

## Zero Knowledge Protocols

## A Mathematical Proof

When referring to a proof in logic we usually mean:

1. A sequence of statements.
2. Based on axioms.
3. Each statement is derived via the derivation rules.
4. The proof is fixed, i.e., in any time, anyone can read it, and get convinced.

## Other Kinds of “Proofs”

However, in many situations, we “prove” a statement by convincing someone. For example, in court the prosecutor tries to convince the judge that the defendant is guilty. The prosecutor challenges the defendant. In case he fails to answer in a consistent manner, we say that the prosecutor proved his point. This kind of “proof” has an interactive nature.

## Interactive Proof System

An interactive proof for the decision problem  $\Pi$ , is a the following verification protocol:

1. There are two participants, a **prover** and a **verifier**.
2. The proof consists of a specified number of rounds.
3. In the beginning of the proof both participants get the same input.
4. In each round, the verifier challenges the prover, and the prover responds to the challenge.
5. Both the verifier and the prover can perform some private computation (they are both modeled as a randomized Turing machine).
6. At the end, the verifier states whether he was convinced or not.

## Interactive Proof System (cont.)

Let  $L$  be some language and let  $\pi(x)$  be the decision problem whether  $x \in L$ . An interactive proof system for  $\pi(x)$  must have the following properties:

1. **Completeness:** Every  $x \in L$  is accepted with a high probability (e.g., at least  $2/3$ ).
2. **Soundness:** Every  $x \notin L$  is rejected with a high probability.
3. **Polynomial verification:** The verifier must do his private computation in polynomial time.

## Example — Graph Isomorphism

**The Graph Isomorphism Problem:** Given two graphs  $G_1$  and  $G_2$ , where  $|V_1| = |V_2| = N$ . Is there a permutation  $\pi$  on  $V_1$  such that  $(u, v) \in E_1 \iff (\pi(u), \pi(v)) \in E_2$ .

We give two different interactive proofs for it.

## A Trivial Interactive Proof

1. Given  $G_1, G_2$ .
2. The prover sends a permutation  $\pi$  which maps the vertices of  $V_1$  to  $V_2$ .
3. The verifier checks whether this permutation maps  $V_1$  to  $V_2$ . If it is, the verifier accepts the instance, otherwise he rejects it.

**Completeness:** If the graphs are isomorphic, the verifier always accepts it.  
**Soundness:** If the graphs are not isomorphic, the prover can not provide an isomorphism. Therefore, the verifier always rejects it.  
**Polynomial verification:** The verifier has to generate  $\pi(G_1)$ , and check its equality to  $G_2$ . This can be done in linear time.  
**Result:** The above protocol is an interactive proof.

## Example of A Zero Knowledge Interactive Proof

1. Given  $G_1, G_2$ .
2. Do  $n$  rounds of the following:
  - (a) The prover chooses a random permutation  $\sigma$  and computes  $H = \sigma(G_2)$ . Then he sends  $H$  to the verifier.
  - (b) The verifier chooses a random  $i \in \{1, 2\}$  and sends it to the prover.
    - (c) The prover computes a permutation  $\rho$  such that  $H = \rho(G_i)$ :
      - If  $i = 1$ , then  $\rho = \pi \circ \sigma$ ,
      - If  $i = 2$ , then  $\rho = \sigma$ .Then the prover sends  $\rho$  to the verifier.
    - (d) The verifier checks that  $H = \rho(G_i)$ .
3. The verifier accepts the input if in all the rounds  $H = \rho(G_i)$ .

### Example of A Zero Knowledge Interactive Proof (cont.)

**Completeness:** If the graphs are isomorphic, the prover can always provide an isomorphism, and the verifier accepts the input with probability 1.

**Soundness:** If the graphs are not isomorphic, then in case the prover chooses  $H$  as specified, the verifier can see that the permutation is wrong (since there is no right permutation).

