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

Ronen Gradwohl[*] Moni Naor[†] Benny Pinkas[‡] Guy N. Rothblum[§]

August 12, 2007

Abstract

We consider cryptographic and physical zero-knowledge proof schemes for Sudoku, a popular combinatorial puzzle. We discuss methods that allow one party, the prover, to convince another party, the verifier, that the prover has solved a Sudoku puzzle, without revealing the solution to the verifier. The question of interest is how a prover can 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 to the verifier.

In this paper we consider several protocols that achieve these goals. Broadly speaking, the protocols 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. 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 physical protocols utilize items such as scratch-off cards, similar to those used in lotteries, or even just simple playing cards.

The cryptographic protocols are direct and efficient, and do not involve a reduction to other problems. The physical protocols are meant to be understood by "lay-people" and implementable without the use of computers.

[*]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`. Research supported in part by the Israel Science Foundation (grant number 860/06).

[§]CSAIL, MIT, Cambridge, MA 02139, USA; email: `rothblum@csail.mit.edu`. Research supported by NSF grant CNS-0430450 and NSF grant CFF-0635297.

1 Introduction

Sudoku is a combinatorial puzzle that swept the world in 2005 (especially via newspapers, where it appears next to crossword puzzles), following the lead of Japan (see the Wikipedia entry [20] or the American Scientist article [12]). In a Sudoku puzzle the challenge is a 9×9 grid subdivided into nine 3×3 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 one to solve them.

A natural issue, at least for cryptographers, is how to convince someone else that you have solved a Sudoku puzzle without revealing the solution. In other words, the question of interest here is: 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 [7] for an accessible account and [8] 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 items such as playing cards scratch-off cards, similar to those used in lotteries.

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 [21], 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 [9], there exists one for Sudoku, via a reduction to 3-Colorability or some other NP-Complete problem with a known zero-knowledge proof (see definition in Section 2). In this work, however, we are interested in more than the mere existence of such a proof, but rather in 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 requirement is that the participants have an intuitive understanding of the correctness of the proof, and thus are convinced by it, rather than relying blindly "on the computer". (For another example in which this intuitive understanding is important, see the work of Moran and Naor [14] on methods for polling people on sensitive issues.) Physical protocols whose security is intuitively clear are also a great tool for teaching zero-knowledge to non-experts (see [2, 4, 6, 16, 18] for other explorations of simple cryptographic protocols for education and fun).

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 same task.

Organization: In Section 2 we outline the definition of a zero-knowledge protocol, and the properties of the cryptographic and physical protocols. In section 3 we describe two cryptographic zero-knowledge protocols: the first protocol is very simple and direct, and the second is slightly more involved, but has a lower (better) probability of error. In Section 4 we describe several physical protocols using cards. Finally, in Section 5 we discuss further research directions.

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 $k \times k$. Some of the cells are already filled with values in the range $\{1, \ldots, n\}$. The goal is to fill the remaining cells with numbers from the same range so that each number appears exactly once in each row, column and subgrid.

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 proofs and of commitments. For more details, see the above mentioned books by Goldreich [7, 8], or the writeup by Vadhan [19]. In general, a zero-knowledge proof, as defined by Goldwasser, Micali and Rackoff [10], 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 protocol is probabilistic, i.e., the messages that the two parties send to each other are functions of their inputs, the messages sent so far and their private random coins (sequence of random bits that each party is assumed to have in addition to its input). Once the programs of the verifier and prover are fixed, for a given instance the messages sent are a function of the random coins of the prover and verifier only. We will be discussing several properties of such protocols: completeness, soundness, zero-knowledge and proof-of-knowledge.

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; a correct proof is *always* accepted (i.e. with probability 1). The probability is over the random coins of the prover and the verifier. The *soundness error* (or 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.

The goal in designing the protocols is to prevent the verifier from gaining any new knowledge from a *correct* (interactive) proof. I.e., the protocol should be *zero-knowledge* in the following sense: whatever a verifier could learn by interacting with the correct prover, the verifier could learn itself. To formalize this requirement, we require that there is an efficient *simulator* that could have generated the verifier's conversation with the prover without the benefit of the conversation actually occurring, based on knowing the puzzle alone, without knowledge of the solution. Since the protocol is probabilistic, we consider the *distribution* of the conversation, the messages sent back and forth between the prover and verifier. We want the two distributions, the one of a conversation between the real prover and verifier, and the one that the simulator produces, to be indistinguishable. Furthermore, we want a simulator for any possible behavior of the verifier, even a verifier that does not follow the prescribed protocol.

