Bachelor's Thesis Computing Science

Radboud University Nijmegen

Constructing Zero-Knowledge Proof Systems on Sudoku Variants

Author:
Mika Sipman
s1054189

First supervisor/assessor:
Dr. Simona Samardjiska

Second assessor:
Dr. Abraham Westerbaan

August 18, 2024

Acknowledgements

Special thanks to professor Simona Samardjiska for being my supervisor and first assessor. Her time and expertise have been of great value regarding the general process of writing this work.

Special thanks to Lars Ran for being my (bi)weekly supervisor. His availability, interest and expertise not only gave me deeper insight and meaning to the theory revolving around this thesis, but his suggestions regarding on how to use my proof (3.2.4) to improve *protocol 1* (leading to *protocol 2*) and some specific insights on mathematical/logical correlations in *protocol 3* also have been particularly helpful.

Special thanks to professor Abraham Westerbaan, my second assessor, for his time and expertise.

Abstract

Zero-knowledge proof systems, where one party (the prover) can convince another party (the verifier) of a statement revealing no knowledge beyond the validity of that statement, are a vital component in today's' cryptography. However, these systems are often hard to get a mental grasp on. Three Σ-protocols, a specific type of interactive zero-knowledge proof system, will be constructed on two variants of the well known Sudoku puzzle. With it, an intuitive path will be set to step-by-step get a feel for these type of proof systems, their potential and limitations in construction, and their respective properties.

Contents

Introduction

1.1 An Example; Where's Waldo?

Where's Waldo is a series of puzzle books. The objective is simple; There is a large crowded picture containing a person named Waldo (whose appearance is known), which needs to be found. This is already somewhat challenging on a piece of A4-paper, even more so on a larger sheet, maintaining the same crowdedness per unit square.

Now, imagine a 'Where's Waldo'-puzzle is produced of a size infeasible to solve even when given a nonsensible amount of time. Perhaps, whilst the puzzle is not solved, it will be thought that it doesn't contain Waldo at all. After all, it would be an easy way out for the creator of the puzzle to not include Waldo, falsely securing eternal fame for the hardest 'Where's Waldo'-puzzle ever created. However, the creator honestly did include Waldo in their puzzle. Now, how can it be shown that Waldo is in the puzzle, without actually revealing its location?

Let's put the 'Where's Waldo'-puzzle on a really large table. Over the puzzle, a paper, whose length and width exceed at least twice the length and width of the puzzle, is thrown by a person who has knowledge of Waldo's location. This paper, in the middle, has a little cutout the size of Waldo. Now, the puzzle underneath the big sheet can be shifted such that the cutout reveals Waldo's face, proving knowledge of Waldo's location. However, as the puzzle can lie anywhere underneath the larger sheet, nobody knows where Waldo is.

1.2 Motivation

1.2.1 Zero-Knowledge Proofs, Systems and their Importance

Zero-knowledge proofs (hereafter ZKP) were first discovered in 1985 by Shafi Goldwasser, Silvio Micali, and Charles Rackoff in their paper *"The Knowledge Complexity of Interactive Proof-Systems"*[16]. Such a proof takes place between two parties, where one party, the prover, tries to convince the other party, the verifier, of the validity of a certain statement, revealing no additional knowledge than the validity of that statement. There are three core

requirements a proof must have to achieve this;
Completeness. For a true statement, a prover will always convince a verifier.
Soundness. For a false statement, a prover will never be able to convince a verifier (even if the prover cheats and deviates from the protocol).
Zero-Knowledge. The verifier will not learn anything apart from the fact that the statement is true (even if the verifier is malicious).

The statements we are concerned with are of the form $u \in L_R$, where L_R is a NP-language defined by a polynomial time decidable binary relation R. For $(u, w) \in R$, u is the statement and w is a witness for $u \in L_R$. The prover knows w, and wants to convince the verifier that $u \in L_R$ without revealing anything else. In particular, the prover does not want to reveal the witness w. Definition inspired by Bootle et al. [6]. In the "Where's Waldo" scenario (1.1), the task of finding waldo resembles a NP-language, where u is the statement "I know where Waldo is" and the witness w is Waldo's location.

Since the discovery that every NP-language with a proof has a ZKP [15], the vastness of the realm concerning possible cryptographic applications of zero-knowledge proof systems (hereafter ZKPS) began to be explored. Next to cryptographic surface-level applications such as Schnorr's signature scheme [25], ZKPs' general concept and properties can be applied in multiple sensitive matters. One particular example is verification of nuclear warheads for arms control [13], where all parties want to authenticate each other's arsenal, without revealing classified information. Another area where ZKPS's play a big role is in the contemplation of introducing E-voting [18, 9], where theoretical protocols have already been developed [23, 2]. Some other cases where ZKPS's can be effective are in proving of sufficient money in a bank balance, without revealing the exact amount and identifying as part of a group, without revealing your identity. Lastly, the mere fact that it is possible for a party to convince another party of something without having to build prior trust, needed to mitigate abuse of additional knowledge, is a phenomenon applicable in many scenarios.

1.2.2 This work

For many, their first encounter with a ZKP, where a prover can convince a verifier of the validity of a statement without revealing any knowledge about it, feels paradoxical and even impossible. To gradually bridge the gap between story-like explanations of ZKPS's such as the Where's Waldo example in (1.1) or other well-known intuitive stories like the strange cave of Ali Baba [24] and mathematical systems, it is attempted to give three ZKPS's with two distinct approaches in their design. Specifically, for the first (easiest) protocol there will be taken the effort to explain it in fine detail such that a complete mathematical conceptualisation of a ZKPS can be formed. After, an improvement on the first protocol will be discussed. Lastly, the third, and most advanced, ZKPS will be discussed, portraying a different technique and its respective/potential result. This last protocol won't be defined as mathematically in-depth as the first two protocols due to its size and complexity. Rather, the focus will lie more on logical intuition and exploring how to circumvent giving away knowledge.

To maintain a theme, make it a little more fun and remain close to the paper "Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles" [17], which this work is most inspired by, all discussed ZKPS's will revolve around variants of the well-known Sudoku game. Sudoku's have already been playing a large part of interest in the realm of cryptography for its interesting mathematical properties and their respective potential applications. Usage of the structure of Sudoku varies from ZKPS [17] to fully implemented symmetric block ciphers [10]. In this context, a Sudoku 'variant' is defined as a puzzle in the NP-language with a grid, a finite set of placeable elements, and a rule set of criteria a valid solution should adhere to. To reflect on this, a statement u is the prover which says that they have a valid solution for a Sudoku variant and witness w is the actual solution.

Since its discovery by Blum et al. in 1988 [5], a lot of attention in this field has been going toward non-interactive zero-knowledge proofs [27] (hereafter NIZKP), where instead of the prover and verifier having multiple back and forths, the entire proof is completed in one send from the prover to the verifier. This property has the obvious benefit of mitigating potential unwanted interaction in scenarios like online transactions. It is for this reason that NIZKP are the central concept in research of decentralized systems like blockchain [11]. The concept of NIZKP's has, however, been deliberately left out of scope as instead focus will lie on the specific 3-way interactive ZKPS's named Σ-protocols (or just 'sigma protocols') [8]. These Σ-protocols will be easier to fully portray and more intuitive to grasp, with the added benefit of a lot of examples and research already having been conducted in this area [6]. Additionally, relevance remains as it has been proven that each ZKP can be transformed into a NIZKP [12]. There even has been found a Fiat-Shamir type transform for any Σ-protocols to a NIZKP [21].

In this thesis, each section discussing a protocol will have the general form of first defining the ruleset of the sudoku variant, then constructing the Σ-protocol, after which required properties will be shown, proving that the constructed protocol indeed is complete, sound and zero-knowledge. Additionally, relevant roofs and/or remarks can be found in between the aforementioned structure.

Preliminaries

2.1 Notations

2.1.1 Some Notations

$a \overset{?}{=} b$; checking for equality of a and b.

$a \overset{?}{\neq} b$; checking for inequality of a and b.

$a \leftarrow b$; assigning b as/to a.

$a \overset{\$}{\leftarrow} B$; define an arbitrary random picking of B as a.

$a \| b$; concatenation of b to a.

$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\} = \{(A, B)\}$; Cartesian product.

$\mathbb{N}_k = \{1, 2, ..., k\}$, $\mathbb{N}_k^0 = \{0, 1, 2, ..., k\}$

2.1.2 Protocol Visualisation

To better visualise flows of protocols, figures will be produced in order to capture the general structure and potentially also the finer details. These figures will have the general format of figure 2.1.

Disregarding the optional column r, used for allocating numbers to rows which concretise action order on the vertical dimension, there are three columns. Both *column* 1 and *column* 3 denote two entities and the respective actions they make in row 1 to n. In our use-case, entity 1 will be the prover $\mathcal{P}$ and entity 2 be the verifier $\mathcal{V}$. In the row where $n = 0$, known information of each respective entity before running the protocol is displayed. *Column* 2 denotes the sending and receiving of information. Here, an incoming arrow to an entity denotes the reception of information and an outgoing arrow denotes sending information. The specific information received and send is that displayed on top of the arrow.

r	*column* 1	*column* 2	*column* 3

	Entity 1 $(\mathcal{P})$		**Entity 2** $(\mathcal{V})$
0	known information		known information
1	does computation a		
2		$\xrightarrow{a}$	
3			does computation b
4		$\xleftarrow{b}$	
.		.	
.		.	
n		.	

Figure 2.1: Protocol Visualisation

2.2 Cryptographic Primitives

2.2.1 Permutation function

A bijective function from a set to itself: for an ordered set $S := \{s_1, s_2, ..., s_k\}$, a permutation function is a function $f : S \mapsto f(S)$ such that $s_1 \mapsto s_x$, where $x \in \mathbb{N}_k$, $s_2 \mapsto s_y$, where $y \in \mathbb{N}_k \backslash \{x\}$, and so on.