### Example of A Zero Knowledge Interactive Proof (cont.)

**Question:** Can the prover lie, and deceive the verifier?

**Answer:** In order to lie, the prover must guess the value of  $i$  in advance, and give  $H = \sigma(G_i)$  for some  $\sigma$ . Since he has no way of doing it, then the verifier is wrong with probability  $\frac{1}{2}$  in each round. Since the choices are independent, the probability of getting the correct answers in all the rounds is  $2^{-n}$ .

**Polynomial verification:** The verifier can be implemented in polynomial time, from the same reasons of the previous proof.

**Result:** The above protocol is another interactive proof for the GI problem.

### Example of A Zero Knowledge Interactive Proof (cont.)

**So, what is the motivation beyond this complicated proof?**

At the end of the second proof, the verifier does not know the permutation that maps  $G_1$  to  $G_2$ .

This fact does not prevent him from being convinced that  $G_1$  and  $G_2$  are isomorphic.

Loosely speaking we say that after the proof the verifier does not know anything new about the instance, apart from whether the claim we wanted to prove is true or false.

### Perfect Zero-Knowledge Proofs — Motivation

Zero knowledge proofs, are proofs that yield no information apart from the validity of the claim we wanted to prove:

Given any input  $x$ , anything that the verifier can compute efficiently after the interaction with  $P$  on  $x$ , could also be computed before the interaction.

Showing a protocol is zero knowledge guarantees a high level of security for the protocol, since no matter what the verifier does, he does not get any new information about the prover's secrets.

### Perfect Zero-Knowledge Proofs — Motivation (cont.)

In order to show that the verifier gains no new knowledge we show that the verifier could generate the same interaction without the prover's help, and that the distribution of the generated interactions is identical to the distribution of the real interactions.

### Perfect Zero-Knowledge Proof — Definition

A **transcript**  $T$  of an interaction is the following:

1. The input.
2. The messages sent by the participants.
3. The random numbers used by the verifier.

Informally a transcript contains all the information that the verifier might have gained.

A polynomial time probabilistic machine  $M$  is called a **simulator** for an interaction of a verifier and a prover if for every  $x \in L$  the output of  $M$  is a transcript.

### Perfect Zero-Knowledge Proof — Definition (cont.)

An interactive proof system  $(P, V)$  is **Perfect Zero Knowledge** if:

1. For every probabilistic polynomial time machine  $V^*$ , there exists a simulator  $M$  of the interaction  $(P, V^*)$  for every  $x \in L$ .
2. The transcripts generated by  $M$  are distributed exactly as those generated in true interactions on  $x$ .

It is impossible to distinguish a real transcript from a simulated transcript when  $x \in L$ . Thus, anything that the verifier knows after the proof, could have been obtained by running the simulator without the prover.

When  $x \notin L$  a real cheating prover is almost always detected, but the simulator can still generate transcripts.

Hence, such proofs give no information to the verifier, except for the fact that the claim holds.

### Simulator for the GI Problem

We prove that the proof presented for the GI problem is Perfect zero knowledge, by giving a simulator for the problem.

The input for the simulator is an instance of the GI problem, and its output is a forged transcript of a proof (denoted by  $T$  in the algorithm). Note that any transcript has the form:

$$(G_1, G_2)(H_1, i_1, \rho_1) \dots (H_n, i_n, \rho_n)$$

### Simulator for the GI Problem (cont.)

1.  $T = (G_1, G_2)$
2. Do the following till  $n$  triples are found:
  - (a) Let  $j$  be the round index  $\{1, \dots, n\}$
  - (b) Choose  $i_j$  to be 1 or 2 at random
  - (c) Choose a random permutation  $\rho_j$
  - (d) Compute  $H_j = \rho_j(G_{i_j})$
  - (e) Call the original  $V$  with input  $H_j$  and obtain a challenge  $i'_j$
  - (f) If  $i_j = i'_j$ , concatenate the triple  $(H_j, i_j, \rho_j)$  to  $T$
  - (g) Otherwise, reset  $V$ 's state, and repeat this round with new random choices

### Simulator for the GI Problem (cont.)

We claim that the simulator's transcripts have exactly the same distribution as a true interaction transcripts.

