to a previous state, including the state of its random tape. The algorithm K is called the *knowledge extractor*.

The final question is what to do with provers P^* that are only successful with a small probability. The intuition is that a prover that does not know a witness w should only be successful with negligible probability. Therefore the extractor K should be successful for all provers P^* whose success probability is at least the inverse of any polynomial. However, it is intuitively clear that the smaller the success probability of P^* is, the more difficult extracting will get: The extractor will not have much chance unless it observes at least one case where P^* is successful. The current definition handles this by allowing K a running time inversely proportional to the success probability of P^*.

The following definition is due to Bellare and Goldreich. Except for the soundness, it is quite similar to Definition 15.1, but we now have a relation R instead of the language L.

Definition 15.3 *An* interactive proof-of-knowledge system *has the following components:*

- *The parameters are a binary relation R and certain security parameters, which we collectively call par. Let L_R denote the language of all values x that have a witness w, i.e.,*

$$L_R := \{x \mid \exists w : (w, x) \in R\}.$$

- *There are roles and subprotocols as in Definition 15.1.*[3]

[3]We could live without the generation algorithm Gen in this definition. (Compare how completeness is defined here and in Definition 15.1). However, it will be needed again in the computational zero-knowledge property.

Again we denote the event that V *outputs* b *by*

$$Exp_{\mathsf{P},\mathsf{V}}((par, x, aux), (par, x)) = b.$$

The requirements are:

- Correct generation*: For all valid parameters* par*, the output* (x, w) *of* $\mathsf{Gen}(par)$ *is always a value* x *with its witness, i.e.,* $(w, x) \in R$.
- Completeness*: A correct prover whose input* w *is a witness for the common input* x *can always convince a correct verifier. More precisely, for all valid parameters* par *we have*

  $$\mathsf{Pr}\left[Exp_{\mathsf{P},\mathsf{V}}((par, x, w), (par, x)) = 0;\ (x, w) \leftarrow \mathsf{Gen}(par)\right] = 1.$$

- Weak soundness *(see the remarks below): We now assume that* par*, the tuple of parameters, contains a computational parameter* n*. There should exist a probabilistic polynomial-time algorithm* K*, called* knowledge extractor*, that can interact with and reset one other machine* P^**, and a polynomial* pol *such that the following holds: For all probabilistic polynomial-time interactive algorithms* P^**, all strings* aux *and all* $x \in L_R$ *we define* $p_{\mathsf{P}^*}(par, x, aux)$ *to be the probability that* V *accepts on input* (par, x) *in interaction with* P^* *using* aux*, i.e.,*

  $$p_{\mathsf{P}^*}(par, x, aux) := \mathsf{Pr}\left[Exp_{\mathsf{P}^*,\mathsf{V}}((par, x, aux), (par, x)) = 0\right].$$

  Then on input (par, x)*, and interacting with this* P^**, the knowledge extractor* K *should output a witness* w *for* x *in expected time*

  $$\frac{pol(n)}{p_{\mathsf{P}^*}(par, x, aux)}.$$

$\diamond$

Remarks and Variants The soundness in the above definition is called weak because only strings $x \in L_R$ are considered. For instance, this means that we don't bother if a cheating prover can convince an honest verifier that he knows non-trivial factors p and q of a prime number N, or a square root of a number $x \notin QR_N$. Weak soundness is sufficient in several applications, in particular in identification schemes, where x and w are initially chosen by an honest participant. We define *strong soundness* to mean weak soundness plus the requirement that prove, together with a suitable generation algorithm, is a proof of language membership for L_R.[4]

15.3 Defining Zero-Knowledge

Zero-knowledge is, as mentioned, a property of an individual machine, in our case the prover. Just as with knowledge, the definition deserves some motivation. (But the two notions of knowledge and zero-knowledge have not much in common.)

Some Introduction

The first point to notice is that the definition is not about keeping a specific secret hidden, but it is supposed to capture that the prover gives the verifier "no knowledge at all". The reason for this is that one wants to use zero-knowledge proofs in a modular way:

[4]It is tempting to simply extend the definition of weak soundness to all x. However, for $x \notin L_R$, this does not work if there is any error probability because the knowledge extractor can never find a witness.