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 another machine, called the *extractor*, that can communicate with the prover and actually come up with the solution itself. This must involve running the prover several times using the same randomness (which is not possible under normal circumstances), so as not to contradict the zero-knowledge properties.

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 latter not learning anything meaningful about the value. Such a protocol consists of two phases. The first is the *commit* phase, following which the sender is bound to some value v, while the receiver cannot determine anything useful about v. In particular, this means that the receiver cannot distinguish between the case $v = b$ and $v = b'$ for all b and b'. This property is called *hiding*. 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. This property is called *binding*. Bit commitments can be based on any one-way function [15] and are fairly efficient to implement. Both the computational complexity and the communication complexity of such protocols are reasonable and in fact one can amortize the work if there are several simultaneous commitments. In this case, the amortized complexity of committing to a bit is $O(1)$.

Note that in this setting we think of the adversary as controlling one of the parties (prover and verifier) and as being malicious in its actions. 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 well established and reasonably standard, when discussing ‘physical’ protocols there are many different options, ranging from a deck of cards [4, 18] to a PEZ dispenser [2], a broadsheet newspaper [16], and more (see [13] for a short survey). In our setting we will be using tamper-evident sealed envelopes, as defined by Moran and Naor [13]. 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). Actually for two of our three physical protocols the tamper evident sealed envelopes can be implemented via standard playing cards. These are ‘sealed’ by turning a card face down, and opened by turning the card over. For a demonstration of a zero-knowledge proof for Sudoku using only playing cards, see the web page [11].

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 uses 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 subset of the cells with predetermined 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, as described in Section 2. The structure of the proof is as follows, which is common to many zero-knowledge protocols:

1. The prover commits to several values. These values are functions of the instance, the solution

and some randomization known only to the prover.

2. The verifier requests that the prover open some of the committed values – this is called the *challenge*. The verifier chooses the challenge at random from a collection of possible challenges.

3. The prover opens the requested values.

4. The verifier checks the consistency of the opened values 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.

To prove that a protocol with the structure above is zero-knowledge we use the so called 'standard' argument, due to [9]: we require that the distribution of the values opened in Step 3 is an efficiently computable function of the Sudoku puzzle and the challenge the verifier sent in Step 2 (but *not* of the puzzle's solution. If the *number* of possible challenges in Step 2 is polynomial in the size of the Sudoku puzzle, then this property, together with the indistinguishably property of the commitment protocol, implies the existence of an efficient simulator, as described below.

The simulator operates in the following way: it picks at random a challenge that the verifier might send in Step 2 (i.e. it guesses what the verifier's challenge will be), and computes commitments for Step 1 that will satisfy this challenge. The simulator simulates sending these commitments to the verifier, then it runs the verifier's algorithm with the puzzle as its input, a fresh set of random bits and these commitments being the first message it receives. It then obtains the challenge the verifier sends in Step 2. If this challenge is indeed the value it guessed, then the simulator can open the commitments it sent and the verifier should accept; the simulator can continue simulating the protocol and output the transcript of the simulated protocol execution. Otherwise, the simulator resets the simulation and starts it all over again.

If the number of possible challenges is polynomial, then each time the simulator "guesses" the verifier's challenge, it is correct with some 'reasonably high' probability (i.e. at least an inverse polynomial). Therefore within a polynomial number of tries the simulator is expected to guess the verifier's challenge correctly and the simulation process succeeds. This procedure guarantees that the protocol is zero knowledge because the output of the simulator looks very much like a successful execution of the proof protocol. I.e., the output of the simulator is indistinguishable from what the verifier would see when interacting with the prover, but is computed without ever talking with the prover!

The two protocols we provide are based on two classic zero-knowledge protocols for NP problems: for 3-Colorability and Graph Hamiltonicity. We find it interesting that while the original protocols seem to fit different types of problems, we could efficiently adapt both of them for the same problem.