2.2.2 One-Way Function, Hashes and Salts

A one-way function $f : \{0,1\}^* \to \{0,1\}^*$ is defined as a function which is polynomial to compute, however any polynomial-time randomized algorithm F has a negligible chance of succeeding in finding f^{-1}.

A specific type of one-way function, widely-used in today's cryptography, are hash functions. Hash functions will be (shortly) used to demonstrate the commitment scheme, an essential cryptographic primitive whose explanation comes later.

A common problem which arises when using hash functions is the (ab)use of dictionary attacks[22], where one looks for matching hashes of commonly used passwords against leaked login credential, and then on other services try the passwords of the corresponding matched hashes with the attached username. To cryptographically circumvent unwanted determinism in hashing, some randomness is concatenated to the value before hashing, creating different hashes for identical values. This randomness is called a salt. For better visualisation, let's say salt s and s' are two salts generated from an arbitrary uniform random number generator, then for hash function h, we have $h(v\|s) \neq h(v\|s')$, despite identical value v.

2.2.3 Commitment Scheme

Its concept arguably being first discovered by Blum [4], a commitment scheme is a cryptographic primitive, where an entity can 'commit' to an encrypted message which, at a later time, can be opened such that the message cannot be mutated.

> $a = \mathit{Commit}(m, r)$; given a message m and randomness r, compute value a such that it is infeasible to compute message m' and randomness r' which satisfies $\mathit{Commit}(m, r) = \mathit{Commit}(m', r')$.
>
> $o = \mathit{Open}(a, m, r)$; given a commit a, message m and randomness r, the algorithm verifies whether m and r correspond with a.

It should be computationally infeasible to produce an a such that we have $\mathit{Open}(a, m, r) = \mathit{Open}(a, m', r')$, where $m \neq m'$. This property is called *binding*. Also, a cannot give away any information of m and r, which is called *hiding*. Given that r is sampled uniformly, it gives that a is statistically dependent on m.

There are numerous different types of commitment scheme such as polynomial [19], vector [7], functional [20] and more, all with their own benefits. Even though this is an interesting field in and of itself, for this thesis, it is out of scope. Rather, as the aim is to intuitively demonstrate (the concept of) Σ-protocols, it is essential to be minimal and concise yet accurate in explanation in areas which aren't the main focus. Therefore, the commitment scheme used (in protocol 1 and 2) will be one constructed of a one-way-function, specifically, hashes and salts. In the last protocol, we will only mention the notion of commitment as a cryptographic primitive, disregarding in-depth functionality during construction.

To further elaborate by a minimal example; Consider current industry standard hash function SHA-3 [3], which will have function abbreviation $h()$ and a salt s, where $1 \leq s \leq 2^{16}$, which is being chosen by an arbitrary uniform random number generator. Now, a commitment a of message m can be produced by $a = h(m||s)$, which has both implied hiding and binding properties from the properties of a hash function.

2.3 Proof of Knowledge

Complementary Definitions and Notations: Let u be a statement of the NP-language L_R, W the set of all possible witnesses for L_R, and $W(u)$ the set of witnesses for u that should be accepted by the proof. Now, relation R can be described as the set of tuples (u, w), where $u \in L_R$ and $w \in W(u)$.

Proof of Knowledge: For relation R, a proof of knowledge is any protocol between a prover P and a verifier V such that the following two properties hold:

1. **Completeness.** For any common statement u, if $\mathcal{P}$ knows a $W(u)$ and $\mathcal{P}$ and $\mathcal{V}$ follow the protocol, then $\mathcal{V}$ will always accept.

2. **Validity.** We define an efficient algorithm called the knowledge extractor E which has access to converse with $\mathcal{P}$ and can extract the witness $w \in W(u)$ for statement u from $\mathcal{P}$ with non-negligible probability.

2.4 Interactive Zero-Knowledge Proof Systems

Complementary Definitions and Notations: Let $\{0,1\}^n$ denote the set of n-bit strings and let $\{0,1\}^*$ denote the set of all strings. Also, let A and B be two interactive turing machines, then (A, B) denotes a linked pair of the two individual interactive turing machines and $\langle A, B\rangle(u)$ denotes the output of B when interacting with machine A on common input u. A linked pair of interactive turing machine, in this sense, may sound unintuitive. The basic idea, however, is that two interactive turing machines communicating represent a minimal deterministic interaction with unlimited potential larger configurations. In other words, it concretizes any interaction model. For more detail, see chapter 4.2 of [14].

Two probability ensembles are said to be computationally indistinguishable (denoted by $\approx_c$), if no probabilistic polynomial time Turing machine can distinguish them with non-negligible probability.

Interactive Zero-Knowledge Proof Systems: For language $L \subseteq \{0,1\}^*$ and a pair of interactive Turing machines $(\mathcal{P}, \mathcal{V})$, in which $\mathcal{P}$, the prover, possesses unlimited computational power and $\mathcal{V}$, the verifier, is probabilistic polynomial time, $(\mathcal{P}, \mathcal{V})$ is said to be a zero-knowledge interactive proof system of language L if the following three conditions are true.

1. **Completeness.** For any common input $u \in L$ and polynomial $p(\cdot)$,

$$Pr[\langle \mathcal{P}, \mathcal{V}\rangle(u) = 1] \geq 1 - \tfrac{1}{p(|u|)}$$

2. **Soundness.** For any common input $u \notin L$ and any interactive Turing machine $\mathcal{P}'$ and polynomial $p(\cdot)$,

$$Pr[\langle \mathcal{P}', \mathcal{V}\rangle(u) = 1] < 1 - \tfrac{1}{p(|u|)}$$

3. **Zero-Knowledge.** For each probabilistic polynomial time Turing machine $\mathcal{V}'$, there is a probabilistic polynomial time algorithm M^* such that, for any $u \in L$,

$$\langle \mathcal{P}, \mathcal{V}'\rangle(u) \approx_c M^*(u)$$

Machine M^* is called a *simulator* for the interaction of $\mathcal{V}'$ and $\mathcal{P}$. This simulator, as the name suggests, is able to (perfectly) simulate the interaction between $\mathcal{P}$ and $\mathcal{V}'$, without having access to the interactive machine $\mathcal{P}$, proving the property of zero-knowledge. To further elaborate on intuition; The simulator concept embodies the idea that anybody, having no knowledge of the proof, could generate the interaction taken place by $\mathcal{P}$ and $\mathcal{V}'$. If this generation is possible, then, by implication, no knowledge was required for this proof, making the proof zero-knowledge by definition.

To clarify the three properties in more instinctive terms; completeness reflects correctness of the system, so for a true statement ($u \in L$), an honest prover can always complete the proof successfully such that the verifier accepts. Honest, in this context, refers to an entity following the protocol as intended. Soundness is defined against the malicious prover, which means, for $u \notin L$, no prover $\mathcal{P}'$ can construct a valid proof system such that the verifier accepts. So, a prover can never convince a verifier of an incorrect statement. While for the verifier, zero-knowledge means no (malicious) verifier $\mathcal{V}'$ is able to derive extra knowledge from the process of interaction apart from learning that the statement is true.

It should be noted that, whilst completeness is often *perfect* in designed ZKPS's, there are different 'closeness levels' of soundness and zero-knowledge according to different computational capabilities of the prover and verifier. These definitions can be modified accordingly. If the indistinguishability of the two probability ensembles in the zero-knowledge (ZK) property is statistically indistinguishable or identically distributed, ZK will be correspondingly defined as statistical ZK and perfect ZK. Perfect ZK is, to our knowledge, too strict of a definition to go beyond trivial cases (Chapter 4.3.1 in "Foundations of Cryptography: Volume I" [14]). Almost-perfect (statistical) ZK is a relaxation of perfect ZK. The definition given above, called computational ZK, is an even more drastic relaxation of perfect ZK than statistical ZK. However, it still suffices for all practical purposes. There also exists the highly non-trivial variant called honest-verifier zero-knowledge (HVZK), defined as being ZK given that the verifier behaves honestly according to the protocol. Whilst not as secure as the former notions of ZK, the modification in definition of HVZK often reduces great stress on the strictness of the protocol, sometimes being able to gain more efficiency. What's more, HVZK is often sufficient for many cryptographic applications. In the case of the second property, if soundness holds for any probabilistic polynomial time prover, that is, computational soundness, then the interactive proof system is called the zero-knowledge argument system.

Note: Credits to the concise yet clear definitions on zero-knowledge proof systems by Wu and Wang [27], which this definition uses as its foundation. For a more in-depth understanding, see the elaborated definitions in Goldreich's book "Foundations of Cryptography: Volume I" [14].

2.5 Σ - Protocols

Complementary Definitions and Notations: Let there be a polynomial time decidable binary relation $R = \{(u, w)\} \subseteq U \times W$, where $u \in U$ is the statement, common to both prover and verifier, and $w \in W$ is a witness, known only to the prover. Now, let L_R be the language of statements u for which there exists a witness w such that $(u, w) \in R$.

As this thesis revolves around using Σ-protocols for the purpose of puzzles which are in NP (non-trivial to solve for a computer), we say L_R is a NP-language and, therefore, relation R is a NP-relation. This implies that the problem of determining whether $(u, w) \in R$ holds for a specified u and w can be solved in polynomial time.

Σ - Protocols: Σ-Protocols [8] are a specific type of interactive zero-knowledge proof system, having a 3-way pattern consisting of the prover $\mathcal{P}$ sending a commitment a, the verifier $\mathcal{V}$ replying with challenge c picked from a uniformly random challenge set C, and the prover responding with response r, giving final *transcript* (a, c, r). See figure 2.2. As the full protocol should be polynomially computable to maintain reasonable efficiency, generating commitment a, response r and determining boolean value resulted from *Open*() are all computed in polynomial time, also $r^{\$}$ denotes an arbitrary string of random bits, resembling randomness.

