# Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles

Ronen Gradwohl* Moni Naor† Benny Pinkas‡

**Abstract**

We consider various cryptographic and physical zero-knowledge proof schemes for Sudoku, a popular combinatorial puzzle. The cryptographic protocols are direct and efficient, and the physical protocols are meant to be understood by "lay-people" and implementable without the use of computers.

## 1 Introduction

Sudoku is the name of a combinatorial puzzle that has swept the world in 2005 (especially via newspapers, where it appears next to crossword puzzles), following the lead of Japan (see the Wikipedia entry [15] or the American Scientist article [8]). In a Sudoku puzzle the challenge is a $9 \times 9$ grid subdivided into nine $3 \times 3$ subgrids. Some of the cells are already set with values in the range $1$ through $9$ and the goal is to fill the remaining cells with numbers $1$ through $9$ so that each number appears exactly once in each row, column and subgrid. Part of the charm and appeal of Sudoku appears to be the ease of description of the problems, as compared to the time and effort it takes a human individual to solve them.

A natural issue, at least for cryptographers, is how to convince someone that you have solved a Sudoku puzzle without revealing the solution. In other words, the questions of interest here are: how can a prover show (i) that there is a solution to the given puzzle, and (ii) that he knows the solution, while not giving away any information about the solution? In this paper we consider several types of methods for doing just that. Broadly speaking, the methods are either *cryptographic* or *physical*. By a *cryptographic* protocol we mean one in the usual model found in the foundations of cryptography literature. In this model, two machines exchange messages and the security of the protocol relies on computational hardness (see Goldreich [5] for an accessible account and [6] for a detailed one). By a *physical* protocol we mean one that is implementable by humans using common objects, and preferably without the aid of computers. In particular, our protocols utilize scratch-off cards, similar to those used in lotteries.

*Department of Computer Science and Applied Math, The Weizmann Institute of Science, Rehovot 76100, Israel; email: ronen.gradwohl@weizmann.ac.il. Research supported by US-Israel Binational Science Foundation Grant 2002246.

†Incumbent of the Judith Kleeman Professorial Chair, Department of Computer Science and Applied Math, The Weizmann Institute of Science, Rehovot 76100, Israel; email: moni.naor@weizmann.ac.il. Research supported in part by a grant from the Israel Science Foundation.

‡Department of Computer Science, University of Haifa, Haifa, Israel; email: benny@pinkas.net, benny@cs.haifa.ac.il.

**This Work:** The general problem of Sudoku (on an $n \times n$ grid) is in the complexity class NP, which means that given a solution it is easy to *verify* that it is correct. (In fact, Sudoku is known to be NP-Complete [16], but we are not going to use this fact, at least not explicitly.) Since there are cryptographic zero-knowledge proofs for all problems in NP [7], there exists one for Sudoku, via a reduction to 3-Colorability or some other NP-Complete problem with a known zero-knowledge proof. In this work, however, we are interested in more than the mere existence of such a proof, but rather its efficiency, understandability, and practicality, which we now explain.

First, the benefits of a direct zero-knowledge proof (rather than via a reduction) are clear, as the overhead of the reduction is avoided. Thus, the size of the proof can be smaller, and the computation time shorter. In addition, we wish our proofs to be easy to understand by "non-experts". This is related to the practicality of the proof: the goal is to make the interaction implementable in the real world, perhaps even without the use of a computer. One of the important aspects of this implementability is that the participants have an intuitive understanding of the correctness of the proof, and thus are convinced by it, rather then relying blindly "on the computer". For another example in which this intuitive understanding is important, see the work of Moran and Naor [10] on methods for polling people on issues for which their answers may be embarrassing.

The contributions of this paper are efficient cryptographic protocols for showing knowledge of a solution of a Sudoku puzzle which do not reveal any other useful information (these are known as zero-knowledge proofs of knowledge) and several transparent physical protocols that achieve the task.

**Organization:** The rest of the paper is organized as follows: in Section 2 we give some definitions, and then in Section 3 we describe two cryptographic zero-knowledge proofs. The first is very simple and direct, and the second is slightly more involved, but has a lower probability of error. Finally, in Section 4 we describe several physical protocols, using envelopes and scratch-off cards.

## 2 Definitions

**Sudoku:** An instance of Sudoku is defined by the size $n = k^2$ of the $n \times n$ grid, where the subgrids are of size $k \times k$. The indices, values in the filled-in cells and the values to be filled out are all are in the range $\{1 \ldots n\}$. Note that in general the size of an instance is $O(n^2 \log n)$ bits and this is the size of the solution (or witness) as well.