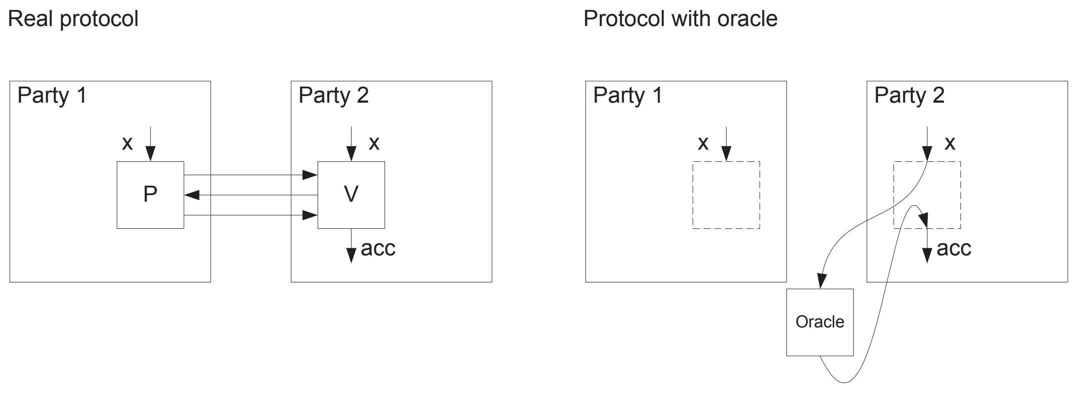


Figure 15.1: Sketch of modular proof of a protocol that uses a zero-knowledge proof as a subprotocol

First, one designs a surrounding protocol, in our examples a commitment scheme, and simply assumes that in the place where the zero-knowledge proof is needed, some trusted oracle tells the verifier $b = 0$ or $b = 1$, see Figure 15.1. It is typically much easier to prove security of this simplified surrounding protocol than of the real one, in particular for the party who plays the prover.

Second, one hopes that the real protocol, i.e., with the real zero-knowledge proof instead of the oracle, is also secure. Intuitively, it makes no significant difference whether the party who plays the verifier gets an entire view of a run of the zero-knowledge proof or only the final result.[5] So how can the verifier's view in a zero-knowledge proof be "no knowledge at all"? The basic idea is: Something is no new knowledge if one could easily have computed it alone. Thus we want to define that the verifier could easily have computed his view in the zero-knowledge proof alone, given only x and no interaction with the prover. This may sound strange: If one can compute such views without knowing the prover's secret, where is this secret used at all? Can't anyone else cheat the verifier by also computing such views? The point is in the order of computation: Typically a zero-knowledge proof is a game of questions (often called challenges) and answers. It will only be possible to answer arbitrary questions correctly if one knows a secret. However, one can somehow make up pairs of answers together with suitable questions even without the secret.

Example Imagine that the questions (asked by the verifier) were random quadratic residues x modulo a number N, and the correct answers were roots y of x. Then one should hope that only someone who knows the factors of N can answer all questions correctly. Nevertheless, the view, seen afterwards, only consists of pairs (x, y) with $y^2 = x \bmod N$, which one can easily make up oneself by choosing y first. Note, however, that this is not yet a real zero-knowledge proof because it only considers an honest verifier: A cheating verifier could choose quadratic residues where he already knows one root, and then use the second root obtained from the prover to factor N. We can already see from the example that the point of "computing the view alone" is not to compute one particular view, but to generate possible views with the correct probability distribution. Then an outside observer (or surrounding protocol) to whom the verifier later gives a view cannot distinguish

[5]Now one can also see why the fact that the verifier is convinced after the proof, but not before, is no knowledge at all: The zero-knowledge property is only defined for an honest prover. The honest prover should not try to prove wrong statements. Thus if the prover is honest, the verifier's result is always $b = 0$, i.e., it is not even one bit of new knowledge.

whether it is a real view or one computed by the verifier himself. This statement exists in three versions:

- *Perfect zero-knowledge*: This is the usual word for information-theoretical zero-knowledge without error probability. It is defined exactly as explained so far, i.e., the correct views and those that the verifier computes himself have exactly the same distribution.
- *Statistical zero-knowledge*: Here the probability distributions may have very small differences.
- *Computational zero-knowledge*: Here it is not required that the distributions are actually equal. It is only required that nobody can distinguish them in polynomial time. This roughly means that someone who gets either a correct view or a simulated view (each drawn according to their probability distributions) cannot guess much better than with probability 1/2 what he got.