A Σ-protocol, largely similar to IZKPS's, need to satisfy the following three properties:

1. **Completeness.** For any common true statement, if $\mathcal{P}$ and $\mathcal{V}$ follow the protocol, then $\mathcal{V}$ will always accept.

2. **Special Soundness.** There exists a p.p.t. (probablistic polynomial time) algorithm E (*extractor*) such that any statement u and pair of accepted conversations (a, c, r) and (a, c', r'), where $c \neq c'$, always computes a witness w such that $(u, w) \in R$. *k-Special Soundness*, defined as being able to extract said w with $(a, c_1, r_1), (a, c_2, r_2), ..., (a, c_k, r_k)$ is equal to *Special Soundness* for $k = 2$ [1].

3. **Special Honest-Verifier Zero-Knowledge.** There exists a p.p.t. algorithm M (*simulator*) which on any statement $u \in L_R$ and given challenge c produces conversations (a, c, r) with the same probability distribution as conversations between honest $\mathcal{P}$ and $\mathcal{V}$ on common statement u and challenge c, where $\mathcal{P}$ uses any witness w satisfying $(u, w) \in R$. Furthermore, given any $u \in U \backslash L_R$, simulator M is just required to produce arbitrary accepting conversations (a, c, r), for any given challenge $c \in C$.

A Σ-protocol is required to be *special* honest-verifier zero-knowledge (SHVZK). SHVZK is a particular case of HVZK such that if a protocol is SHVZK it implies HVZK. HVZK is typically proven as follows for a Σ-protocol: Show that there exists a simulator that can take arbitrary fixed challenge c and show it is possible to efficiently generate transcript (a, c', r), where $c = c'$.

In Σ-protocols, it is not only proved that $u \in L_R$, but also that $\mathcal{P}$ knows a w such that $(u, w) \in R$, in other terms; Σ-protocols are *proofs of knowledge*. Gaining knowledge that $\mathcal{P}$ knows a w is formalized in the special soundness property by means of the extractor. This implies that $\mathcal{P}'$, which has no w such that $(u, w) \in R$, must lie in at least one challenge, meaning that $\mathcal{P}'$ can succeed at most $\frac{1}{|C|}$ times. As a Σ-protocol whose challenge set is singleton is *trivial*, the maximum soundness a Σ-protocol can achieve is $\frac{1}{2}$. This means that, theoretically, $\mathcal{P}'$ has a chance for $\mathcal{V}$ to accept a conversation of a statement u without knowing a w such that $(u, w) \in R$. This is why Σ-protocols get repeated between the prover and the verifier with same statement u, but different commitments and challenges. Now, with n repeats, a protocol with arbitrary soundness $\frac{1}{x}$, where $x \in \mathbb{R}$ and $x \geq 2$, a cheating prover can convince a verifier with probability $(1 - \frac{1}{x})^n$, which approaches 0 for a sufficiently large n.

Some intuitive elaboration on the seeming contradiction of Σ-protocols being proofs of knowledge whilst at the same time zero-knowledge is in order. The idea is that, whilst in a Σ-protocol $\mathcal{P}$ indeed reveals information (note; different from knowledge) to $\mathcal{V}$ in the form of the response r of commitment a, $\mathcal{P}$ gives this information in a way which by itself gives no implication on the content of witness w, just an indication that it exists (getting more convincing by doing more iterations of the protocol with same statement u).

A minimal example of a well-known Σ-protocols operating in this probabilistic manner to gain certainty, is Schnorr's protocol [25].

Note: Credits to the clear definitions on Σ-protocols by Schoenmakers [26], which this definition uses as its foundation.

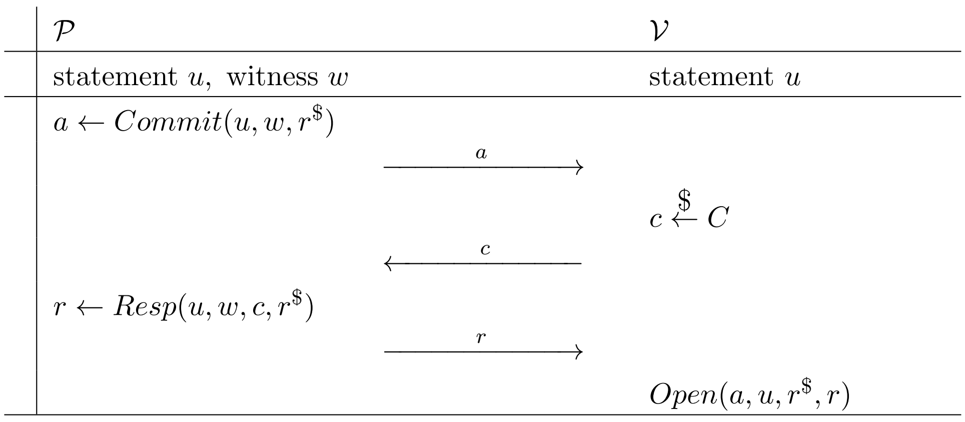

$\mathcal{P}$
$\mathcal{V}$
statement u, witness w
statement u
$a \leftarrow Commit(u, w, r^{\$})$
a
$c \overset{\$}{\leftarrow} C$
c
$r \leftarrow Resp(u, w, c, r^{\$})$
r
$Open(a, u, r^{\$}, r)$

Figure 2.2: Sigma Protocol

Research

3.1 Protocol 1 - Jigsaw Sudoku

3.1.1 Rules and Definitions

Jigsaw Sudoku (hereafter JS) is a Sudoku variant which doesn't deviate much from original Sudoku. Most rules of Sudoku still apply; There is a $n^2 \times n^2$ grid where each row and column must contain the distinct numbers $1, 2, ..., n^2$. However, JS deviates from original Sudoku in its constraint in subgrids. In original Sudoku, the $n^2 \times n^2$ grid is further divided in $n \times n$ blocks where all numbers ought to be distinct, like in the rows and columns. In JS, subgrids still maintain their property of distinct numbers, but don't need to be of size $n \times n$, rather, they can be a collection of shapes each containing n^2 cells such that the full $n^2 \times n^2$ grid can be filled in.

The modification JS has in its subgrid constraint also makes it possible to have working puzzles with a chosen n^2 s.t. $n \notin \mathbb{N}$, i.e. the length along the dimensions don't have to be a number having a perfect square root. Therefore, modifications in definitions of JS with respect to original Sudoku are made to the following: The grid of JS is of size $q \times q$ and the distinct numbers property of the rows, columns and subgrids need to adhere to the range $1, 2, ..., q$, where $q \in \mathbb{N}$. In addition, The position of the cells are defined as (x, y), x-axis and y-axis respectively, where $x, y \in \{1, 2, ..., q\}$ and $(1, 1)$ is the top-left most cell. Each cell (x, y) in a Jigsaw Sudoku p has a value, which will be denoted as $p(x, y) = v_{(x,y)}$ interchangeably, where, as previously mentioned $p(x, y) \in \{1, 2, ..., q\}$. An example of a Jigsaw Sudoku can be found in figure 3.1.

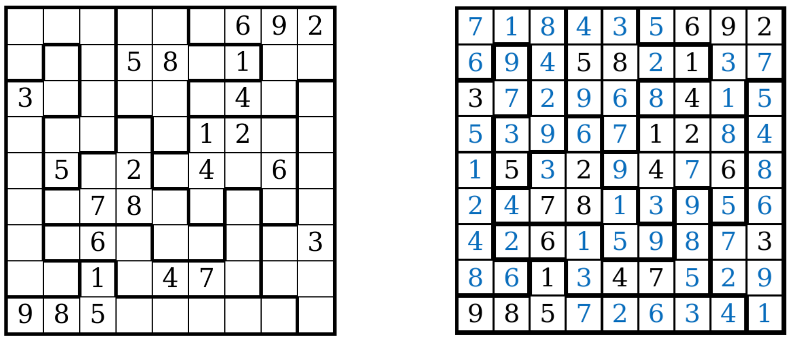


Figure 3.1: 9 × 9 Jigsaw Sudoku with Solution

3.1.2 Proof: Jigsaw Sudoku is NP-Complete

Whilst not completely necessary, it is nice to prove that an arbitrary $q \times q$ JS p indeed is a NP-hard and in particular NP-complete problem. To prove NP-completeness, it must be shown that $p \in NP$ and $p \in NP$-hard.

To prove $p \in NP$, we will show that given certificate p^* (a solution to p) there exists a polynomial time algorithm such that it can be verified if p^* indeed is a solution to p. This can be easily realised by defining an algorithm which checks for all rows, columns and subgrids whether their elements are equal to the (unordered) set of the elements containing $\mathbb{N}_q$, which can be done in $\mathcal{O}(q^2)$. Additionally, next to checking whether p^* has any constraint violations, it can be checked that p^* contains the initial values of p in the correct locations by going over the full grid of p^* and checking cell by cell for correspondence, which is done in $\mathcal{O}(q^2)$ as well.

To prove $p \in NP$-hard, we have to reduce a known NP-hard or NP-complete problem to JS with a polynomial transformation. As Sudoku (S) is known to be NP-complete [28] and it can be shown by definition that $S \subset JS$ (where S and JS denote the set of all possible Sudoku and Jigsaw Sudoku configurations respectively), it is implicit that the hardest JS problem is at least as hard as problem as the hardest S problem. Specifically, there exists a trivial (identity) reduction from S to JS.