3.1 A Protocol Based on Coloring

The following protocol is an adaptation of the famed GMW zero-knowledge proof of 3-Colorability of a graph [9] (see [8]) for Sudoku puzzles. The idea there was for the prover to randomly permute the colors and then commit to the (permuted) 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 one should also check consistency with the partial coloring. The resulting Sudoku 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:

Protocol 1 ***A cryptographic protocol with*** $\mathbf{1 - \frac{1}{3n+1}}$ ***soundness error***
Prover:

1. *Prover chooses a random permutation* $\sigma : \{1, \ldots, n\} \mapsto \{1, \ldots, n\}$*.*
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, the verifier checks 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 original value may be different than the committed one), and that cells with different values are still different, i.e. that* σ *is indeed a permutation over the values in the filled-in cells.*

Proof of the required properties: The perfect *completeness* of the protocol is straightforward. As for *Soundness*, note 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 $1/(3n + 1)$. The fact that the protocol is a *proof-of-knowledge* follows from observing that a prover that convinces the verifier with high probability (greater than $1 - 1/(3n + 1)))$ is able to answer *all* $3n + 1$ queries properly. It is therefore possible to extract the solution by running the protocol and rewinding it multiple times until the prover answers all queries; then find a reverse permutation σ^{-1} mapping the filled-in values to the original ones and use it to deduce the solution (if not all values appear in the filled-in cells, then any permutation σ^{-1} that maps the filled-in cells correctly is sufficient). To verify the *zero-knowledge* property note that the distribution on the values of the answer when the challenge is a row, column or subgrid is simply a random permutation of $\{1, \ldots, n\}$. The distribution in case the challenge is filled-in cells is a random injection of the values appearing in those cells to $\{1, \ldots, n\}$. Therefore it is easy to simulate the prover's answers and the zero-knowledge property of the protocol follows the standard arguments.

The witness/solution size of this protocol, 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 Error

Below is a more efficient zero-knowledge protocol for the solution of a Sudoku puzzle. It is closest in nature to Blum's protocol for proving the existence of a Hamiltonian Cycle [3]. The protocol has constant (2/3) soundness error 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 following properties hold:

a. The cells corresponding to the rows, columns and subgrids have all possible values.

b. The three copies of each cell have the same value.

c. The cells corresponding to the predetermined values indeed contain them.

If all three conditions are met, then, as we show below, there is a solution and the prover knows it. The following protocol implements this idea:

Protocol 2 ***A cryptographic protocol with 2/3 soundness error***
Prover:

1. *Commit to $3n^2$ values $v_1, v_2, \ldots, v_{3n^2}$ 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 $\{1, \ldots, 3n^2\}$, 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 queries 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 and that no two sets intersect.*

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, that every number appears in n triples, and that no two triples intersect.*

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 (i) that the opened committed values are consistent with the predetermined values, (ii) that the set partitions of Item 4 are consistent with the rows, columns and subgrids as defined by the grid locations of the commitments and (iii) that the naming of the triples is consistent with the grid locations of the commitments.*

Proof of the required properties: The perfect *completeness* of the protocol is straightforward. To examine other properties, note that each option for the verifier's query checks a corresponding property from the list of properties that the prover must prove. The first query (query (a)) checks the constraint that all values appear in each row, column and subgrid (item (a) in the list of properties above). The second option for the query (query (b)) makes sure that the value of the cell is consistent in its three appearances (item (c) in the list of properties above). The third option for the query (query (c)) makes sure that the filled-in cells have the correct value (item (c) in the list of properties above), and that the partitioning of the cells to rows, columns and subgrids is legitimate. Therefore, if all three challenges are met, then we have a solution to the given Sudoku puzzle. As a result, a prover which cannot solve the puzzle cannot answer all three queries, and it caught with probability of at least 1/3 (*Soundness* is therefore proved, with soundness error of 2/3). The protocol is a proof-of-knowledge as well, since given the answers to all three possible queries of the verifier it is easy to find the solution to the puzzle. Regarding the *zero-knowledge* property, note that for each challenge it is easy to describe the distribution on the desired response:

a. The answer to the first query is a random permutation of the sequence of the length $3n^2$ where each element in $\{1, \ldots, n\}$ appears $3n$ times, together with a random partitioning of the sequence into non-intersecting sets containing all the values in $\{1, \ldots, n\}$.

b. The answer to the second query is a sequence as above plus a random partition into equi-valued triples.

c. The answer to the third query consists of n^2 triples of *locations* in the range $\{1, \ldots, 3n^2\}$, a random one-to-one mapping of these triples to the n^2 cells of the grid, $3n$ sets containing the locations of the cells of each rows, columns and subgrids, and the list of values of filled-in cells.