Note that if the verifier is honest, we could avoid calling him in the simulation.

### Simulator for the GI Problem (cont.)

The proof is based on the following facts:

1. In each round the simulator has probability  $\frac{1}{2}$  to guess the correct bit as the verifier. Therefore, we expect to find a valid triple every two trials. This yields a polynomial bound on the expected running time of the simulator.
2. Both the simulator and the prover select the permutation at random. Therefore, the probability of selecting any particular graph is  $1/n!$ , provided that the selected graph in this round is  $H_j$ . This proves that we only need to ensure that the random bits have the same distribution as in a real interaction.
3. The simulator uses the verifier to check his bit. Since after each failure the verifier's state is reset, the distribution of the random bit is the same.

### Graph Non Isomorphism

**The Problem:** Given two graphs  $G_1, G_2$ , where  $V_1 = V_2 = n$ ,  $P$  wants to prove to  $V$  that no permutation  $\pi$  exists such that  $G_1 = \pi(G_2)$ .

**An interactive protocol for GNI:**

Repeat  $t$  times:

1. Both  $P$  and  $V$  get  $G_1, G_2$ .
2.  $V$  randomly chooses  $b \in \{1, 2\}$  and a permutation  $\pi$ .
3.  $V$  sends  $H = \pi(G_b)$  to  $P$ .
4.  $P$  returns  $b'$  to  $V$ , such that  $H$  is isomorphic to  $\pi(G_{b'})$ .
5. If  $b \neq b'$  then  $V$  rejects the proof.

### Graph Non Isomorphism (cont.)

- If  $G_1 \not\cong G_2$  then  $H$  is isomorphic to exactly one of them. Thus,  $P$  always sends the correct answer to  $V$ .
- If  $G_1 \cong G_2$  then  $H$  is isomorphic to both,  $P$  will send  $b' = b$  with probability  $\frac{1}{2}$ .

**Question:** Is this a ZK protocol?

### Graph Non Isomorphism (cont.)

**Answer:** No.

It is easy to simulate a round in the protocol for the honest verifier  $V$ :

$S_V$  calls  $V$  and receives a graph  $H$ , then randomly chooses a bit  $b$  as a reply. If  $V$  does not accept  $b$  reset  $V$  and repeat the process.

What about another verifier  $V^*$ ?

Consider the following scenario:  $V^*$  has a graph  $H$ , which he knows to be isomorphic to one of  $G_1, G_2$ . By sending  $H$  in the first round, he gets information which he could not have computed himself (assuming  $\text{GNI}, \text{GI} \notin \text{BPP}$ ), even though  $G_1 \not\cong G_2$ .

Thus, the above protocol is not a ZK protocol.

### Computational Zero-Knowledge Proof — Definition

Perfect zero knowledge is a very strong demand, and therefore we might be interested in a weaker model, which can be applied to a wider set of problems. Still, we want the new model to catch the notion that practically the prover does not give away any of its secret.

Given two random variables  $X, Y$  we say that they are **computationally indistinguishable** if for any polynomial number of samples of the variables, the distribution is the same.

Note that two variables can have a different distributions but still can be computationally indistinguishable.

### Computational Zero-Knowledge Proof — Definition (cont.)

An interactive proof system  $(P, V)$  is **Computational Zero Knowledge** if:

1. For every probabilistic polynomial time machine  $V^*$ , there exists a simulator  $M$  for the interaction  $(P, V^*)$  for every  $x \in L$ .
2. The transcripts generated by the simulator and the transcripts generated by a real interaction are **computationally indistinguishable**.

### IP=PSPACE

Every problem in PSPACE has a zero knowledge interactive proof protocol. Of course, the prover should be powerful enough to solve/generate the problems.