3.1.3 Σ - Protocol

We produce a zero-knowledge proof system, heavily inspired on *Protocol 1* in *"Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles"* [17]. In this ZKPS, the prover $\mathcal{P}$ makes use of the ambiguity in variable swapping a solved (Jigsaw) Sudoku has in its solution. For example, all 1's and 2's can be swapped in a Sudoku, whilst not changing the structure and therefore the validity of the solution. This design choice of the first protocol has been made for its high level of intuition and to illustrate that a modification in constraints can still work with the same protocol. The protocol is given in figure 3.2. For a more detailed view, see figure 3.3.

To be able to make a shuffled solution, $\mathcal{P}$ will produce a function σ denoting a random mapping of cell values (1 in figure 3.2). After applying permutation function σ to each cell of the solution p^* (2 in figure 3.2), a commitment a of this permutation is made and subsequently sent to the verifier $\mathcal{V}$ (3 in figure 3.2). Having received commitment a, $\mathcal{V}$ now (uniformly random) picks a challenge c (4 in figure 3.2). This challenge is for $\mathcal{P}$ to reveal the permuted cells of either one of q rows, columns or subgrids, or to show the permuted values of the initial filled-in cells of p. After sending the chosen challenge c to $\mathcal{P}$ (5 in figure 3.2), $\mathcal{P}$ will compliantly respond with formulating response r (6 in figure 3.2) and sending it back to $\mathcal{V}$ (7 in figure 3.2). Finally, $\mathcal{V}$ will check whether the commit and the criteria of the puzzle check out (8 in figure 3.2).

	$\mathcal{P}$		$\mathcal{V}$
	JS p, JS solution p^*		JS p
1	$\sigma \overset{\$}{\leftarrow} \{1,2,...,q\} \mapsto \{1,2,...,q\}$		
2	$a \leftarrow Commit(\sigma(p^*))$		
3		$\xrightarrow{a}$	
4			$c \overset{\$}{\leftarrow} \{\text{row } x, \text{col } x, \text{sub } x, \text{init}\}$
			, where $x \in \{1,2,...,q\}$
5		$\xleftarrow{c}$	
6	$r \leftarrow \sigma(p^*)$, for each cell in c		
7		$\xrightarrow{r}$	
8			$Open(a,r)$ and
			if $c =$ init:
			check permutation validity
			else $c =$ other:
			check uniqueness values

Figure 3.2: Σ - Protocol 1, JS

Having explained the general structure of the protocol, to be more elaborate and precise, the system will again be explained, but now more thoroughly with help of figure 3.3; To be able to make a shuffled solution, $\mathcal{P}$ will produce a function σ denoting a random picking of $\{1,2,...,q\} \mapsto \{1,2,...,q\}$ (1 in figure 3.3). Now, a list of commitments a will be formed (2 in figure 3.3). This a, is a collection of individual cell commitments, denoted as $a(x,y)$. Note; The summation sign $\sum$ is being used as an indicator of an addition of elements forming a set, not as an addition in commitment value. The commitment scheme used, as mentioned in 2.2.3, is by use of a hash function h with as randomness aspect a salt s, generated from an arbitrary random number generator up to k. The commitment list a, subsequently, gets sent to $\mathcal{V}$ (3 in figure 3.3). $\mathcal{V}$ just having received this commit a, now chooses a challenge c from the set of potential challenges C consisting of q rows, q columns, q subgrids and the initial filled-in cells (4 in figure 3.3). This, subsequently, is sent to the *Prover* (5 in figure 3.3). The idea behind this specific challenge set is that a cheating *Prover* must lie in either

> at least one cell, in which case not all numbers are distinct in at least one criterion of a row, column or subgrid. $\mathcal{V}$ might discover this by challenging a row, column or subgrid as the response would include at least two identical values. In addition, notice that for the protocol to remain zero-knowledge, $\mathcal{P}$ can maximally only send one row, column, subgrid or initial value in response to a challenge per iteration of the protocol. If multiple were challenged and given in the later response by $\mathcal{P}$, then $\mathcal{V}$ could match the permutations with the initial values and step by step gain knowledge of the solution (see appendix A.1). When shown the soundness property (section 3.1.5), this example will be generalized for all cases.
>
> the permutation function σ, where $\mathcal{P}$ simply made/has a solved Jigsaw Sudoku, but of a different initial grid p'. This case gets caught by challenging the initial values, where it can be checked that the permutation indeed occurred properly on all distinct (and identical) variables on the correct positions of the initial grid.

When $\mathcal{P}$ receives challenge c, she prepares response r containing a list of tuples corresponding to the cells required in c. Each tuple contains the permutation of value $p^*(i,j)$ and it's corresponding salt $s_{(i,j)}$ (6 in figure 3.3). Subsequently, r is sent to $\mathcal{V}$ (7 in figure 3.3). $\mathcal{V}$ checks the response by checking if the response matches the commitment and if the criteria are not violated or if the permutation function has been applied correctly to the initial values (8 in figure 3.3).

	$\mathcal{P}$		$\mathcal{V}$
	JS p, JS solution p^*, hash function h		JS p, hash function h
1	$\sigma \overset{\$}{\leftarrow} \{1,2,...,q\} \mapsto \{1,2,...,q\}$		
2	$a \leftarrow \Sigma_{i=0}^{q}\Sigma_{j=0}^{q}\ a(i,j)$, where $a(i,j) = h(\sigma(p^*(i,j))\vert\vert s_{(i,j)})$, where $s_{(i,j)} \overset{\$}{\leftarrow} \{1,2,...,k\}$		
3		$\xrightarrow{a}$	
4			$c \overset{\$}{\leftarrow} \{\text{row } x, \text{col } x, \text{sub } x, \text{init}\}$, where $x \in \{1,2,...,q\}$
5		$\xleftarrow{c}$	
6	$r \leftarrow \Sigma_{i=0}^{q}\Sigma_{j=0}^{q} r(i,j)$ iff $p^*(i,j) \in c$, where $r(i,j) = (\sigma(p^*(i,j)), s_{(i,j)})$		
7		$\xrightarrow{r}$	
8			$(\forall i, j,$ s.t $a^*(i,j) \in c$, $a^*(i,j) = a(i,j)$, where $a^*(i,j) = h(\sigma(p^*(i,j)) \cdot s_{(i,j)}))$ $\wedge$ $((c = \text{init}) \leftrightarrow$ $(\vert p\vert = \vert r\vert$ $\wedge$ $\forall i, j, k, l$ s.t. $a^*(i,j) \in c$ $\wedge\ a^*(k,l) \in c$ $\wedge\ \neg(i = k \wedge j = l)$, $(\sigma(p^*(i,j)) \neq \sigma(p^*(k,l)))$ $\leftrightarrow (a^*(i,j) \neq a^*(k,l)))$ $\vee$ $(\vert r\vert = q$ $\wedge$ $\forall i, j, k, l,$ s.t $a^*(i,j) \in c$ $\wedge\ a^*(k,l) \in c$ $\wedge\ \neg(i = k \wedge j = l)$, $\neg(\sigma(p^*(i,j)) = \sigma(p^*(k,l)))))$

Figure 3.3: Σ - Protocol 1, JS, Elaborate Visualisation

3.1.4 Elaborate Example Runs

As this particular Jigsaw Sudoku ZKPS is still somewhat on the intuitive side, some elaborate examples have been enclosed in the appendix (See A.2) to get a better mental grasp on both the steps and validity of a played out scenario. Of course, in order to make this example cognitively feasible, the size of the Sudoku used here is almost trivial. The regions of the Jigsaw Sudoku have been colour-coded to make subgrid challenges clear.

3.1.5 Proof of required properties

Perfect Completeness

If honest $\mathcal{P}$ has an arbitrary $q \times q$ Jigsaw Sudoku p, with a corresponding solution p^*, then this implies that p^* still has the initial values of p and each of q rows, q columns and q subgrids has numbers $\{1, 2, ..., q\}$. Now, for randomly chosen permutation function $\sigma = \mathbb{N}_q \mapsto \mathbb{N}_q$, $\sigma(p^*)$ still remains a valid solution to $\sigma(p)$ (see lemma 3.1.1). Also, with σ, the position and relative equality between initial values doesn't change (see lemma 3.1.2).

For every challenge c that is one of q rows, q columns and q subgrids, $\mathcal{P}$ sends a response r containing $\sigma(p^*)$ of the cells prompted by this c. As proven, this $\sigma(p^*)$ will contain numbers $\{1, 2, ..., q\}$ proving no duplication of element in chosen c. This, combined with $\mathcal{V}$ being able to check if the response r corresponds with the received commitment a, causes $\mathcal{V}$ to accept.

On the other hand, when $\mathcal{V}$ chooses the initial values as her challenge, $\mathcal{P}$ sends $\sigma(p)$ as response r. As proven, this $\sigma(p)$ will be unchanged in relative values, thus, it can be easily checked that $\mathcal{P}$ worked with the correct pre-agreed p. This, combined with $\mathcal{V}$ being able to check if the response r corresponds with the received commitment a, causes $\mathcal{V}$ to accept.