Therefore, the zero-knowledge of the protocol follows from standard arguments, as outlined in the beginning of the section.

Overhead of our protocols: The communication complexity and computation time of both protocols presented here is similar (assuming efficient commitments), and is $O(n^2 \log n)$. However, the first protocol allows the prover to cheat (without being caught) with relatively high probability, $(1 - 1/(3n + 1))$, while the second protocol has a constant probability of catching a cheater. In both cases the soundness error 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 [8], to preserve the zero-knowledge property). Therefore, to reduce the cheating probability to ε, 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.

4 Physical Protocols

The protocols described in Section 3 can both 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.

Tamper evidence as a physical cryptographic primitive: 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].

[1]Perhaps quantum cryptography can yield an approximation to such a box, but not a perfect one. See the discussion in [13].

A different physical metaphor was proposed by Moran and Naor [13], who suggested concentrating on the *tamper-evident* properties of sealed envelopes and scratch-off cards. That is, anyone holding the envelopes 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. We require this property from the physical primitives we use. 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 [13]).

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 [13, 14]. 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. The protocol does not prevent cheating by adversaries that accept the risk of being labelled as cheaters (in this respect it is similar to the model of covert adversaries [1]).

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 (see Section 5). The presence of the two parties in the same room is required since the protocols use such operations as shuffling a 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.

We also need two additional functionalities that are not included in the vanilla model of sealed envelopes ([13, 14]): *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 4.2.

Defining zero-knowledge and knowledge extraction. It is easy to apply in the physical setting described above the same definitions of completeness and soundness as in the cryptographic setting. The definition of zero-knowledge in the physical setting 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. 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 it does have an advantage over the prover: at any point in time it is allowed to swap a committed value (e.g., an unscratched card) with another. This advantage replaces the ability of simulators to "rewind" the verifier in cryptographic zero-knowledge protocols. The appropriate analogy is editing a movie, as first suggested in [17]. When making a movie of the proof one can swap the cards and edit the movie so this action is unnoticeable. The result is indistinguishable from what one would see in a real execution. We will describe such simulators in Sections 4.1 and 4.2.

Finally, since we want protocols that are also 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.

Implementing the seals: There are several options for implementing the tamper evident seals required for the physical protocols.

- They can be implemented using sealed envelopes. Namely, the required value is put in a sealed envelope which cannot be distinguished from other envelopes. The value cannot be reveale without opening the envelope. It is obviours, from obseving an envlope, to decide whether it was opened or not.
- The protocol can be based on the use of scratch-off cards, which have the same properties as tamper resistant envelopes.
- Given that the setting we consider involves the prover and receiver being in the same room there is a very simple implementation for the seals without scratch-off cards or envelopes: standard playing cards. Sealing a value means that a card with this value is placed faced down. The equivalent of scratching off or opening the value is simply turning the card over so that it is face up. Tamper evidence is achieved by making sure that no card is turned over before it should be. The prevalence of playing cards and the experience people have in shuffling such cards makes this implementation very attractive. This implementation is relevant for the first two protocols. A demonstration of running the first protocol using only playing cards is documented in the web page [11].

4.1 A Physical Zero-Knowledge Protocol with Constant Soundness Error

In the following protocol, the probability that a cheating prover will be caught is at least 8/9. Each cell should have three (identical) cards. The main idea is that1q 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 (described using scratch-off cards):

Protocol 3 ***A physical protocol with* 1/9 *soundness error***

- *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 off all the cards in each packet and verifies that each packet contains all of the numbers.*

An implementation with playing cards: As mentioned above, this protocol can be implemented using standard playing cards, without any scratch-off layer. In the first step the prover puts all cards face down, except for those cards in filled-in cells, which are put face up. In the following steps the verifier randomly assigns the three cards of every cell to the row/column/subgrid containing this cell, and the prover makes packets and shuffles them, without turning over the cards. Only in the last step do the parties turn the cards over and examine their values.

Completeness: Perfect completeness of the protocol is straightforward.

Soundness: We claim that the soundness error of the protocol is 1/9. We describe a simple argument showing that the soundness error is 1/3 and provide a more involved analysis showing that it is indeed 1/9.

Lemma 1 *The cheating probability of a corrupt verifier in Protocol 3 is at most* 1/3*.*

Proof: Assume that the prover does not know a valid solution for the puzzle. Then he is always caught by the protocol as a liar if he places the cards such that each cell has three cards of identical value. 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.