**Cryptographic Functionalities:** We only give rough descriptions of zero-knowledge and commitments. For more details see the books by Goldreich [5] and [6], Chapter 4 or the writeup by Vadhan [14]. In general, a zero-knowledge proof is an interactive-proof between two parties, a *prover* and a *verifier*. They both know an instance of a problem (e.g. a Sudoku puzzle) and the prover knows a solution or a witness. The two parties exchange messages and at the end of the protocol the verifier '*accepts*' or '*rejects*' the execution. The *completeness* of the protocol is the probability that an honest verifier accepts a correct proof, i.e. one done by a prover holding a legitimate solution and following the protocol. All our protocols will have perfect completeness, in which the probability of accepting a correct proof is 1. The *soundness* of the protocol is the (upper bound on the) probability that a verifier accepts an incorrect proof, i.e. a proof to a fallacious statement; in our case this is the statement that the prover knows a solution to the given Sudoku puzzle, even though it does not know such a solution. In addition, the verifier should not gain any new knowledge from the interaction (i.e. *zero-knowledge*); this

means that there is an efficient *simulator* that could have generated the conversation (or an indistinguishable one) without the interaction. Our protocols should also be *proofs-of-knowledge*: if the prover (or anyone impersonating him) can succeed in making the verifier accept, then there is an *extractor* that can communicate with the prover and actually come up with the solution itself (this may involve running the prover several times using the same randomness, which is not possible under normal circumstances).

The only cryptographic tool used by our proofs is a commitment protocol. A commitment protocol allows one party, the sender, to commit to a value to another party, the receiver, with the former not learning anything meaningful about the value. Such a protocol consists of two phases. The *commit* phase, following which the sender is bound to some value $v$, while the receiver cannot determine anything useful about $v$ and in particular cannot distinguish between the case $v = b$ and $v = b'$. Later on, the two parties may perform a *decommit* or *reveal* phase, after which the receiver obtains $v$ and is assured that it is the original value; in other words, once the *commit* phase has ended, there is a unique value that the receiver will accept in the *reveal* phase.

Note that in this setting we think of the adversary as being malicious in his actions and the guarantees we make (both against a cheating prover trying to sneak in a fallacious proof and against a cheating verifier trying to learn more than it should) are with respect to any behavior.

**Physical Protocols:** While the cryptographic setting is pretty standard, when discussing 'physical' protocols there are many different options, ranging from a deck of cards [3, 13] to a PEZ dispenser [1], a newspaper [12], and more (see [9] for a short survey). In our setting we will be using tamper-evident sealed envelopes, as defined by Moran and Naor [9]. It is simplest to think of these as scratch-off cards: each card has a number on it from $\{1, \ldots, n\}$, but that number cannot be determined unless the card is scratched (or the envelope is opened and the seal is broken). We would like our physical protocols to enjoy zero-knowledge properties as well. For this to be meaningful we have to define the power of the physical objects that the protocol assumes as well as the assumptions on the behavior of the humans performing it. In general, the adversarial behavior we combat is more benign than the one in the cryptographic setting. See details in Section 4.

# 3 Cryptographic Protocols

We provide two cryptographic protocols for Sudoku. The setting is that we have a prover and a verifier who both know an instance of an $n \times n$ Sudoku puzzle, i.e. a bunch of cells with values. The prover knows a solution to the instance and the verifier wants to make sure that (i) a solution exists and (ii) the prover knows the solution.

The protocols presented are in the standard cryptographic setting. The structure of the proof is the usual:

1. Prover commits to a bunch of values. They are a function of the instance, the solution and some randomization.
2. Verifier asks to open some of them – this is called the challenge.
3. Prover opens the required values.
4. Verifier makes some consistency checks with the given instance and accepts or rejects accordingly.