Finally, we have to define what it means that "the verifier could compute something". This will be done by requiring that a machine S, the simulator, exists, which actually carries out this computation, given only the input of the verifier. We must also consider cheating verifiers V^*. Their views will usually have a different probability distribution than those of honest verifiers because already their own questions are distributed differently. Thus there will be a different simulator $\mathsf{S}_{\mathsf{V}^*}$ for each V^*, see Figure 15.2. However, when proving a zero-knowledge property in practice, one usually makes a black-box reduction. This means that one constructs one global algorithm S that performs the simulation for any verifier V^* by using V^* as a black box, including resetting V^* to a previous state (i.e., just like a knowledge extractor can use a cheating prover P^*).

Definition of Indistinguishability

To make the zero-knowledge definition modular, one typically defines indistinguishability of ensembles of probability distributions, like real views and simulated views, first. An ensemble of probability distributions is defined as a family

$$(Prob_i)_{i \in I}$$

of probability distributions. This means that a set I, called *index set*, is given, and for each $i \in I$, there is a probability distribution (over some other, unnamed set). In our case, all sets will be subsets of the bitstrings.

Example Let's see that we really need ensembles of probability distributions and not just individual distributions, even for a specific verifier V^*. In the above example the index set I would be the numbers $N = pq$, and one would try to distinguish the probability distribution of real views (x, y) for this N and simulated views (x, y) for the same N. The success probability hopefully gets very small if N is large enough. Indistinguishability is defined for two ensembles over the same index set.

Definition 15.4 *Let two ensembles $(Prob_i)_{i \in I}$ and $(Prob^*{}_i)_{i \in I}$ of probability distributions over bitstrings for one index set $I \subseteq \{0,1\}^*$ be given.*

- *They are called* perfectly indistinguishable *if they are equal.*

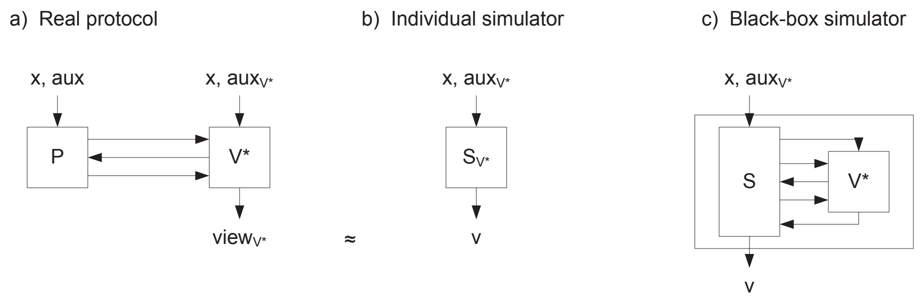


Figure 15.2: Notion of a simulator

- *They are called* computationally indistinguishable *for a certain generation algorithm* Gen^* *that generates indices* $i \in I$*, given parameters par, if the following holds for all probabilistic polynomial-time algorithms* Dist *(the distinguisher): For all* $i \in I$*, one defines probabilities*

$$\begin{aligned} Pr_i &:= \mathsf{Pr}\left[\mathsf{Dist}(i, v) = 0;\ v \leftarrow Prob_i\right], \\ Pr_i^* &:= \mathsf{Pr}\left[\mathsf{Dist}(i, v) = 0;\ v \leftarrow Prob^*{}_i\right], \end{aligned}$$

  and their difference

$$\Delta_i := |Pr_i - Pr_i^*|.$$

  Now one requires that the differences are negligible on average, i.e.,

$$\mathsf{E}(\Delta_i;\ i \leftarrow \mathsf{Gen}^*(par)) \quad \textit{is negligible.}$$