Lemma 3.1.1. *For an arbitrary filled-in $q \times q$ Jigsaw Sudoku p^* which adheres to the criteria defined in the ruleset, that is, each of q rows, q columns and q subgrids has numbers $\{1, 2, ..., q\}$, and an arbitrary permutation function $\sigma : \mathbb{N}_q \mapsto \mathbb{N}_q$, $\sigma(p^*) = \{p^*(x, y) \mid x, y \in \mathbb{N}_q\}$ also adheres to the aforementioned criteria.*

Proof. As any row, column and subgrid in p^* has value set $\{1, 2, ..., q\}$, then it is implicit that in every one of those sets for any $a \in \{1, 2, ..., q\}$ and $b \in \{1, 2, ..., q\} \backslash \{a\}$ we have $a \neq b$. By definition of permutation function σ it can be noticed that as this is a bijective function it follows that $\sigma(a) \neq \sigma(b)$. As this reasoning applies for all elements in any row, column and subgrid, it follows that $\sigma(p^*)$ still is a Jigsaw Sudoku solution. $\square$

Lemma 3.1.2. *For an arbitrary, yet to be solved, $q \times q$ Jigsaw Sudoku p with a set of cells locations of the initial values $Z = \{z_1, z_2, ..., z_q\}$ and their respective values $p(z_1), p(z_2), ..., p(z_q)$, and an arbitrary permutation* *function $\sigma : \mathbb{N}_q \mapsto \mathbb{N}_q$, we have that for any $z_i, z_j \in Z$, where $i, j \in \mathbb{N}_q$ and $i \neq j$, if $p(z_i) = p(z_j)$ then $\sigma(p(z_i)) = \sigma(p(z_j))$ and if $p(z_i) \neq p(z_j)$ then $\sigma(p(z_i)) \neq \sigma(p(z_j))$.*

Proof. As the permutation function σ is bijective, it follows from definition that if $p(z_i) = p(z_j)$ then $\sigma(p(z_i)) = \sigma(p(z_j))$ and if $p(z_i) \neq p(z_j)$ then $\sigma(p(z_i)) \neq \sigma(p(z_j))$. $\square$

Special Soundness

Consider two accepting transcripts (a, c, r) and (a, c', r'), where $c \neq c'$. Now, there exists at least two response cells $(\sigma(v), (x, y))$ and $(\sigma(v'), (x', y'))$ s.t. $(x, y), (x', y') \in (c \cup c') \setminus (c \cap c') \wedge \sigma(v) = \sigma(v')$, where $\neg(x = x' \wedge y = y')$. Note, v serves as an intuitive abbreviation of $p(x, y)$, representing the value of an arbitrary cell. With the two cells, it is possible that either v or v' is known to extractor E due to knowledge of the initial values given in p. Then, if exactly one of the two of v and v' was known by E, now they both are due to σ being a permutation, therefore gaining partial knowledge of witness w. What's more, even if none of $(c_1 \cup c_2)$ are in the set of initial values, E still gains knowledge as equal permuted values imply equal pre-permuted values. Thus, gaining knowledge of the 'relative structure' of witness w.

As it is now straightforward to imagine that with the aforementioned technique extractor E can gain (full) knowledge of witness w (with k-special-soundness), it has been proved that this protocol is special sound. This realisation, along with the fact that the challenge set C has cardinality $3q + 1$, that is, q row challenges plus q column challenges plus q subgrid challenges and the challenge of initial values, a soundness error of at most $1 - \frac{1}{3q+1}$ can be set.

Honest-Verifier Zero-Knowledge

We define a p.p.t. simulator M as a machine which having access to JS p for arbitrary challenge c can define accepting conversation (a, c, r).

This is easily achieved in the case where c is a challenge in row, column or subgrid as M can, in polynomial time, generate any arbitrary solved JS p' such that a is a commitment and r the corresponding response with respect to challenge c. p', in this context, disregards the initial values of p as this won't be checked in this challenge.

In the case of c being a challenge in initial values, M can generate a commitment a and corresponding r of an in-polynomial-time generated JS p' with the initial values of p still present in the correct cells but otherwise filled in with all 1's (or any arbitrary filling with numbers from the set $\mathbb{N}_q$ for that matter). The fact that p' is not a solution is no problem, as challenge c won't check any of the criteria besides correct permutation of the initial values.

3.2 Protocol 2 - Jigsaw Sudoku; An Improvement

3.2.1 Rules and Definitions

See 3.1.1.

3.2.2 Σ - Protocol

This proposed sigma protocol is identical to protocol 1 (See 3.1.3) with one alternation, that is, we enlarge the challenge set C with an additional challenge of the initial values so that it is twice as likely that an honest verifier $\mathcal{V}$ uniformly randomly picks the *initial values* as her challenge. The usefulness of this addition will become apparent in the soundness analysis.

3.2.3 Proof of required properties

Perfect Completeness

As merely the likelihood of a uniformly random picking from C has changed, and the proof in protocol 1 regarding perfect completeness (see 3.1.5) was formalized for an arbitrary challenge picking, it follows that its argumentation also holds for this protocol.

Special Soundness

For Jigsaw Sudoku, it can be proven that if there exist a violation of a constraint in a row, column or subgrid, it implies that there must at least be one other row, column or subgrid being violated, as can be noticed from *Lemma 3.2.3* in section 3.2.4 respectively. Here, one constraint means that one row, column or subgrid doesn't contain values $\{1, 2, .., q\}$.

Now, with the addition of the extra *initial values* in challenge set C, and similar underlying argumentation of the previously determined soundness error in protocol 1 (see 3.1.5), the following can be deduced: $\mathcal{P}'$ now has the minimum chance $\frac{2}{3q+2}$ of being caught by invalid solution on constraint violation (as proven by *Lemma 3.2.3*) and an exact chance of $\frac{2}{3q+2}$ being caught on changing to a valid solution with different initial values. This makes for an improved soundness error bound of a theoretical minimum of $1 - \frac{2}{3q+2}$, which for a sufficiently large q (which for any computationally non-trivial example is the case) essentially doubles the originally defined soundness of previously proposed protocol 1.

This improvement in soundness is also realisable on *Protocol 1* of [17] if the protocol, just as this one, adds one additional *initial values* to their challenge set. The reason why the lemma's (see 3.2.4) also apply on normal Sudoku is that Sudoku is a subset of Jigsaw Sudoku by definition of their ruleset and structure (see also 3.1.2).

Special Honest-Verifier Zero-Knowledge

Same proof as 3.1.5, which is allowed to be used due to the same argumentation declared in 3.2.3.

3.2.4 Proof: Minimal 2 Constraint Violations

Lemma 3.2.1. *Given a filled-in $q \times q$ JS p', if there is a violation of one constraint with respect to the row, column or subgrid, then there is at least one other row, column or subgrid which is violated.*

Proof. Proof by contradiction. Let's assume that JS p' breaks exactly one constraint. This implies that there are two cells (x_1, y_1) and (x_2, y_2) with $p'((x_1, y_1)) = p'((x_2, y_2)) = v$, where $\neg(x_1 = x_2 \wedge y_1 = y_2)$ and $(x_1, y_1), (x_2, y_2) \in c$. Here, c is one of q constraints belonging to the group g_c, where $g_c \in \{\text{rows, columns, subgrids}\}$. The other two left-over groups, which c is not a part of, are called g_1 and g_2.

Now, as all other $(q-1)$-amount of g_c, that is, g_c numbers $\mathbb{N}_q \backslash \{c\}$, also need to have exactly one value v, it implies that in the whole grid there exist $(q+1)$-amount of v's. However, as there are exactly q of g_1 and q of g_2, this means, by the pigeonhole principle, that there is at least one constraint in g_1 and one constraint in g_2 also having a violated constraint due to a duplicated value v, concluding that with this case there are at least 3 distinct constraint violations with regards to rows, columns and subgrids.

The alternative case is choosing to have q-amount of value v's despite the duplicate of v in c. This has the unavoidable consequence of having one $c' \in g_c \backslash \{c\}$ with no value v as there are $(q-1)$-amount of constraints left in g_c whilst there are only $(q-2)$-amount of v's left. Again, pigeonhole principle. If there is no value v in row c', then there is a value v' s.t. $v' \in \{1, 2, ..., q\} \backslash \{v\}$ which has two occurrences in c', also, by the pigeonhole principle due to having to fill in (q)-amount of values in $(q-1)$-amount of cells. This results in, next to having c containing duplicate v, c' containing duplicate v', concluding that with this case there are at least 2 distinct constraint violations with regards to rows, columns and subgrids. □

Lemma 3.2.2. *For any $q \times q$ JS p', there exists a filled-in grid such that exactly two distinct constraints have been violated with respect to different rows, columns or subgrids.*

Proof. From any filled in JS p^* which has no constraint violations, we take cells $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$, where $\neg(x_1 = x_2 \wedge y_1 = y_2)$ and $p^*(x_1, y_1) \neq p^*(x_2, y_2)$ such that c_1 and c_2 share either a column and a subgrid or a row and a subgrid (which by definition in puzzle structure always has an occurrence in any $q \times q$ JS, where $2 \leq q$).

In the case that c_1 and c_2 share the same row and subgrid, it implies that they occupy different columns, otherwise they would be the same cell.

When the values of c_1 and c_2 are swapped, forming p', the row and subgrid have no constraint violation as the values of their sets haven't changed. However, due to the swap, the two columns c_1 and c_2 originally occupied now both have a duplicate value in their sets, giving 2 constraint violations. The exact same reasoning can be applied for the alternative case of a c_1 and c_2 swapping rows which originally shared the same column and subgrid. □

Lemma 3.2.3. *In a filled in JS of any size which violates a constraint with respect to distinct rows, columns and subgrids, the minimum amount of distinct constraint violations is 2.*

Proof. Follows from *Lemma 3.2.1* and *Lemma 3.2.2.* □

3.3 Protocol 3 - Tents

3.3.1 Rules and Definitions

A game of Tents (T) is played on a $n \times n$ grid. In this grid, a cell can have 3 possible values; *tent*, *tree*, *empty.* The game starts out by the grid being partially filled with trees, as the rest of the remaining cells yield empty. The goal of the game is for the player to place tents in the grid such that the following criteria are met:

1. Pair each tree with a tent adjacent horizontally or vertically. This should be a 1-to-1 relation.

2. Tents never touch each other, not even diagonally.

3. The *clues* outside the grid indicate the number of tents on that row/-column.

In figure 3.4 an example of a starting grid p and its respective solution p^* of a game of Tents is given.