A more involved argument shows that the soundness error is actually only 1/9.

Lemma 2 *The cheating probability of a corrupt verifier in Protocol 3 is at most* 1/9*.*

The full proof appears in the Appendix. The basic idea is that while we know that there is a cell where not all three values are the same, we also know that 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. The proof considers different ways in which a prover can cheat on these cells, and shows that his success probability is bounded above by 1/9.

Zero-Knowledge: To show that Protocol 3 is zero-knowledge, we have to describe an efficient simulator that interacts with a cheating verifier, 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 it does have an advantage over the prover: before handing the shuffled packets to the verifier, it is allowed to swap the packets for different ones (see the discussion above). The simulator acts as follows:

- 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 proof convinces the verifier with high probability, then the scratched-cards give a solution.

Overhead: Finally, in terms of the complexity of the protocol, we utilize $3n^2$ scratch-off cards, and $3n$ shuffles by the prover. Consider the typical 9×9 case. The total number of cards needed is $3 \cdot 81 = 243$ cards, 27 cards of each type. We want to use standard packs of playing cards, (it is important that they have identical backs). Using only the cards numbered 1 to 9, discarding all other cards, requires 7 packs (if all the cards are used, 5 packs suffice). So the equipment needed to execute the protocol for any puzzle is a large sheet with the 9×9 grid marked on it and several packs of cards. A demonstration of running the protocol in this manner is documented in the web page [11]. However, recall that we are interested in making the protocols accessible to humans. For a standard 9×9 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.

4.1.1 Reducing the Number of Shuffles

We now discuss a variant of the previous protocol, where the number of required shuffles is $c - 1$, at the expense of each shuffle being applied to a larger set of envelopes (expected size $3n^2/c$) and with worse soundness $(1 - \frac{8}{9}\frac{c-1}{c})$. 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. Note that the special case of only one shuffle has soundness error 4/9.

Protocol 4 ***A physical protocol with*** $c - 1$ ***shuffles and*** $\mathbf{1 - \frac{8(c-1)}{9c}}$ ***soundness error***

- *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 into a packet).*
- *The verifier marks each packet with a number chosen uniformly at random from* $0, \dots, c-1$*, where 0 corresponds to leaving the packet unmarked.*
- *For* $i = 1, \dots, c-1$*:*
  - *The prover takes all packets marked with* i*, shuffles them all together, and hands them to the verifier.*
  - *The verifier scratches off all the cards and verifies that in each packet, each number appears the correct number of times (namely, if* t *packets were marked* i*, each number must appear* t *times in the packet corresponding to* i*).*

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 (i.e. by a number i from 1 to $c - 1$), and no other unbalanced packet is marked by i, 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. The exact result is proven in the following lemma.

Lemma 3 *The cheating probability of a corrupt verifier in Protocol 4 is at most* $(1 - \frac{8}{9}\frac{c-1}{c})$*.*

Proof: With probability 8/9, some packet, say a, is unbalanced. Now suppose the verifier has already gone through all other packets and marked them. Thus far, each marked packet is either balanced or unbalanced. If they are all balanced, then with probability $(c-1)/c$ the verifier will mark packet a with one of $1, \ldots, c-1$, and the final mix will be unbalanced. If one marked packet is unbalanced, then with probability $(c-1)/c$ the verifier will **not** mark the packet a with the correct number, and again the final mix will be unbalanced. Finally, if more than one marked packet is unbalanced, then with probability 1 the final mix will be unbalanced. Thus, with probability $(c-1)/c$, 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 $8/9 \cdot (c-1)/c$ as claimed.

The *zero-knowledge* and *proof-of-knowledge* properties can be proved in the same way as they were proved for Protocol 3.

4.2 A Physical Zero-Knowledge Protocol with no Soundness Error

In this section we describe another physical zero-knowledge protocol, this time with the optimal soundness error 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. This functionality generates three identical copies of a card, without revealing its value.

4.2.1 The triplicate functionality

We show here two possible methods of implementing the triplicate functionality:

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 $\{1, \ldots, n\}$ ($3n$ 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.

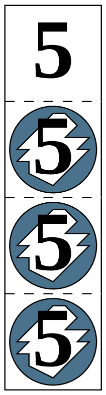

Figure 1: A scratch-off card with the triplicate functionality.