  Here, $\mathsf{E}(x;\ \ldots)$ *denotes the expected value of a random variable* x *in the probability space defined after ";", similar to the notation* $\mathsf{Pr}\left[(pred;\ \ldots)\right]$.[6]

◇

Definitions of Zero-Knowledge

The final addition to be made to the explanation is that a cheating verifier V^* in a zero-knowledge protocol may have an auxiliary input aux_{V^*} even if an honest verifier has no such thing. This corresponds to a-priori knowledge of the verifier. The most important cases of such a-priori knowledge are that the prover uses the same secret in a zero-knowledge proof and a surrounding protocol, or in several successive zero-knowledge proofs. For arbitrary a-priori knowledge, the view of the verifier should give no new knowledge.

We now present the details of the definition.

Definition 15.5 *Let an interactive proof system with generation algorithm be given, i.e., a protocol according to Definition 15.1.*

- *It is called* perfectly zero-knowledge *if the following holds: For all probabilistic polynomial-time interactive algorithms* V^* *(the cheating verifier), there exists a probabilistic polynomial-time algorithm* $\mathsf{S}_{\mathsf{V}^*}$*, called* simulator*, such that the following two ensembles are* perfectly indistinguishable*:*

[6]There is also a definition for computational indistinguishability where the distinguisher actually tries to guess which of the two probability distributions v was drawn from. However, the above definition better fits our needs.

  - *The index set is the set of possible inputs, i.e., the set I of tuples $i = (par, x, aux, aux_{\mathsf{V}^*})$ with $(x, aux) \in [\mathsf{Gen}(par)]$.*
  - *The two probability distributions for such an i are V^*'s view in the experiment*

$$Exp_{\mathsf{P},\mathsf{V}^*}((par, x, aux), (par, x, aux_{\mathsf{V}^*}))$$

  and the output of $\mathsf{S}_{\mathsf{V}^*}(par, x, aux_{\mathsf{V}^*})$.

- *For* computational indistinguishability, *we only consider auxiliary information aux_{V^*} generated from x and aux in a feasible way, i.e., by an arbitrary probabilistic polynomial-time algorithm G. This is natural because we assume that this information results from previous executions of protocols using x and aux, and both the honest party and the attacker are now computationally restricted. Given such an algorithm G, the entire generation algorithm $\mathsf{Gen}_{\mathsf{G}^*}$ that generates an index i from given parameters is defined by*

$$(x, aux) \leftarrow \mathsf{Gen}(par); aux_{\mathsf{V}^*} \leftarrow \mathsf{G}(x, aux).$$

  Now we require that the two ensembles defined as above are computationally indistinguishable for any such generation algorithm $\mathsf{Gen}_{\mathsf{G}^*}$.

$\diamond$

15.4 Proving Quadratic Residuosity

In this section we look at a concrete zero-knowledge proof system for language membership. The prover P wants to prove to the verifier V that a number x is a quadratic residue modulo another number N. The language characterizing the goal of the proof is therefore

$$L = \{(N, x) | x \in QR_N\}.$$

Typically such a proof system is used for composite N, because if N is prime, the verifier can decide quadratic residuosity by himself, but the proof system works in all cases. The auxiliary information that the prover needs is a square root of x, i.e.,

$$aux = y \text{ with } y^2 = x \bmod N.$$

This is a rather week prerequisite: If a prover knows the factors of N, he can compute such a y for any quadratic residue and therefore use the proof system. However, the proof system even works if the prover does not know these factors and has generated x by choosing y first and squaring it.