The only cryptographic primitive we use in both protocols is bit or string *commitment* as described above. Bit commitments can be based on any one-way function [11] and are fairly efficient to implement. To prove zero-knowledge of a protocol we use the 'standard' argument, due to [7]: for this we need that the distribution of the values opened in Step 3 be easy to describe as a function of the instance and the challenge of Step 2 (for example that it will be a random permutation of $\{1 \dots n\}$). If the number of possible challenges is not large (polynomial in the instance size), then this property together with the indistinguishably property of the commitment protocol imply the existence of an efficient simulator. (The simulator operates in the following way: it picks at random a challenge that the verifier might send in Step 2, and computes commitments for Step 1 which agree with this challenge. It sends these commitments to the receiver and is given the challenge of Step 2. If this is the challenge it prepared to it can continue with the protocol, and otherwise it resets the simulation and starts it all over again. If the number of possible challenges is polynomial in the instance size, this process is expected to succeed in a polynomial number of tries.)

The communication complexity and computation time of both protocols is similar, and is $O(n^2 \log n)$. However, the first protocol has a relatively high soundness (the prover can cheat with probability $(1 - 1/(3n + 1))$), while the second protocol has constant probability of catching a cheater. In both cases the soundness can be decreased by repeating the protocols several times, either sequentially or in parallel (for parallel repetition more involved protocols have to be applied (see [6]) to preserve the zero-knowledge property). Therefore, to reduce the cheating probability to $\varepsilon$, the first protocol has to be repeated $O(n \log(1/\varepsilon)$ times and the resulting communication complexity is $O(n^3 \log n \log 1/\varepsilon)$ bits, while the second protocol should be repeated only $O(\log 1/\varepsilon)$ times, and the resulting communication complexity is $O(n^2 \log n \log 1/\varepsilon)$ bits.

### 3.1 A protocol based on coloring

The following protocol is an adaptation of the well-known GMW zero-knowledge proof of 3-Colorability of a graph [7] (see [6]) for Sudoku puzzles. Recall that the idea there was for the prover to randomly permute the colors and then commit to the color of each vertex. The verifier picks a random edge and checks that its two end points are colored differently. To apply this idea in the context of Sudoku it helps to think of the graph as being partially colored to begin with. So the protocol consists of the prover randomly permuting the numbers and committing to the resulting solution. What the verifier checks is either the correctness of the values of one of the rows, columns or subgrids, or consistency with the filled-in values.

The protocol operates in the following way:
**Prover:**

1. Prover chooses a random permutation $\sigma : \{1, \dots, n\} \mapsto \{1, \dots, n\}$.
2. For each cell $(i, j)$ with value $v$, prover sends to verifier a commitment for the value $\sigma(v)$.

**Verifier:** Chooses at random one of the following $3n + 1$ possibilities: a row, column or subgrid ($3n$ possibilities), or 'filled-in cells', and asks the prover to open the corresponding commitments. After the prover responds, in case the verifier chose a row, column or subgrid, he verifies that all values are indeed different. In case the verifier chose the filled-in cells option, it checks that cells that originally had the same value still have the same value (although the value may be different), and that cells with different values are still different, i.e. that $\sigma$ is indeed a permutation.

*Proof sketch:* The perfect completeness of the protocol is straightforward. Soundness follows from the fact that any cheating prover must cheat either in his commitments for a row, column, subgrid, or the filled-in cells (namely, there is at least one question of the verifier for which the prover cannot provide a correct answer). Thus, the verifier catches a cheating prover with probability at least $1/(3n+1)$. Note also that the protocol is a proof of knowledge, since if *all* the $3n+1$ queries can be answered properly, then it is possible to find a solution to the original puzzle (simply find a reverse permutation $\sigma^{-1}$ mapping the filled-in values). The distribution on the values of the answer when the challenge is a row, column or subgrid is simply a random permutation of $\{1 \dots n\}$. The distribution in case the challenge is filled-in cells is a random injection of the values appearing in those cells to $\{1 \dots n\}$. Therefore the zero-knowledge of the protocol follows the standard arguments. The witness/solution size, as well as the number of bits committed, are both $O(n^2 \log n)$ bits.

### 3.2 An efficient cryptographic protocol with constant soundness

Below is a more efficient zero-knowledge protocol for the solution of a Sudoku puzzle. It is closest in nature to the Hamiltonian Paths protocol of Blum [2]. The protocol described has constant $(2/3)$ soundness for an $n \times n$ Sudoku problem, and its complexity in terms of the number of bits committed to is $O(n^2 \log n)$, which is also the witness/solution size.

The idea of the protocol is to triplicate each cell, creating a version of the cell for the row, column and subgrid in which it participates. The triplicated cells are then randomly permuted and the prover's job is to demonstrate:

- That the cells corresponding to the rows, columns and subgrids have all possible values.
- That the three copies of each cell have the same value.
- That the cells corresponding to the predetermined values indeed contain them.