Triplicate without trusted setup: It is preferable to be able to implement the triplicate functionality in the absence of a trusted party preparing the cards in advance. To do so we utilize a property of the human visual system: it can easily distinguish between a uniformly colored patch and one which has more than one color. We will use scratch-off cards as before, but the underlying numbers are replaced by colors, in a straightforward encoding, e.g. '1' is encoded by yellow, '2' by red etc. The idea is that the prover prepares a scratch-off card which is (or at least should be) uniformly colored. The verifier partitions (cuts) the card at random to three parts of equal shape and size. When it is time to peel off the top layer, if the color in one of the parts is not uniform then it is evident the prover was cheating and the verifier will summarily reject. Concretely, let the prover use a circular scratch card. When the prover wishes to triplicate a card, he asks the verifier to cut the card into three equally shaped parts (if it is easier to perform, he could ask the verifier to partition into four parts, one of which will be thrown away or shuffled and checked separately). The point is that the partitioning should be *random*.

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 (the exact value depends on assumptions on resolution and on the model random partition).

Using the tamper-evident seals with the additional *shuffle* and *triplicate* functionalities, the protocol is as follows:

Protocol 5 ***A physical protocol with 0 soundness error, using triplicate***

- *The prover lays out the seals corresponding to the solution in the appropriate place. The seals placed on the filled-in squares are scratched off; they and must be the correct value (otherwise the verifier rejects).*
- *The verifier 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 $\{1, \ldots, n\}$ appear.*

The *completeness* of the protocol is clear. As for soundness, note that the *triplicate* functionality solves the problem of the first physical protocol (Protocol 3), by preventing the prover from assigning different values to the same cell. Therefore the prover has no way of cheating. Thus, the soundness error of the protocol is 0 (assuming that the triplicate functionality is perfect, i.e., that the prover can never generate different copies of the same card).

The simulator for this protocol is nearly identical to that of Protocol 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. The protocol will therefore be zero-knowledge and be a proof-of-knowledge.

4.3 A Protocol Using Scissors

The physical protocols we have described so far are based on the use of cards or other methods of sealing information. We now describe a protocol (and several variants) based on the prover writing down the solution on a sheet of paper and then cutting it to pieces according to the verifier's orders. We first describe the basic protocol, then explain why a simpler variant is insecure, and then describe and analyze some variants of the basic protocol.

Protocol 6 ***A physical protocol using scissors***

a. *The prover takes a sheet of paper on which the puzzle is printed. He then writes down, for every cell with a filled-in value, this filled-in value on back side of the cell (namely, on the back of the page, right behind the printed filed-in value). The result is that filled-in cells, and only them, have their values written on both sides of the page.*

b. *The prover writes down the solution to the puzzle on the (original) printed puzzle, and keeps this side of the page hidden from the verifier.*

c. *The verifier verifies that the prover wrote the right values on the back of the puzzle.*

d. *If the previous check is fine, the verifier chooses one option out of rows/columns/subgrids.*

e. *Suppose that the verifier chooses "rows". The prover then cuts the puzzle and separates it into n rows. (If the choice is "columns" the prover separates the columns from each other, and similarly for subgrids. In the rest of the protocol description we then replace the word "row" by "column", or "subgrid", according to the verifier's choice above.) The prover then cuts each row to separate it into n cells. He shuffles the cells of each row (separately from the cells of other rows) and then hands them to the verifier.*

f. *The verifier checks (1) that each row contains all n values, (2) that in each row the cells whose value is written on both sides agree with the filled-in values of that row in the puzzle, and (3) that these cells have the same value written on both their sides.*

Given this protocol, it is unclear at first why the prover is required to write the values of the filled-in cells on both sides of the page. A simpler protocol could use a standard puzzle (with no values written on the back) and continue as Protocol 6 (without requiring the verifier to verify the values written on the back of the cells). The problem with this simple protocol is that it does not detect whether the prover solves a different puzzle in which every row/cells/subgrid has the same set of filled-in values as the original one, but in different locations.[2]

Before analyzing the protocol, let use describe two simple variants which have the same properties:

- In the first variant, the original puzzle already has the filled-in values printed on both sides of the page.

[2]It is interesting to find out whether the following statement is correct: "If for two Sudoku puzzles it holds that the set of filled-in values of every row/column/subgrid of the first puzzle is equal to the set of filled-in values of this row/column/subgrid in the second puzzle, then given a solution to the first puzzle it is easy to solve the second puzzle". If this statement is correct then the simple protocol described here can replace Protocol 6.