15.4.1 Construction

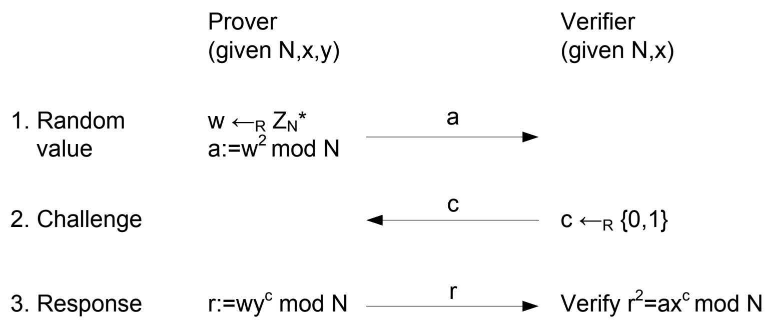
The scheme has the following parameters. The language L describing what is to be proved is

$$L = \{(N, x) | x \in QR_N\}.$$

There is a security parameter σ such that the error probability in the soundness will be at most $2^{-\sigma}$.

It has the following protocols:

- *Generation algorithm*: Any algorithm Gen can be used that in fact outputs pairs $((N,x),y)$ with $y^2 = x \bmod N$, no matter with what probability distribution.
- The *main protocol* consists of the following basic protocol, which is iterated σ times:



  In each iteration, new and independent values w and c are chosen. The verifier accepts if all σ verifications yield true.

The security of this protocol is based on the following ideas:

- *Perfect zero-knowledge*: The only thing the prover sends that depends on the secret y is the response r. This is either a random value w, which cannot possibly tell anything about y, or it is y encrypted with a multiplicative one-time pad w modulo N. In the formal version it will also become clear that a, the public version of the "one-time pad", does not give any new information in this case.
- *Soundness*: If x is not a quadratic residue, then at most one of the values a and ax can be a quadratic residue. Hence for at most one of them an acceptable response r exists. Thus in each iteration, the verifier will only accept with probability $1/2$, and overall only with probability $2^{-\sigma}$. Now we show the security in more detail.

15.4.2 Security of the Construction

Theorem 15.1 *The above construction is a* perfect zero-knowledge proof system. □

Proof. In the sequel we will prove that the above construction provides completeness, soundness, and perfect zero-knowledge.

Completeness Here both parties are honest, and the prover's input fulfills $y^2 = x \bmod N$. We have to show that the verifier always accepts. This can easily be seen if we substitute the definition of r into the verification equation:

$$r^2 = (wy^c)^2 = w^2(y^2)^c = ax^c \bmod N.$$

Soundness Now a pair (N,x) with $x \notin QR_N$ is given, and an arbitrary prover P^* with an arbitrary auxiliary input aux^*. In each iteration of the protocol, P^* must first send a value a.

- If $a \notin QR_N$, no value r with $r^2 = a$ exists, and thus no acceptable response for $c = 0$.

- If $a \in QR_N$, then $ax \notin QR_N$ (otherwise $x = (ax) \cdot x^{-1} \in QR_N$ would follow). Hence no value r with $r^2 = ax$ exists, and thus no acceptable response for $c = 1$.

Hence no matter how P^* chooses a, it can respond correctly to at most one of the two possible challenges c. Moreover, at the time when P^* chooses a, it cannot know which challenge it will get. Hence it passes each iteration with probability at most $\frac{1}{2}$, and all iterations together with probability at most $2^{-\sigma}$.

Perfect zero-knowledge Here we have to construct a simulator that can produce correct-looking views (a, c, r, v^*) of the protocol, where v^* denotes the final output of V^* (which can contain the entire internal view of V^* without loss of generality). We start with two simpler tasks to make the ideas clearer, but we will only show that the views have the correct probability distribution in Part (d) for our final simulator.

(a) Honest-verifier zero-knowledge for one iteration: Here we construct a specific simulator S_V that only works for the correct verifier V in one iteration of the protocol. The simulator's input is only (N, x). Hence it cannot compute the views in the correct order. Instead, we let it start at the end:

S_V:
1. It chooses $r \leftarrow_\mathcal{R} \mathbb{Z}_N^*$.
2. It chooses $c \leftarrow_\mathcal{R} \{0, 1\}$ just like the correct V.
3. Now it solves the verification equation for the remaining variable a: $a := r^2 \cdot x^{-c}$.