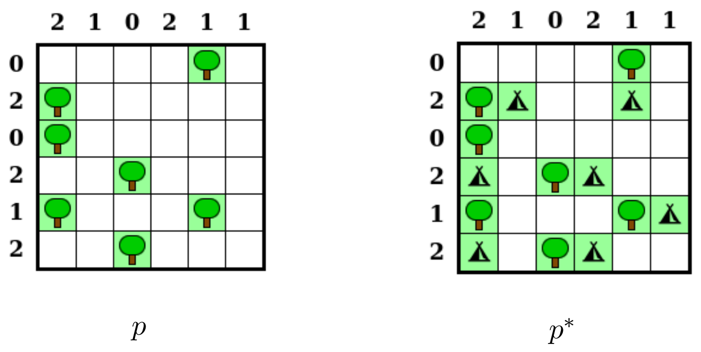


Figure 3.4: Game of Tents

3.3.2 Σ - Protocol

A Σ-protocol will be produced, inspired on *Protocol 2* in *"Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles"* [17]. The general idea of this protocol is that for a solved tents game p^* (in regard to unsolved p) the prover makes copies of the solution, which are concatenated to a 3rd dimension by allocating each copy an additional z-coordinate, after which a uniformly random picked permutation σ is applied on all indices, essentially shuffling all values with all locations. With the introduction of copies, all criteria can be checked without index overlap of cells, which would otherwise give away partial positional knowledge.

As like the previously discussed protocols, a figure (3.5) has been produced to keep track of the broader encapsulation of the protocol. Note that, as this protocol is significantly larger and more complex than the aforementioned protocols, the figure only discusses surface-level operations. It has been contemplated to work out a figure displaying a higher level of depth, but after careful consideration it has been determined that this would not only get incredibly large and complex, but also not particularly of the essence as this protocol can be better (more intuitive) explained at surface-level.

In this protocol, a cell is defined as a location (x, y, z), where $x, y \in \{0, 1, ..., n+1\}$ and $z \in \{1, 2, ..., 11\}$ (argumentation regarding the ranges of indices will follow in 3.3.3), with a corresponding value $p^*_{11}(x, y, z) = v_{(x,y,z)}$, where v is the value such that $v \in \{empty, tent\}$.

The way $\mathcal{P}$ produces p^*_{11} is as follows: Add a 1-cell thick border to p^* all with values equal to empty. This gives a new grid size $(n+2) \times (n+2)$, with the set of possible locations now consisting of $\mathbb{N}^0_{n+1} \times \mathbb{N}^0_{n+1}$ with as top-left cell index $(0, 0)$. The two rows and two columns clues of the added border are consistent with the amount of trees in them, namely 0. Subsequently, all cells with the value of a tree in the enlarged grid will be assigned the empty value. Lastly, $\mathcal{P}$ makes 11 copies and puts them along the newly defined z-axis, making the set of possible locations $\mathbb{N}^0_{n+1} \times \mathbb{N}^0_{n+1} \times \mathbb{N}_{11}$. Visual examples can be found in appendix A.3. Note; argumentation for many aforementioned actions can be found in section 3.3.3.

To be exact; the permutation function σ is a uniformly random picking of $\{(\mathbb{N}^0_{n+1}, \mathbb{N}^0_{n+1}, \mathbb{N}_{11})\} \mapsto \{(\mathbb{N}^0_{n+1}, \mathbb{N}^0_{n+1}, \mathbb{N}_{11})\}$.

Next, 5 distinct types of commits will be made; a_1, a_2, a_3, a_4 and a_5 (2 in figure 3.5):

- a_1: Commit to $11(n+2)^2$ values of the permuted indices, that is $\sigma(p^*_{11})$. These values are ordered on the permuted indices; the verifier doesn't know the pre-permuted locations of the values from the order.

- a_2: Commit to $(n+2)^2$ unordered sets of size 11 of locations in a_1, where each set corresponds with the copies made of a cell in p^* (with border). That is $\{\{\sigma(x, y, 0), \sigma(x, y, 1), ..., \sigma(x, y, 11)\} \mid x, y \in \mathbb{N}^0_{n+1}\}$.

a_3: Commit to the pre-permuted (x, y) position on p^* for each set in a_2.

a_4: Commit to a collection of sets of locations of a_1 corresponding to shared criteria, where

i. $(n+2)$ unordered sets of size $n+2$ corresponding to *rows*, each along with the respective row clue.

ii. $(n+2)$ unordered sets of size $n+2$ corresponding to *columns*, each along with the respective column clue.

iii. $(n+2)^2$ unordered sets of size 4 corresponding to all possible connected 2×2 blocks in the grid.

iv. $(n+2)^2$ singleton sets (one location) corresponding to the center of all possible connected crosses in the grid.

v. $(n+2)^2$ unordered sets of size 4 corresponding to neighbours of the center (*iv*) of all possible connected crosses in the grid.

a_5: Commit to $(n+2)^2$ relations between $a_4.iv$ and $a_4.v$. That is, which center of a cross ($a_4.iv$) corresponds to which set of neighbours ($a_4.v$).

Subsequently, the commits are being sent to the verifier $\mathcal{V}$ (3 in table 3.5). Now, $\mathcal{V}$ can (randomly) pick from challenge set C, consisting of 3 challenges:

c_1: Open a_1 and a_4, verify the following:

1. The amount of tents in all $(n+2)$ rows and $(n+2)$ columns are equal to the respective numeric clues adjoined with sets defined in $a_4.i$ and $a_4.ii$.
2. From (1.), the total collection of adjoined clues, whilst unordered, should in cardinality and size be equal to those defined in p. Keep in mind, there will be four 0's left due to the additionally generated 2 rows and 2 columns for the edge.
3. From the $(n+2)^2$ sets of 2×2 blocks, check that no single of those unordered sets has 2 tents as, if this is the case, it implies that there are neighbouring tents, violating the neighbouring criteria defined in the ruleset.
4. Check that the total amount of tents in

   $a_4.i$ is equal to the amount of trees in p.

   $a_4.ii$ is equal to the amount of trees in p.

   $a_4.iii$ is equal to 4 times the amount of trees in p.

   $a_4.iv$ is equal to the amount of trees in p.

   $a_4.v$ is equal to 4 times the amount of trees in p.

c_2: Open a_1 and a_2, check that all values of each set in a_2, representing copies of each original cell in p^*, correspond to the same value v in a_1; verify that the cells are indeed copies. Additionally, verify that no two sets of copies intersect and that the total amount of tents indeed is 11 times the amount of trees in p.

c_3: Open a_2, a_3, a_4 and a_5 as well as all elements of a_1 corresponding to the locations of the trees in p and their neighbours and verify the following:

1. With the permuted indices of tree locations in a_1 and sets of indices denoting copies a_2, check whether copying has been done honestly (by checking equality in values in sets like done in c_2).
2. With the permuted indices of tree locations in a_1, sets of indices denoting copies a_2 and the actual (x, y) location of these sets from a_3, check whether this corresponds to the tree locations in p.
3. With the permuted indices of tree locations in a_1, sets of indices denoting copies a_2, the actual (x, y) location of these sets from a_3, and the relation of each cell to their cross with a_5 and a_4, check whether
   i. for the corresponding unordered sets of the trees' neighbours (a_4, via a_5), each set contains at least one tent.
   ii. The relation described in a_5 is bijective.

The uniformly random picked challenge c will be sent to $\mathcal{P}$ (5 in table 3.5), who prepares her corresponding response r (6 in table 3.5), sends it to $\mathcal{V}$ (7 in table 3.5), who then accepts or rejects accordingly (8 in table 3.5).

	$\mathcal{P}$		$\mathcal{V}$
	Tents p, Tents solution p^*		Tents p
1	$\sigma \overset{\$}{\leftarrow} \{(\mathbb{N}^0_{n+1}, \mathbb{N}^0_{n+1}, \mathbb{N}_{11})\} \mapsto \{(\mathbb{N}^0_{n+1}, \mathbb{N}^0_{n+1}, \mathbb{N}_{11})\}$		
2	generate $a = a_1 \| a_2 \| a_3 \| a_4 \| a_5$		
3		$\xrightarrow{a}$	
4			$c \overset{\$}{\leftarrow} \{c_1, c_2, c_3\}$
5		$\xleftarrow{c}$	
6	$r \leftarrow \{r_1, r_2, r_3\}$		
7		$\xrightarrow{r}$	
8			check commit and c

Figure 3.5: Σ - Protocol Tents

3.3.3 Specific Design Choices and Limitations

Sending Trees Leaks Knowledge

Consider the scenario where sending trees as values of cells is allowed. Now, it can be shown that, at the least, knowledge gets leaked if the verifier picks a challenge such that, among other things, the contents of the $(n + 2)^2$-amount of unordered sets denoting the 2×2 blocks need to be shown (in our case this is c_1). This is because a distribution can be made of the content of those 2×2 blocks, which gives partial knowledge with regard if (many) tents neighbour more than one tree. To further elaborate by example; say there is a 2×2 block where one value is a tent and 3 values are trees, then knowledge has been gained that the value of one of the cells in p which is adjacent to 3 trees is a tent, which didn't need to be the case.

How To Check The 1-to-1 Relation Criteria

To check the 1-to-1 relation (injective function) between the trees and tents, it can first be checked that the total amount of tents is equal to the total amount of trees in p and, secondly, that each cross with a tree in the middle ought to have at least one tent as neighbour. To put short and intuitive; if the amount of tents is equal to the amount of trees and each tree has a tent, it implies that each tree has one tent and thus each tent one tree.