The following protocol implements these idea:

**Prover:**

1. Commit to $3n^2$ values $v_1, v_2 \dots v_{3n^2}$ where each cell of the grid corresponds to three randomly located indices $(i_1, i_2, i_3)$. The values of $v_{i_1}, v_{i_2}$ and $v_{i_3}$ should be the value $v$ of the cell in the solution.
2. Commit to $n^2$ triples of *locations* in the range $\{1 \dots 3n^2\}$, where each triple $(i_1, i_2, i_3)$ corresponds to the locations of a cell of the grid in the list of commitments of Item 1.
3. Commit to the names of the grid cells of each triple from Item 2.
4. Commit to $3n$ sets of locations from Item 1, corresponding to the rows, columns and subgrids, where each set is of size $n$ and no two cells intersect.

**Verifier:** Ask one of the following three options at random:

a. Open all $3n^2$ commitments of Item 1 and the commitments of Item 4. When the answer is received, verify that each set contains $n$ different numbers.

b. Open all $3n^2$ commitments of Item 1 and the commitments of Item 2. When the answer is received, verify that each triple contains the same numbers.

c. Open the commitments of Items 2, 3 and 4 as well as the commitments of Item 1 corresponding to filled-in cells in the Sudoku puzzle. When the answer is received, verify the consistency of the commitments with (i) the predetermined values, (ii) the set partitions of 4 and (iii) the naming of the triples.

Option (a) takes care of the constraint that all values should appear in each row, column and and subgrid. Option (b) makes sure that the value of the cell is consistent in its three appearances. Option (c) makes sure that the filled-in cells have the correct value and that the partitioning of the cells to rows, columns and subgrids is as it should be. Therefore, if all three challenges (a,b and c) are met, then we have a solution to the given Sudoku puzzle, and this is a proof-of-knowledge as well. If the prover does not know a solution to the puzzle, then with probability at least $1/3$ the verifier rejects, and the probability of cheating is at most $2/3$. As before, perfect completeness of the protocol is straightforward. Note that for each challenge it is easy to describe the distribution on the desired response, and so the zero-knowledge of the protocol follows the standard arguments.

# 4 Physical Protocols

The protocols described in Section 3 can have a physical analog, given some physical way to implement the commitments. The problematic point is that tests such as checking that the set partitions and the naming of the triples are consistent (needed in challenge (c) of the protocol in Section 3.2) are not easy for humans to perform. In this section we describe protocols that are designed with human execution in mind, taking into account the strengths and weaknesses of such beings.

A locked box is a common metaphoric description of bit (or string) commitment, where the commiter puts the hidden secret inside the box, locks it, keeps the key but gives the box to the receiver. At the *reveal* stage he gives the key to the receiver who opens it. The problem with this description is that the assumption is that the receiver can *never* open the box without the key. It is difficult to imagine a physical box with such a guarantee that is also readily available, and its operation transparent to humans[1]. A different physical metaphor was proposed by Moran and Naor [9], who suggested concentrating on the *tamper-evident* properties of sealed envelopes and scratch-off cards. That is, anyone holding the envelope can open them and see the value inside, but this act is not reversible and it will be transparent to anyone examining the envelope in the future. Another property we require from our envelopes is that they be indistinguishable, i.e. it should be impossible to tell two envelopes apart, at least by the party that did not create them (this is a little weaker than the indistinguishable envelope model formalized in [9]).

Another distinction between our physical model and the cryptographic one has to do with the way in which we regard the adversary. Specifically, the adversary we combat in the physical model is more benign than the one considered in the cryptographic setting or the one in [9, 10]. We can think of our parties as not wanting to be labelled 'cheaters', and so the assurance we provide is that either the protocol achieves its goal or the (cheating) party is labelled a cheater.

Furthermore, we use the envelopes in a different manner from that described in [9, 10]. We think of the prover and verifier as being present in the same room, and in particular the protocols we describe are *not* appropriate for execution over the postal system (while the protocols of [9, 10] are). The presence of the two parties in the same room is required since the protocols use such operations as shuffling a

[1] Perhaps quantum cryptography can yield an approximation to such a box, but not a perfect one.

given set of envelopes - one party wants to make sure that the shuffle is appropriate, while the other party wants to make sure that the original set of envelopes is indeed the one being shuffled.