- In the second variant, the prover does not write values on the back of the puzzle. However, after cutting a row into cells he first lets the verifier open the locations of filled-in values and verify that they are correct. Only then does the prover shuffle the remaining cells of the row. The verifier then verifies that the shuffled values contain the values that should appear in all the blank cells.

Analysis. The completeness of the protocol is clear. As for soundness, note that a prover which does not know a solution to the puzzle cannot write a solution which satisfies all three possible queries of the verifier. This prover is therefore caught with probability of at least $1/3$, and consequently the soundness error is at most $2/3$. The zero-knowledge property is shown by a simulator which interacts with the verifier, and replaces the values of the pieces of every row with a random permutation of the values $[1, n]$. Knowledge extraction is demonstrated by an extractor which interacts with the prover and simply turns the page over to examine the solution.

4.3.1 Reducing the soundness error

Using a trusted copier to achieve 0 soundness error. Suppose that the two parties have access to a trusted copier. Say, a copy machine which can be used to make three copies of the original solution in a way with ensures the verifier that the three copies are identical, and ensures the prover that the verifier does not see the copied solution. In this case, the verifier can ask the prover to apply the protocol to the rows of the first copy, the columns of the second copy, and the subgrids of the third copy. Any incorrect solution is identified with probability 1, and the soundness error is 0.

Repetition. Repeating the protocol k independent times obviously reduces the soundness error to $(2/3)^k$. This process might be tedious as the parties need to separately cut each of the k puzzles. Another option is to use k copies of the puzzle printed on sheets of paper of different colors. The prover writes the solution on each one of the copies and then the verifier divides the copies into three random packets. He asks the prover to apply the protocol to the rows of the first packet, the columns of the second packet, and the subgrids of the third packet. The prover can cut in parallel, and shuffle in parallel, all copies in the first packet; and similarly for the second and third packets. The verifier verifies that in every row the pieces of each color form a permutation of the numbers $[1, n]$. The work of the prover (in terms of the number of cuts he has to make) is therefore roughly three times the work in the basic solution, while the soundness error is reduced to $(2/3)^k$. (Note that a similar gain in efficiency can be achieved even if all copies are of the same color, if the prover shuffles the cells in a way which does not mix the values of a cell of one puzzle with the values of cells of different puzzles. Namely, if a packet contains k' copies, then shuffling is done in a careful way which ensures that the k' copies of the value of each cell do not mix with values of a different cell, and do not change their order.)

Repetition without requiring additional properties. A protocol which uses repetition without requiring additional properties, such as sheets colored in different colors, could work in the following way: The prover writes down k copies of the solution; the verifier divides them into three random packets; the prover then cuts *in parallel* the rows and then the cells of the first packet (namely, applies a single cut to separate the first row of each of the puzzles from the other rows, etc.). The prover then shuffles all cells of the first row of the first packet, and the verifier checks that each value appears among these cells the same number of times. They then apply this check to the other rows and the packets corresponding to columns and to subgrids. This protocol is efficient, but it does not check each copy of the solution independently. It therefore does not identify cheating if the cheating prover is lucky and the incorrect solutions whose rows are chosen to be examined are such that their row values cancel each other (namely, the *total* number of cells of a specific value is the same for all values, even though some rows are unbalanced).

We show here that the soundness error of this protocol can be $\Omega(1/\sqrt{k})$, which is quite high compared to the exponentially small error of the previous solutions. Suppose that a cheating prover does not know how to solve a puzzle, but he does know a solution which is correct, except for having two '1' values in the first row and two '2' values in the second row (all other rows, as well as columns and subgrids are correct). He also knows a solution which has two '2' values in the first row and two '1' values in the second row. This prover is only caught if the number of solutions of the first type which is assigned to the "rows" packet is equal to the number of solutions of the second type assigned to the packet. Since the total number of solutions assigned to this packet is $\Omega(k)$ (with all but negligible probability), the probability of these two numbers being equal is $\Omega(1/\sqrt{k})$.

5 Conclusions and Open Problems

We describe the main properties of the different physical protocols in Table 1. Note that Protocol 4 provides a nice tradeoff between the number of shuffles and the soundness error (e.g., setting $c = d$ results in 4 shuffles and a soundness error of 1/3, compared to $3n$ shuffles required in order to achieve soundness error of 1/9 in Protocol 3.