Problems interesting for cryptography are usually in NP, and can be generated easily, leaving a witness in the hands of the prover, who then uses only efficient computations. Thus, all the protocols we will see are in NP.

### Fiat-Shamir ZK Identification Scheme

Zero knowledge proofs can be used to cryptographically identify parties.

Each party has a secret key and a public key. The prover convinces the verifier that he knows his secret key, without revealing any information on his secret key that the verifier could not know otherwise (except that the proven claim holds).

### Fiat-Shamir ZK Identification Scheme (cont.)

In the Fiat-Shamir scheme, the prover has an RSA modulo  $n = pq$  whose factorization is secret. The factors themselves are not used in the protocol.

Unlike in RSA, a center can generate a universal  $n$ , used by everyone, as long as nobody knows the factorization. The center itself should forget the factorization just after he computes  $n$ .

### Fiat-Shamir ZK Identification Scheme (cont.)

**The Secret Key:** The prover chooses a random value  $1 < S < n$  (to be served as the secret key) ( $\text{gcd}(S, n) = 1$ ) and keeps it secret.

**The Public Key:** The prover computes  $I = S^2 \bmod n$ , and publishes the pair  $I$  and  $n$  as the public key.

**The purpose of the protocol:** The prover has to convince the verifier that he knows the secret key  $S$  corresponding to the public key  $(I, n)$ , (i.e., to prove that he knows a modular square root of  $I$  modulo  $n$ ), without revealing  $S$ .

### Fiat-Shamir ZK Identification Scheme (cont.)

#### **The Identification Protocol:**

The verifier wishes to authenticate the identity of the prover, which is claimed to have a public key  $I$ . Thus, he requests the prover to convince him that he knows the secret key  $S$  corresponding to  $I$ .

### Fiat-Shamir ZK Identification Scheme (cont.)

1. The prover chooses a random value  $1 < R < n$ , and computes  $X = R^2 \bmod n$ .
2. The prover sends  $X$  to the verifier.
3. The verifier requests from the prover one of the following requests at random:
  - (a)  $R$ , or
  - (b)  $RS \bmod n$ .
4. The prover sends the requested information to the verifier.

### Fiat-Shamir ZK Identification Scheme (cont.)

5. The verifier verifies that he received the correct answer by checking whether:

- (a)  $R^2 \stackrel{?}{=} X \pmod{n}$ , or
- (b)  $(RS)^2 \stackrel{?}{=} XI \pmod{n}$ .

6. If the verification fails, the verifier concludes that the prover does not know  $S$ , and thus he is not the claimed party.
7. This protocol is repeated  $t$  (usually 20, 30, or  $\log n$ ) times, and if in all of them the verification succeeds, the verifier concludes that the prover is the claimed party.

### The Protocol does not Reveal Information

We show that no information is revealed on  $S$  from the protocol:

Clearly, when the prover sends  $X$  or  $R$ , he does not reveal any information on  $S$ .

When the prover sends  $RS \bmod n$ :

1.  $RS \bmod n$  is random, since  $R$  is random and  $\text{gcd}(S, n) = 1$ .
2. If somebody can compute some information on  $S$  from  $I, n, X$ , and  $RS \bmod n$ , he can also compute the same information on  $S$  from  $I$  and  $n$ , since he can choose  $T = R'S \bmod n$  at random, and compute  $X' = T^2 I^{-1} \bmod n$ , from which he can compute the information on  $S$ .

Thus, the verifier, and anybody else, cannot gain any information on  $S$  using the protocol, or from the messages transmitted in the protocol.

### Security

Clearly, if the prover knows  $S$ , the verifier is convinced in his identity.  
If the prover does not know  $S$ , he can either

1. know  $R$ , but not  $RS \bmod n$ , as he is choosing  $R$ , but cannot multiply it by the unknown value  $S$ , or
2. choose  $RS \bmod n$ , and thus can answer the second question  $RS \bmod n$ , but in this case he cannot answer the first question  $R$ , since he needs to divide by the unknown value  $S$ .