Other than the different view of the adversary, in our protocols we also need a couple of additional functionalities that are not included in the model of [9, 10]: *shuffle* and *triplicate*. The *shuffle* functionality is essentially an indistinguishable shuffle of a set of seals. Suppose some party has a sequence of seals $L_1, \ldots, L_i$ in his possession. Invoking the *shuffle* functionality on this sequence is equivalent to picking $\sigma \in_R S_i$, i.e. a random permutation on $i$ elements, to yield the sequence $L_{\sigma(1)}, \ldots, L_{\sigma(i)}$. The *triplicate* functionality is used only in our last protocol, so we defer its description to Section 4.3.

In the physical setting described above, the definition of zero-knowledge can be made rigorous. As in the cryptographic case, we need to come up with a simulator that can emulate the interaction between the prover and verifier. We will describe the simulators in Sections 4.1 and 4.3.

Finally, since we wish our protocols to also be proofs-of-knowledge, we will describe *extractors* that interact with honest provers in the physical setting and extract a correct solution for the Sudoku instance.

### 4.1 A physical zero-knowledge protocol with constant soundness

In the following protocol, the probability that a cheating prover will be caught is at least $8/9$. The main idea is that each cell should have three (identical) cards; instead of running a subprotocol to check that the values of each triple are indeed identical we let the verifier make the assignment of the three cards to the corresponding row, column and subgrid at random.

The protocol operates in the following way:

- The prover places three scratch-off cards on each cell. On filled-in cells, he places three cards with the correct value, which are already open (scratched).
- For each row/column/subgrid, the verifier chooses (at random) one of the three cards of each cell in the corresponding row/column/subgrid.
- The prover makes packets of the verifier's requested cards (i.e. for every row/column/subgrid, he assembles the requested cards). He then shuffles each of the $3n$ packets separately (using the *shuffle* functionality), and hands the shuffled packets to the verifier.
- The verifier scratches all the cards in each packet, and verifies that in each one, all numbers appear.

Perfect completeness is straightforward.

**Soundness:** We claim that the soundness of the protocol is $1/9$. We first describe a simple argument that the soundness is $1/3$ and then provide a more involved analysis showing that it is indeed $1/9$. The only way a cheating prover can cheat is by placing three cards that are not all of the same value on a cell, say cell $a$. This means that in this cell at least one value $y$ must be different from all others. Suppose that for all other cells the verifier has already assigned the cards to the rows, columns and subgrids. A necessary condition for the (cheating) prover to succeed is that given the assignments of all cells except $a$ there is exactly one row, column or subgrid that needs $y$ to complete the values in $\{1 \ldots n\}$. The probability that for cell $a$ the verifier assigns $y$ to the row, column or subgrid that needs it is $1/3$.

We now provide a more involved argument that shows that the soundness is actually $1/9$. We know that there is a cell where not all three values are the same. Also, the total number of cards of each value

must be correct, otherwise the prover will be caught with probability $1$. Thus, there must be at least two cells on which the prover cheats, say $a$ and $b$. We now consider different ways in which a prover can cheat on these cells, and show that his success probability is bounded above by $1/9$.

First suppose the prover cheats on exactly two cells, say $a$ and $b$, and suppose the values are $(x, x, y)$ for cell $a$ and $(y, y, x)$ for cell $b$. Note that this is the only way he can cheat on exactly two cells without being caught with probability 1. There are three possibilities for the location of cells $a$ and $b$, and we analyze the probability of being caught for each.

We will often assume the verifier has assigned all values to packets except those of cells $a$ and $b$, and then analyze the probability that he makes the correct assignments of those cells. Before assigning these two cells, however, we have some incomplete packets. We will say that a packet that has all values except some value $x$ "needs" $x$.

(i) The simplest case, cells $a$ and $b$ are not in the same row, column, or subgrid, and are thus "independent" in some sense. Suppose the verifier already assigned every card to a row/column/subgrid except the cards of cells $a$ and $b$. Then there are six packets that are not yet complete – 2 each for a row, column, and subgrid. But each one of these packets can have only 1 value that will yield a complete set, since it cannot be missing both an $x$ and a $y$ (if it does, then the final card will not complete the packet regardless, and the cheating prover will be caught). Thus, the only way the prover will not be caught is if the verifier assigns $x$ to the rows/columns/subgrids that need $x$, and $y$ to the ones that need $y$. But this happens with probability at most $1/9$, and so the probability of being caught is at least $8/9$.