(b) Arbitrary verifier V^*, one iteration: Compared with S_V, we now have a problem in Step 2: We do not know how V * chooses c. In particular, it may choose c as a function of a. Our simulator cannot do that, because it has to compute a last. We deal with this as follows:

$\mathsf{S}_{\mathsf{V}^*}^{(1)}$
1. Choose r, c, a like S^V.
2. Start V^*, send a to it, and wait until it answers with a challenge c^*.
3. If $c^* = c$, then send r to V^* and get its final output v^*. Output $v = (a, r, c, v^*)$.
   Else try again (from Step 1).

This means that $\mathsf{S}_{\mathsf{V}^*}^{(1)}$ first simply guesses some c. Then, when it has computed a, it checks what value c^* the cheating verifier V^* would have output for this a. (Recall that the simulator can use V^* as a subprotocol and reset it.) If $c^* = c$, we have a consistent view and output it. Otherwise, this view is thrown away and another attempt is made.

(c) Overall simulator $\mathsf{S}_{\mathsf{V}^*}$: Essentially, it treats each iteration like $\mathsf{S}_{\mathsf{V}^*}^{(1)}$. It must only take care that V^* might act differently in each iteration, possibly depending on the results of the previous iterations. Hence at the end of each iteration i, it also stores the resulting state $state_i$ of V^*. In each attempt to produce a view for iteration $i+1$, it resets V^* to $state_i$. Moreover, it need not output intermediate values v_i^* for each round, but only the final v^*.

(d) Correct distribution: We have claimed perfect zero-knowledge. Thus we have to show that the outputs of $\mathsf{S}_{\mathsf{V}^*}$ have exactly the same probability distribution as the views of V^* in interaction with the real prover (for the same values N, x, y, and aux_{V^*}). We show this even for V^* with any fixed content rand of its random tape; then it obviously also holds on average over all values rand. Thus we only need to consider deterministic verifiers. Thus in each iteration of the real protocol, the only random choice is w_i, and $view_i$ and $state_i$ are a function of $state_{i-1}$

and w_i. (We have now added an index i to each variable.) Let the function be f. We first show that any simulated view v_i and its $state_i$ are also $f(state_{i-1}, w_i)$ for $w_i := r_i/y^{c_i}$. (It does not matter that the simulator does not know w_i; this is only our proof technique.)

1. a_i should be w_i^2, and in fact, $w_i^2 = (r_i/y^{c_i})^2 = r_i^2/x^{c_i} = a_i$ by the choice of a_i.
2. Now c_i is the correct challenge from V^* for this situation because the simulator explicitly verified this.
3. r_i should be $w_i y^{c_i}$. In our case, this is clear by the choice of w_i.
4. Now $state_i$ is produced by the same V^* in the simulation as in the real run, based on the same.

It remains to show that the values w_i, which determine the entire view for this iteration, occur with the correct probability in the simulation, i.e., that they are uniformly distributed. In the simulation, r_i/y^{c_i} where r_i and c_i are chosen uniformly. Thus each w_i occurs for exactly one pair $(r_i, c_i = 0)$ and one pair $(r_i, c_i = 1)$. One of these cases is that with $c_i^* = c_i$, so in one case the view is thrown away and in the other case it is be output. Hence we have both shown that each attempt at producing v_i is successful with probability 1/2 (which shows that the running time of the simulator is on average twice that of the real protocol) and that the retained views have the correct probability distribution.

The overall proof is by induction: We have just shown that when $state_{i-1}$ is distributed correctly, then so are v_i and $state_i$. This clearly yields that the overall view and the final output of V^* are distributed correctly.

∎

Remark This protocol is also a proof of knowledge that the prover knows y. The basic idea for the knowledge extractor is the following: If it can get correct responses to both $c = 0$ and $c = 1$ for the same value a, then these should be $r_0 = w$ and $r_1 = wy$, and thus he can compute y from them as r_1/r_0. More precisely, it is only verified that the responses fulfill $r_0^2 = a$ and $r_1^2 = ax$, but this also implies that $(r_1/r_0)^2 = ax/a = x$ and thus $y = r_1/r_0$ is the correct knowledge.