### Security (cont.)

In any case, he cannot answer both questions, since then he can compute  $S$  as the ratio between the two answers. But it is assumed that computing  $S$  is difficult, actually the difficulty is equivalent to that of factoring  $n$ .

Since the prover does not know in advance (when he chooses  $R$  or  $RS \bmod n$ ) which question the verifier will ask, he cannot choose the required choice. He can succeed in guessing the verifier's question with probability  $1/2$  for each question, and thus the verifier can catch him in half of the times, and fails to catch him half of the times. The protocol is repeated  $t$  times, and thus the probability that the verifier fails to catch the prover in all the times is only  $2^{-t}$ , which is exponentially reducing with  $t$ .

### Security (cont.)

In particular, for  $t = 20$ , the prover succeeds to cheat less than once in a million trials, and for  $t = 30$ , the prover succeeds to cheat less than once in a billion trials. Verifiers wishing a smaller probability of error, can use larger  $t$ 's.

The verifier cannot use the information he received in the protocol to convince others that he is the original prover, since he cannot answer both questions  $R$  and  $RS \bmod n$  for any  $R$ . If he could, he would know  $S$ .

### A Simulator for the Fiat-Shamir Scheme

We prove that the Fiat-Shamir scheme is zero knowledge, by giving a simulator for the problem.

The input for the simulator are numbers  $I, N$ , which the prover claims to know the square root of  $I$  modulo  $N$ . The output of the simulator is a forged transcript of a proof. A transcript for the problem is of the form:

$$(I, N)(X_1, i_1, M_1) \dots (X_n, i_n, M_n)$$

Where  $M_i$  is either the square root of  $X_i$  (in case  $i_i = 1$ ), or the square root of  $IX_i$  (in case  $i_i = 2$ ).

### A Simulator for the Fiat-Shamir Scheme (cont.)

1.  $T = (I, N)$
2. Do the following till  $n$  triples are found:
  - (a) Let  $j$  be the round index  $\{1, \dots, n\}$
  - (b) Choose  $i_j$  to be 1 or 2 at random
  - (c) Choose a random number  $1 < R_j < n$
  - (d) Compute  $U_j = R_j^2$  if  $i_j = 1$ , and  $U_j = R_j^2 I^{-1}$  if  $i_j = 2$
  - (e) Call the original  $V$  with input  $U_j$  and obtain a challenge  $i'_j$
  - (f) If  $i_j = i'_j$ , concatenate the triple  $(U_j, i_j, R_j)$  to  $T$
  - (g) Otherwise, reset  $V$ 's state, and repeat this round with new random choices

### A Simulator for the Fiat-Shamir Scheme (cont.)

The correctness of the simulator is derived from the following facts:

1. The expected running time is polynomial for the same reasons we gave in the GI simulator proof.
2. In each round the relation between  $R_j$  and  $U_j$  can be verified correctly.
3. The simulator does not need to know the root of  $I$ . Even if the chosen bit is 2, still the equation  $U_j I = R_j^2$  holds. Moreover, it is not detected even if  $I$  is a quadratic non-residue.
4. The distribution of the transcripts is the same as the distribution of real transcripts, since the random bit distribution is the same.

### Parallel Fiat-Shamir

We can apply all the rounds of Fiat-Shamir in parallel, instead of sending them sequentially. This modification makes the protocol more efficient.

Is this modified protocol zero-knowledge?

### Parallel Fiat-Shamir (cont.)

Assume we have a ZK system, for which the honest prover can always respond to  $V$ 's challenges, where as a dishonest prover can fool  $V$  with probability  $\frac{1}{2}$  in every round.

After  $n$  rounds the dishonest prover can fool  $V$  with probability  $2^{-n}$ .

Can this protocol be executed in parallel and still remain ZK?