	# of cards	shuffles	soundness error
Protocol 3	$3n^2$	$3n$	$1/9$
Protocol 4	$3n^2$	$c-1$	$1/9 + \frac{8}{9c}$
Protocol 5	n^2 special cards	$3n$	0
Protocol 6	no cards	n	2/3

Table 1: Comparing the different physical protocols

There are many remaining open questions: Is there an implementable physical protocol that can be executed by (snail) mail, i.e. without assuming that the prover and the verifier are in the same room? In principle we know that such protocols exist, based on the scratch-off functionality, since in [13] it was shown how to construct commitments from this functionality and hence the cryptographic protocols of Section 3 can be used. However, since there is an amplification step in the construction of commitments from the tamper-evident envelopes of [13], involving a large number of repetitions, the result is not really human implementable. On the same note, Tom Berson asked whether it is possible to construct non-interactive physical protocols proving that a puzzle is solvable. Such protocols could be used by puzzle designers to convince potential solvers that a solution to a puzzle exists.

One of the major applications of zero-knowledge proofs in the cryptographic setting is as a mechanism for converting a protocol that is resilient to semi-honest behavior of the participants into one that is resilient to *any* malicious behavior. This conversion is not necessarily always possible with physical protocols. It would be interesting to see whether it is possible to do so for the Sudoku protocols.

Acknowledgments. We are grateful to Tal Moran for helpful discussions and comments. We thank Tobias Barthel and Yoni Halpern for providing the initial motivation for this work. We thank Efrat Naor for helping to implement the protocol with a deck of playing cards and Yael Naor for diligently reading the paper.

References

[1] Yonatan Aumann and Yehuda Lindell, *Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries*, In TCC 2007, Springer-Verlag (LNCS 4392), pp. 137-156, 2007.

[2] 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).

[3] Manuel 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.

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

[5] Ron Fagin, Moni Naor and Peter Winkler, *Comparing Information Without Leaking It*, Comm. of the ACM, vol 39, May 1996, pp. 77–85.

[6] M.R. Fellows and N. Koblitz, *Kid Crypto*, Advances in Cryptology - Crypto '92, Lecture Notes in Computer Science 740, Springer-Verlag, pp. 371–389, 1992.

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

[8] Oded Goldreich, **Foundations of Cryptography: Basic Tools**, Cambridge U. Press, 2001.

[9] Oded Goldreich, Silvio Micali and Avi Wigderson, *Proofs that Yield Nothing But their Validity, and a Methodology of Cryptographic Protocol Design*, J. of the ACM 38, 1991, pp. 691–729.

[10] Shafi Goldwasser, Silvio Micali and Charles Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Computing Vol. 18, no. 1, 1989, pp. 186–208.

[11] Ronen Gradwohl, Efrat Naor, Moni Naor, Benny Pinkas and Guy N. Rothblum, *Proving Sudoku in Zero-Knowledge with a Deck of Cards*, January 2007.
`http://www.wisdom.weizmann.ac.il/~naor/PAPERS/SUDOKU_DEMO/`

[12] Brian Hayes, *Unwed Numbers.* American Scientist Vol. 94, no. 1, January-February 2006. http://www.americanscientist.org/template/AssetDetail/assetid/48550

[13] 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.

[14] Tal Moran, Moni Naor, *Polling With Physical Envelopes: A Rigorous Analysis of a Human Centric Protocol*, Advances in Cryptology - EUROCRYPT 2006, Lecture Notes in Computer Science 4004, Springer, 2006, pp. 88–108.

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

[16] 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`

[17] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou and Tom Berson, *How to explain zero-knowledge protocols to your children*, Advances in Cryptology - CRYPTO'89, Lecture Notes in Computer 435, Springer, 1990, pp. 628–631.

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

[19] Salil P. Vadhan, *Interactive Proofs & Zero-Knowledge Proofs*, lectures for the IAS/Park City Math Institute Graduate Summer School on Computational Complexity. `http://www.eecs.harvard.edu/~salil/papers/pcmi-abs.html`

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

[21] 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`

A Improved Analysis for Soundness of Protocol 3

We now provide a more involved argument that shows that the soundness error 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) In 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.

(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. 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 y for cell a's subgrid and x for cell b's column with probability 1/9, since each assignment is made independently with probability 1/3.
  – 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.
  – Any other situation results in the prover losing with probability 1, as there is no way to select the cards to satisfy all constraints.

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 8/9.

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 error is 1/9.