(ii) In this case, cells $a$ and $b$ are in the same row, column or subgrid (exactly one of them). Without loss of generality, assume they are in the same row, and again that the verifier already assigned every card to a row/column/subgrid except the cards of cells $a$ and $b$. Here there are several options:

- If the column and subgrid of cell $a$ both need $x$, and the column and subgrid of cell $b$ both need $y$, then the verifier makes the correct assignment with probability $1/9$. This is because in order to accept, the verifier needs to assign $x$ to the row of $a$ and $y$ to the row of $b$, and each occurs independently with probability $1/3$.
- If the column of cell $a$ needs $x$ and the subgrid needs $y$ (or vice verse), and the column of cell $b$ needs $x$ and its subgrid needs $y$ (or vice versa), then again the verifier makes the correct assignment with probability $1/9$: He chooses $x$ for cell $a$'s row and $y$ for cell $b$'s row with probability $4/9$, since each assignment is made independently with probability $2/3$. He then makes the remaining assignments correctly with probability $1/4$, since each assignment is made independently with probability $1/2$.
- Any other situation results in the prover losing with probability 1, as there is no way to select the cards to satisfy all constraints.

(iii) In the final case, cells $a$ and $b$ are in the same row (or column) and the same subgrid. Without loss of generality, assume they are in the same row and subgrid. Consider the following situations:

- Suppose cell $a$'s column needs $y$ and cell $b$'s column needs $x$. In this case, the verifier makes the correct assignment with probability $1/9$, since each assignment is made with probability $1/3$.

– Now suppose the column of cell $a$ needs $x$ and the column of cell $b$ needs $y$. In this situation, however, the prover did not really need to cheat: he could have placed $(x, x, x)$ on cell $a$, and $(y, y, y)$ on cell $b$, and the constraints on rows, columns, and subgrids would have been satisfied. However, since we are assuming the prover does not know a correct solution to the Sudoku problem, there must be some other cells on which he is cheating.

Thus, either the correct assignment is made with probability $1/9$, or some additional cells have multiple-valued cards on them (in which case we can repeat the analysis for those cells). In either case, if the prover does not lose with probability 1, he is caught with probability at least $1/9$.

Thus, if the prover cheats on exactly two cells, he is caught with probability at least $8/9$. We now argue that this is also true if he cheats on three or more of the cells. Let $a$ and $b$ be two of the cheating cells. The values may be $(x, x, y)$ and $(y, y, x)$ as above, they may be $(x, x, y)$ and $(y, y, z)$, or one or both of the cells may have three distinct values. In any case, we can do the same analysis as above regarding the location of the two cells. A similar type of proof goes through, in some cases with even lower probabilities of success for the cheating prover.

In all the above possibilities, the prover is caught with probability at least $8/9$ and hence the soundness is $1/9$.

**Zero-Knowledge:** In order to show that the protocol above is zero-knowledge, we now describe a simulator. The simulator interacts with a cheating verifier, runs in probabilistic polynomial time, and produces an interaction that is indistinguishable from the verifier's interaction with the prover. The simulator does not have a correct solution to the Sudoku instance, but he does have an advantage over the prover: before handing the shuffled packets to the verifier, he is allowed to swap the packets for different ones. This advantage is similar to the ability of simulators to "rewind" the verifier in cryptographic zero-knowledge protocols. Such a simulator suffices in order to prove the zero-knowledge property of the protocol because of the following: since the simulator produces an indistinguishable interaction (except for the swap) from that of the prover, whatever the cheating verifier could have potentially learned from the prover, he could also have learned from the simulator: The verifier could have run the simulator himself, and so he learns nothing from the prover that he could not have learned on his own. We now describe the simulator.

- The simulator places three *arbitrary* scratch-off cards on each cell.
- After the verifier chooses the cards for the corresponding packets, the simulator takes them and shuffles them (just as the prover does).
- Before handing the packets to the verifier, the simulator swaps each packet with a randomly shuffled packet of scratch-off cards, in which each card appears once. If there is a scratched card in the original packet, there is one in the new packet as well.

Note that the final packets, and therefore the entire execution, are indistinguishable from those provided by an honest prover, since the *shuffle* functionality guarantees that the packets each contain a randomly shuffled set of scratch-off cards.