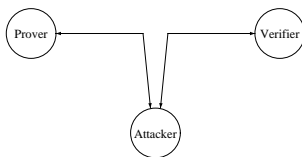
**Partial Answer:**

We cannot use the simulator from the original protocol, because this simulator has probability of  $2^{-n}$  to succeed. Thus, we get exponential expected running time.

In the case of Parallel Fiat-Shamir: Parallel Fiat-Shamir is not ZK.

### An Active Attack

The only attack that some party can do is to actively use the protocol with both the prover and the verifier, to convince the verifier he is the prover, asking the prover to do the real work:



In this attack, the attacker sends all the verifier's questions to the real prover, and all the answers of the prover are sent to the verifier. When the identification ends, the attacker can act as if he is the real prover.

### ZK Proofs of Knowledge

The Fiat-Shamir protocol convinces the verifier that the prover knows the square root of  $I$ , without revealing any information on  $S$ . However, the verifier gets one bit of information: he learns that  $I$  is a quadratic residue.

### ZK Proofs of Knowledge (cont.)

The following scheme does not even reveal whether  $I$  is a quadratic residue or not — it reveals only that the prover knows whether it is a quadratic residue or not:

The moduli  $n = pq$  is chosen such that both  $p$  and  $q$  are of the form  $4m+3$  (i.e.,  $n$  is a Blum integer). Such moduli have the property that  $-1$  is a quadratic non-residue whose Jacobi symbol modulo  $n$  is  $+1$  (since  $-1$  is a quadratic non-residue modulo  $p$  nor  $q$ ). Thus, it is difficult to distinguish which of two numbers: a quadratic residue and its negation is the quadratic residue.

### ZK Proofs of Knowledge (cont.)

**The Secret Key:** The prover chooses  $k$  random values  $S_1, S_2, \dots, S_k$ , where  $1 < S_i < n$ , and keeps them secret.

**The Public Key:** The prover computes  $I_i = \pm 1/S_i^2 \pmod{n}$ , where the sign is chosen randomly and independently, and publishes  $I_1, I_2, \dots, I_k$  and  $n$  as the public key.

In this protocol,  $k$  secrets are proved in parallel, resulting with a smaller probability of cheating in each iteration. (However,  $k$  should be kept constant to keep it ZK, due to the details of the definition of ZK).

### ZK Proofs of Knowledge (cont.)

#### The Identification Protocol:

1. The prover chooses a random value  $1 < R < n$ , and computes  $X = \pm R^2 \pmod{n}$ .
2. The prover sends  $X$  to the verifier.
3. The verifier sends a random boolean vector  $E_1, E_2, \dots, E_k$ .
4. The prover sends

$$Y = R \cdot \prod_{E_j=1} S_j \pmod{n}.$$

5. The verifier verifies that

$$X = \pm Y^2 \cdot \prod_{E_j=1} I_j \pmod{n}.$$

### ZK Proofs of Knowledge (cont.)

6. If the verification fails, the verifier concludes that the prover does not know  $X$ , and thus he is not the claimed party.
7. This protocol is repeated  $t$  times, and if in all of them the verification succeeds, the verifier concludes that the prover is the claimed party.

In this protocol, the cheating probability is  $2^{-k}$  for each iteration, and thus after  $t$  iterations the cheating probability is  $2^{-kt}$ .

### ZK Proofs of Knowledge (cont.)

In this protocol, the values  $I_i$ ,  $X$  and  $Y$  can be any numbers with Jacobi symbol  $+1$ , unlike in the original scheme in which they could be only half of the numbers with Jacobi symbol  $+1$  (i.e., the quadratic residues).

This is a zero knowledge protocol, since if the prover can answer two distinct questions, for two distinct values of the boolean vector  $E_1, \dots, E_k$ , he can compute the square root of a product of a subset of the  $I_i$ 's.

### Permuted Kernels Identification Scheme

This scheme is particularly designed to be applicable on smart cards, with a small memory and slow speed.