Inconveniently enough, due to the limitation of inability to send tree-values in the commits, there is no proper way to check whether each tree has a tent in c_1. Doing so requires revealing cell locations of $a_4.iv$ and, via a_5, their relation to the sets of respective neighbours ($a_4.v$), as otherwise a malicious prover can trivially construct these sets (give every set at least one tent). Sending cell locations in c_1 would reveal locations of other cells as a_2 and a_3 would need to be (partially) revealed. Thus giving away knowledge. Checking this requirement however can be realized in c_3 as that challenge deals with actual locations of initial values (which are the trees).

On the flip side, in c_3 we are unable to check if the amount of tents equals the amount of trees in p, as this challenge only focuses on verifying the initial values (the trees). Doing this requires revealing actual (x, y) locations, which, in combination with the need to reveal all a_1 (for counting tents), would also give away positional knowledge of tents.

This is why the twofold requirement to prove the 1-to-1 relation is split up between c_1 (checking if the amount of trees equals the amount of tents) and c_3 (checking if every tree has at least one tent). Even though proving the criteria is now split up between challenges, it is not an issue, as a prover failing to comply with the 1-to-1 relation rule gets caught in either c_1 or c_3.

Adding a One-Cell-Thick Edge

The introduction of the sets in a_4 representing centers and neighbours of 'crosses', needed to check for 1-to-1 relations, comes with a caveat. If a cross needs to be defined on the edge of the puzzle, there might be one or two cells which go out of bounds (see figure 3.6, note: a 'T' denotes a tree and a '-' denotes a tent). We cannot simply ignore the cell(s) out of bounds as it would result in a set with a cardinality lower than 5, suggesting that the pre-permuted locations of these cells are close to or on an edge, giving away knowledge.

Now, a seeming solution avoiding this phenomenon is wrapping around the puzzle (see figure 3.7). However, here, the question mark could have been a tent, possibly passing a constraint check whilst the tree actually has no neighbouring tent, enabling a malicious prover to take advantage of this unreasonable relaxation in the protocol. Additionally, when wrapping 2×2 blocks, which otherwise faces the same issues of knowledge extraction as the crosses when defined at the edge, a valid statement could be flagged as invalid due to neighbouring tents when wrapping.

To work around this problem, an enclosing edge is introduced around the puzzle. As with this introduction of empty cells, the degree of passing criteria of crosses and 2×2 blocks are unmodified; wrapping is not a problem anymore for both crosses and 2×2 blocks (see figures 3.8 and 3.9). Additionally, now, every cell has the same total amount of needed duplication. This aspect is crucial as, like mentioned prior, a quantitatively unequal distribution gives (slight) bias towards the indices the permuted cells are derived from, gaining partial knowledge of original positioning. Note; It is because of this introduced border that many things get defined as $(n+2)^2$, as this is the new size with respect to the original grid size n^2. Additionally, the added border cells have pre-permuted x- and y-coordinate 0 and/or $n+1$.

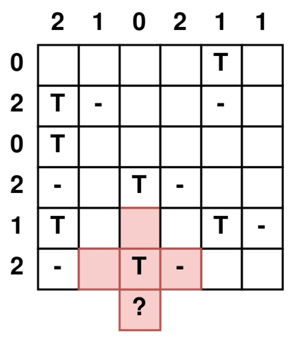


Figure 3.6: Cross Out of Bounds

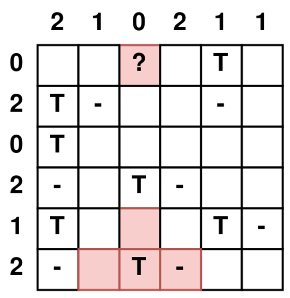


Figure 3.7: Borderless Wrap

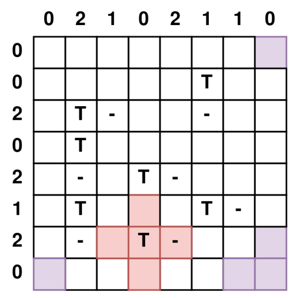


Figure 3.8: Cross Wrapping

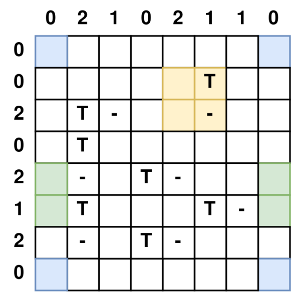


Figure 3.9: Block Wrapping

3.3.4 Proof of required properties

Completeness

If honest $\mathcal{P}$ has an arbitrary $n \times n$ Tents p, with corresponding solution p^*, then this implies that in p^* all trees are in the same locations as they are in p as well as the row- and column-clues being unmodified with respect to p. Also, in p^*, tents have been added such that the three criteria defined in the ruleset (see 3.3.1) are adhered to. It follows that it is assumed that $\mathcal{P}$ correctly produces commitments a_1, ..., a_5 after having constructed p^*_{11} as described in 3.3.2 and permutation function σ being a randomly picked of $\{(\mathbb{N}^+_{n+1}, \mathbb{N}^+_{n+1}, \mathbb{N}_{11})\} \mapsto \{(\mathbb{N}^+_{n+1}, \mathbb{N}^+_{n+1}, \mathbb{N}_{11})\}$.

For challenge c_1, it can be noticed that every of its verifications essentially is a check in criteria. If p^* indeed is a solution, then all criteria indeed are met. And, given the aforementioned transformations required to produce p^*_{11} and subsequently a_1 and a_4, it can be noticed that as we are working with copies, these criteria/constraints still hold given that $\mathcal{P}$ produces and sends a corresponding response r_1 (opening a_1 and a_4).

For challenge c_2, it can be noticed that it is a verification regarding if the copying has occured honestly. That is; for all $x, y \in \mathbb{N}^+_{n+1}$ and $z, z' \in \mathbb{N}_{11}$, where $z \neq z'$, we have $p^*_{11}(x, y, z) = p^*_{11}(x, y, z')$. this clearly is the case if $\mathcal{P}$ copied (transformed) solution p^* and produced correct commitments a_1 and a_2 and response r_2 (opening the two).

For challenge c_3, it can be noticed that it is a verification in initial values (with checking crosses on 1-to-1 relation), this always gets accepted in the case of p^* as the initial values haven't changed and produced correct commitments a_1, a_4, a_3, a_4 and a_5 and response r_3 (opening all of them and a_1 only partial) can show this as defined in argumentation on how c_3 is checked.

All in all, a prover which has solution p^* to arbitrary $n \times n$ Tents p can always convince a verifier.

Special Soundness

Consider two accepting transcripts (a, c, r) and (a, c', r'), where $c \neq c'$. Now, as challenge set C only has tree distinct challenges it can be shown exhaustively that any combination c and c' gives away (part of) witness w.

In the case of transcripts (a, c_1, r_1) and (a, c_2, r_2) it can be deduced that given the sets of permuted locations used to verify criteria (c_1) and given all $(n + 2)^2$ sets of permuted locations to verify correct copying (c_2), these sets can be connected by 'linking' every permuted cell in c_1 with its equal in c_2, essential forming a map of correlations regarding relative positions in p^*. With this, a grid of all relative positions of tents can be constructed, which, when shifted over p such that it aligns with the trees (which can be done in polynomial time), reveals the witness.

In the case of transcripts (a, c_1, r_1) and (a, c_3, r_3) it can be shown that that given *full* reveal of all values with paired permuted indices (a_1) and their relations used for checking criteria (a_4) (both given in c_1) and getting all (x, y) locations of the copies (reveal of a_2 and a_3 in c_3) it is easy to acknowledge that now all locations of all values are known.

In the case of transcripts (a, c_2, r_2) and (a, c_3, r_3) it can be shown that that given *full* reveal of a_1 and their sets of copies in (c_2) and getting all (x, y) locations of the copies (reveal of a_2 and a_3 in c_3) it is easy to acknowledge that now all locations of all values are known.

Following from the aforementioned, it has been proved that this protocol is special sound. This realisation, along with the fact that the challenge set C has cardinality 3, a soundness error of at most $1 - \frac{1}{3} = \frac{2}{3}$ can be set.

Honest-Verifier Zero-Knowledge

We define a simulator M as a machine which, having access to Tents p, for arbitrary challenge c can define accepting conversation (a, c, r).

This is achieved in the case where $c = c_1$ as M can exploit the fact that in this challenge, verification of correct duplication isn't prompted. This essentially entails that M can generate five filled-in Tent games (p'_i,p'_{ii},p'_{iii}, p'_{iv} and p'_v) such that when they get added a border, get duplicated, they can be selectively combined, after which permuted, to pass all criteria checked in c_1. To further elaborate:

> p'_i is a solution of p only in the row clues, and gets used as commitment of $a_4.i$.
>
> p'_{ii} is a solution of p only in the column clues, and gets used as commitment of $a_4.ii$.
>
> p'_{iii} is a solution of p only in the 2×2 blocks, and gets used as commitment of $a_4.iii$.

p'_{iv} is a solution of p only in the center of all possible connected crosses, and gets used as commitment of $a_4.iv$.

p'_v is a solution of p only in the neighbours of all possible centers, and gets used as commitment of $a_4.v$.

Remark. All p'_i,p'_{ii},p'_{iii}, p'_{iv} and p'_v can be constructed in polynomial time due to the heavily decreased complexity of only having to meet one criterion.

So, after adding a border and copying all p'_i,p'_{ii},p'_{iii}, p'_{iv} and p'_v 11 times, resulting in p'_{11-i},p'_{11-ii},p'_{11-iii}, p'_{11-iv} and p'_{11-v}, M will construct p'_{11} as the following:
$\forall x, y \in \mathbb{N}^+_{n+1}$,
$p'_{11}(x, y, 1) = p'_{11-i}(x, y, 1)$
$\wedge$ $p'_{11}(x, y, 2) = p'_{11-ii}(x, y, 2)$
$\wedge$ $p'_{11}(x, y, z) = p'_{11-iii}(x, y, z)$, where $z \in \{3, 4, 5, 6\}$
$\wedge$ $p'_{11}(x, y, 7) = p'_{11-iv}(x, y, 7)$
$\wedge$ $p'_{11}(x, y, z) = p'_{11-v}(x, y, z)$, where $z \in \{8, 9, 10, 11\}$
Now, when p'_{11} gets permuted and respective commitments are produced, for challenge c_1 and respective response r_1 (revealing a_1 and a_4), every verification on the criteria defined in c_1 will be met.