**Knowledge extraction:** To show that the protocol constitutes a proof of knowledge, we describe the extractor for this protocol, which interacts with the prover to extract a solution to the Sudoku instance. After the prover places the cards on the cells, the extractor simply scratches all the cards. If the prover is honest, then the scratched-cards give a solution. Otherwise, there will be some cell with three cards that are not all the same number.

Finally, in terms of the complexity of the protocol, we utilize $3n^2$ scratch-off cards, and $3n$ shuffles by the prover. However, recall that we are interested in making the protocols accessible to humans. For a standard $9 \times 9$ Sudoku grid, this protocol requires 27 shuffles by the prover, which seems a bit much. Thus, we now give a variant of this protocol that reduces the number of shuffles to one.

### 4.2 Reducing the number of shuffles

We now describe a variant of the previous protocol, where the number of required shuffles is only one, at the expense of it using a larger set of envelopes (expected size $1.5n^2$) and with a higher soundness ($5/9$). The idea is to run the protocol as above, but then pick a random subset of the rows, columns and subgrids and perform the shuffle on all of them simultaneously.

- The prover places three scratch-off cards on each cell. On filled-in cells, he places three scratched cards with the correct value.
- For each row/column/subgrid, the verifier chooses (at random) one of the three cards for each cell in the corresponding row/column/subgrid.
- The prover makes packets of the verifier's requested cards (i.e. for every row/column/subgrid, he assembles the requested cards).
- The verifier marks each packet with probability $1/2$.
- The prover takes the marked packets, shuffles them all together, and hands them to the verifier.
- The verifier scratches all the cards and verifies that each number appears the correct number of times (namely, if $c$ packets were marked, each number must appear $c$ times).

As before, the protocol is perfectly complete, since an honest prover will always succeed. For analyzing the soundness, note that if the prover is cheating, then with probability $8/9$ (as above) there is at least one packet which is unbalanced. If this packet is marked, and no other unbalanced is marked, then the final count of values is unbalanced and the prover fails. However, we have to be a bit careful here, since there may be two or more unbalanced packets that, when marked together, balance each other out.

A more careful analysis shows that the cheating probability is at most $4/9$: With probability $8/9$, some packet, say $a$, is unbalanced. Now suppose the verifier has already gone through all other packets, and either marked them or not. Thus far, the marked packets are either balanced or unbalanced. If they are balanced, then with probability $1/2$ the verifier will mark packet $a$, and the final mix will be unbalanced. If the marked packets are unbalanced, then with probability $1/2$ the verifier will **not** mark the packet $a$, and again the final mix will be unbalanced. Thus, with probability $1/2$, the final mix will be unbalanced, and the verifier will be caught. Note that this was conditioned on the fact that some packet is unbalanced, so overall, the probability that a cheating prover will be caught is $8/9 \cdot 1/2 = 4/9$.

### 4.3 A physical zero-knowledge protocol with zero soundness

We now describe another physical zero-knowledge protocol, this time with the optimal soundness of $0$. This comes at the expense of a slightly stronger model, as we also make use of the *triplicate* functionality of the tamper-evident seals, which we now describe.

**Triplicate using a trusted setup:** It is simplest to view this functionality as using some supplementary "material" that a trusted party provides to the parties. For instance, if the Sudoku puzzles are published in a newspaper, the newspaper could provide this material to its readers. The material consists of a bunch of scratch-off cards with the numbers $\{1 \dots n\}$ ($3n$ of each value). The cards come in triples that are connected together with an open title card on top that announces the value. The title card can be torn off (see figure below). It is crucial that the three unscratched cards hide the same value, and that it is impossible to forge such triples in which the hidden numbers vary.


Figure 1: A scratch-off card with triplicate functionality.

**Triplicate without trusted setup:** Another way to achieve this functionality in the absence of a trusted party preparing the cards in advance is as follows. Suppose we have scratch-off cards as before, where underlying numbers are replaced by colors. (For example, the number '1' is represented by a circular scratch card, whose color, below the peel-off layer, is, say, yellow.) When the prover wishes to triplicate a card, he asks the the verifier to cut the card into three equally shaped parts. The point is that the partitioning should be *random*. Whenever a part is scratched off (as the protocol suggests) the verifier will reject if it does not see a uniformly colored part.

If this task is performed by humans (which is the objective of this procedure), then slight variations in shapes will most likely go unnoticed by the human eye. A cheating prover may cheat by coloring some third a different color from the rest. However, assuming the cards are circles, there are (infinitely) many places in which the verifier can cut the cards. Thus, the probability that he cuts along the border separating two different colors (which is the only way the prover will not be caught) is nearly zero.