#### Notation:

1. Upper case letters denote vectors and matrices.
2. Lower case letters denote scalars.

### Permuted Kernels Identification Scheme (cont.)

3. Greek letters denote permutations over  $\{1, \dots, n\}$ . Their effect  $V_\pi$  on the vector (of length  $n$ )  $V$  is the vector  $W$  such that

$$w_j = V_{\pi(j)}.$$

The effect of permutations on matrices is defined as the permutation of *columns*

$$A_\pi = [a_{i,\pi(j)}].$$

Thus, the product of the permuted matrix by the permuted vector is

$$A_\pi V_\pi = AV.$$

### Permuted Kernels Identification Scheme (cont.)

4. All arithmetic is done modulo a small prime  $p$ .
5. The kernel  $K(A)$  of an  $m \times n$  matrix  $A$  is the set of vectors  $W$  such that  $AW = 0 \pmod{p}$ . In particular,  $K(A_\sigma) = (K(A))_\sigma$ .

### The Permuted Kernels Problem (PKP)

**Given:** an  $m \times n$  matrix  $A$ , a  $n$ -vector  $V$  and a prime  $p$ .

**Find:** a permutation  $\pi$  such that  $V_\pi \in K(A)$ .

It is easy to find vectors in  $K(A)$ .

However, it is difficult to find a vector in  $K(A)$  with the particular entries from  $V$ . This problem is NP-complete even for  $m = 1$  and  $V$  containing only  $+1$ 's and  $-1$ 's (since it can be reduced to the partition problem).

This problem is NP-complete in the strong sense, i.e., its difficulty grows exponentially in  $p$  rather than in  $\log p$  (under appropriate assumptions). Thus, small numbers can be used in the scheme, without reducing its security.

### The Identification Scheme

All users agree on a universal matrix  $A$ , and a prime  $p$ .

Each user chooses a random permutation  $\pi$  as his secret key.

Each user finds a random vector  $V$  such that  $V_\pi \in K(A)$ .  $V$  serves as his public key.

The protocol uses a cryptographic hash function, which is used to commit the prover to his chosen values, without revealing any information on the values to the verifier.

### The Identification Scheme (cont.)

**The protocol:**

1. The prover chooses a random vector  $R$  and a random permutation  $\sigma$ , and sends the hashed values of the pairs  $(\sigma, AR)$  and  $(\pi\sigma, R_\sigma)$  to the verifier.
2. The verifier chooses a random value  $0 \leq c < p$  and asks the prover to send  $W = R_\sigma + cV_{\pi\sigma}$ .
3. After receiving  $W$ , the verifier asks the prover to reveal either  $\sigma$  or  $\pi\sigma$ . In the first case the verifier checks that  $(\sigma, A_\sigma W)$  hashes to the first given value. In the second case the verifier checks that  $(\pi\sigma, W - cV_{\pi\sigma})$  hashes to the second given value.

### Security and Implementation

An honest prover, who knows  $\pi$  can always pass the tests, since  $A_\sigma W = A_\sigma(R_\sigma + cV_{\pi\sigma}) = A(R + cV_\pi) = AR + cAV_\pi = AR$  and  $W - cV_{\pi\sigma} = R_\sigma$ .

A dishonest prover should be able to answer  $2p$  distinct questions. He can choose the answers of  $p + 1$  questions (one answer for each  $0 < c < p$ , both answers for  $c = 0$ ). If he can answer correctly  $p + 2$  questions, he can find  $\pi$ . Thus, his probability of success is at most  $(p + 1)/2p \approx 1/2$ .

In this protocol the recommended settings are  $p = 251$ ,  $n = 64$ , and  $m = 37$ .

### Security and Implementation (cont.)

**Notes:**

The best known attack on the permuted kernel problem with this setting requires about  $2^{116}$  steps with  $2^{65}$  memory. With a smaller memory it runs slower.

The matrix  $A$  should be randomly chosen. In particular, it can be chosen as a pseudo-random function which can compute each of its entries when it is required, thus the storage of  $A$  in memory is not explicitly required.