Using the tamper-evident seals with the additional *shuffle* and *triplicate* functionalities, we now have the following protocol:

- The prover lays out the seals corresponding to the solution in the appropriate place. The seals

that are placed on the filled-in squares are scratched, and must be the correct value (otherwise the verifier rejects).

- The verifier then triplicates the seals (using the *triplicate* functionality).
- For each seal, each third is taken to be in its corresponding row/column/subgrid packet, and the packets are shuffled by the prover (using the *shuffle* functionality). The prover hands the packets to the verifier.
- The verifier scratches off the cards of each packet, and verifies that in each packet all numbers in $\{1 \dots n\}$ appear.

Note that the *triplicate* functionality solves the problem of the first physical protocol, by preventing the prover from assigning different values to the same cell. Therefore the prover has no way of cheating. Thus, the soundness of the protocol is $0$.

The simulator for this protocol is nearly identical to that of the protocol in Section 4.1, with the exception that the cards in the swapped packets are also formed using the *triplicate* functionality. Since we are assuming that triplicated cards are indistinguishable by the verifier, the packets swapped by the simulator will look the same to the verifier as the original packets.

**Acknowledgments.** We are grateful to Tal Moran and Guy Rothblum for helpful discussions and comments. We also thank Tobial Barthel and Yoni Halpern for providing the initial motivation for this work.

## References

[1] József Balogh, János A. Csirik, Yuval Ishai and Eyal Kushilevitz: *Private computation using a PEZ dispenser*, Theoretical Computer Science 306(1-3): 69-84 (2003)

[2] M. Blum, *How to Prove a Theorem So No One Else Can Claim It*, Proc. of the International Congress of Mathematicians, Berkeley, California, USA, 1986, pp. 1444–1451.

[3] Claude Crépeau, Joe Kilian, *Discreet Solitary Games*, Advances in Cryptology - CRYPTO'93, Lecture Notes in Computer Science 773, Springer, 1994, pp. 319–330.

[4] R. Fagin, M. Naor and P. Winkler, *Comparing Information Without Leaking It*, C. of the ACM, vol 39, May 1996, pp. 77–85.

[5] O. Goldreich, **Modern Cryptography, Probabilistic Proofs and Pseudorandomness**, Springer, Algorithms and Combinatorics, Vol 17, 1998.

[6] O. Goldreich, **Foundations of Cryptography Volume 1 - Basic Tools**, Cambridge U. Press, 2001.

[7] O. Goldreich, S. Micali and A. Wigderson, *Proofs that Yield Nothing But their Validity, and a Methodology of Cryptographic Protocol Design*, J. of the ACM 38, 1991, pp. 691–729.

[8] Brian Hayes, *Unwed Numbers*. American Scientist, January-February 2006. http://www.americanscientist.org/template/AssetDetail/assetid/48550

[9] Tal Moran, Moni Naor, *Basing Cryptographic Protocols on Tamper-Evident Seals*, Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP) 2005, Lecture Notes in Computer Science 3580, Springer, pp. 285–297.

[10] Tal Moran, Moni Naor, *Polling With Physical Envelopes: A Rigorous Analysis of a Human Centric Protocol*, submitted.

[11] M. Naor, *Bit Commitment Using Pseudo-Randomness*, Journal of Cryptology, vol 4, 1991, pp. 151–158.

[12] Moni Naor, Yael Naor, and Omer Reingold. *Applied kid cryptography or how to convince your children you are not cheating*, March 1999.
`http://www.wisdom.weizmann.ac.il/~naor/PAPERS/waldo.ps`

[13] Bruce Schneier. The solitaire encryption algorithm, 1999. http://www.schneier.com/solitaire.html.

[14] Salil P. Vadhan, *Interactive Proofs & Zero-Knowledge Proofs*,
`http://www.eecs.harvard.edu/~salil/papers/pcmi-abs.html`

[15] *Sudoku*, Wikipedia, the free encyclopedia, (based on Oct 19th 2005 version), available
`http://en.wikipedia.org/wiki/Sudoku`

[16] Takayuki Yato, *Complexity and Completeness of Finding Another Solution and its Application to Puzzles*, Masters thesis, Univ. of Tokyo, Dept. of Information Science, Jan 2003. Available:
`http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/MasterThesis.ps`