In the case that $c = c_2$, only the validity of copying gets checked. An accepting conversation can be easily simulated by generating p' which is a filled-in grid of p with the exact amount of tents as there are trees. Where the tents are placed is irrelevant (as long as it is not on the locations of trees in p); the amount of broken criteria is not important. Therefore, it follows that in this instance, filling in tents is a computation of polynomial time. Next, like in the described Σ-protocol, M adds a border around p', replaces the cells with value of trees by value empty. Now, it is only of essence that the copying and permuting gets done as intended, that is, even though p' is not a solution of p, the copying-part of the protocol has occurred accurately. Now, conversation (a, c_2, r_2) will be accepting, as it only checks for equality in the amount of trees and tents and if the sets of copies are equal in value and don't overlap.

In the case that $c = c_3$, M can generate accepting transcript (a, c_3, r_3) by generating filled-in Tents p', which is equal to p and with each tree having at least one tent adjacent to it, disregarding if it violates the criteria of the clues in rows and columns and the fact that no 2 tents can neighbour. This can be done in polynomial time. Next, M can follow the generating process of p'_{11} as described in the Σ-protocol. With this p'_{11} (and subsequent generation of a and response r_3), all checks done for challenge c_3 should be accepted as the trees are in the correct place, with correct copying and each tree has a tree with correct 1-to-1 relation of center to neighbouring cells (a_5).

Conclusions

We described three Σ-protocols, the designs of which inspired by R. Gradwohl, M. Naor, B. Pinkas, and G.N. Rothblum [17], on the Sudoku variants Jigsaw Sudoku and Tents. Whilst formalizing the protocols, several gradual discoveries and realisations were made regarding scope of commitments, soundness bounds and ensuring the property of zero-knowledge. Additionally, it is to hope that the order of the previously mentioned and their argumentation provided an intuitive path to a more complete understanding of interactive zero-knowledge proof systems and give a tasting of its design flexibility.

The protocols described on Jigsaw Sudoku are more oriented toward intuition, whilst the protocol described on Tents is more complex. What's more, the protocol on Tents is more efficient regarding soundness than the ones of Jigsaw Sudoku, given that both puzzles have identical grid sizes (which are sufficiently large). However, in the protocols of Jigsaw Sudoku, the size of commitments, challenges and responses per round are smaller. Again, showing interesting consequences in design choices. This observation, however, is a somewhat weak argumented as the protocols are constructed to cater different puzzles. So, deriving general conclusions on the respective efficiency of the protocol structures is far-fetched, especially with only 2 actually distinct protocols (which is also why it hasn't been discussed in depth). Still, it is interesting.

A more important realisation is that identical design concepts can be applied to different types of puzzles (from [17]), with Tents being particularly deviant. However, it has its limits, as, obviously, the concept of protocol 1 and 2, used for Jigsaw Sudoku, would not have worked on Tents.

Open Problems

It would be interesting to see potential improvements on the described Σ-protocols. This can range from fresh insights in puzzle structure providing improved soundness bounds, to mutating the entire protocol by working from a different angle.

Of course, it would also be exciting to see the protocols expended upon in regard to different commitment techniques, possibly making the full protocol more efficient.

Another expansion is converting the current interactive Σ-protocols into non-interactive zero-knowledge proof systems and observe what the accompanying results entail.

It would also be worthwhile to expend definitions, proofs and examples of protocol 3 (on Tents) to a more detailed level, and see if any additional realisations can be made from this.

Last, and probably most obvious, it would be interesting to see constructions of zero-knowledge protocols on different Sudoku variants, or puzzles in general. Perhaps, when enough are made, conclusions could be drawn of certain puzzle-properties which are particularly (un)advantageous, potentially giving insight in what logical structures make a good/efficient basis to construct zero-knowledge proofs around.

Bibliography

[1] Thomas Attema and Serge Fehr. Parallel repetition of $(k_1, ..., k_\mu)$-special-sound multi-round interactive proofs. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 415–443, Cham, 2022. Springer Nature Switzerland.

[2] Md Abdul Based and Stig Fr Mjølsnes. A non-interactive zero knowledge proof protocol in an internet voting scheme. In *Proceedings of the the 2nd Norwegian Security Conference (NISK 2009)*, pages 148–160. Tapir Akademisk Forlag, 2009.

[3] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 313–314, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[4] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, jan 1983.

[5] Manuel Blum, Paul Feldman, and Silvio Micali. *Non-interactive zero-knowledge and its applications*, page 329–349. Association for Computing Machinery, New York, NY, USA, 2019.

[6] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, and Jens Groth. Efficient zero-knowledge proof systems. In Alessandro Aldini, Javier Lopez, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VIII: FOSAD 2014/2015/2016 Tutorial Lectures*, pages 1–31, Cham, 2016. Springer International Publishing.

[7] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography – PKC 2013*, pages 55–72, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[8] Ivan Damgård. On σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, 2010.

[9] Ivan Damgård, Jens Groth, and Gorm Salomonsen. The theory and implementation of an electronic voting system. In Dimitris A. Gritzalis, editor, *Secure Electronic Voting*, pages 77–99, Boston, MA, 2003. Springer US.

[10] Anjali Deshmukh, Alin Nagraj, and Mansi A Radke. Sudokrypt: A novel sudoku based symmetric encryption scheme. *Journal of Network and Information Security Volume*, 4(2), 2016.

[11] Qi Feng, Debiao He, Sherali Zeadally, Muhammad Khurram Khan, and Neeraj Kumar. A survey on privacy protection in blockchain system. *Journal of Network and Computer Applications*, 126:45–58, 2019.

[12] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[13] Alexander Glaser, Boaz Barak, and Robert J. Goldston. A zero-knowledge protocol for nuclear warhead verification. *Nature*, 510(7506):497–502, Jun 2014.

[14] Oded Goldreich. *Foundations of Cryptology: Basic Tools*. Cambridge, 2001.

[15] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, jul 1991.

[16] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. *The knowledge complexity of interactive proof-systems*, page 203–225. Association for Computing Machinery, New York, NY, USA, 2019.

[17] R. Gradwohl, M. Naor, B. Pinkas, and G.N. Rothblum. Cryptographic and physical zero-knowledge proof systems for solutions of sudoku puzzles. *Theory of Computing Systems*, 44:245–268, 2008.

[18] Jens Groth. Non-interactive zero-knowledge arguments for voting. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 467–482, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[19] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Polynomial commitments. *Tech. Rep*, 2010.

[20] Benoît Libert, Somindu C. Ramanna, and Moti Yung. Functional Commitment Schemes: From Polynomial Commitments to Pairing-Based

Accumulators from Simple Assumptions. In *43rd International Colloquium on Automata, Languages and Programming (ICALP 2016)*, Track A (Algorithms, Complexity and Games), Rome, Italy, July 2016.

[21] Yehuda Lindell. An efficient transform from sigma protocols to nizk with a crs and non-programmable random oracle. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography*, pages 93–109, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[22] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, page 364–372, New York, NY, USA, 2005. Association for Computing Machinery.

[23] Somnath Panja and Bimal Kumar Roy. A secure end-to-end verifiable e-voting system using zero knowledge based blockchain. Cryptology ePrint Archive, Paper 2018/466, 2018. `https://eprint.iacr.org/2018/466`.

[24] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, and Soazig Guillou. How to explain zero-knowledge protocols to your children. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 628–631, New York, NY, 1990. Springer New York.

[25] C. P. Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York.

[26] Berry Schoenmakers. Lecture notes cryptographic protocols version 1.9 at TuE, February 2024.

[27] Huixin Wu and Feng Wang. A survey of noninteractive zero knowledge proof system and its applications. *The Scientific World Journal*, 2014(1):7, 2014. Article ID: 560484.

[28] Takayuki Yato. Complexity and completeness of finding another solution and its application to puzzles. Master's thesis, Univ. of Tokyo, Dept. of Information Science, Januari 2003. Availible: `http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/MasterThesis.ps`.

Appendix

A.1 Protocol 1 - Example Knowledge Leakage for Larger c

To better visualise why a larger picking of the challenge set leaks knowledge of σ^{-1} and thus the witness, let's use an example. Figure A.1 shows a Jigsaw Sudoku p and it's solution p^* respectively. Then, in figure A.2, we see one iterative step, where the challenges are *row* 1 and *row* 3. Due to knowledge of p, $\mathcal{V}$ is able to gain partial knowledge of the function σ, specifically $\sigma(1) = 3$ and $\sigma(3) = 2$. With it, $\mathcal{V}$ is able to determine all values in row 1 and row 3 for which $\sigma^{-1}(\sigma(v))$ is known. For this instance, that includes $\sigma^{-1}(v_{2,3}) = \sigma^{-1}(2) = 3$ and $\sigma^{-1}(v_{3,3}) = \sigma^{-1}(3) = 1$.

This effect of gaining knowledge avalanches per iteration rather quick until all values which are in the initial grid are filled in with all previously unknown cells to $\mathcal{V}'$. To consciously exploit this phenomenon even more, q amount of iterative steps can be gone through where for each iteration i the 2 challenges *initial values* and *row* i are queried, filling up the grid of known values in p row by row. After q iterations, the whole grid is filled with all known values present in p (given that the *initial values* challenge contains all q numbers, which is generally the case).

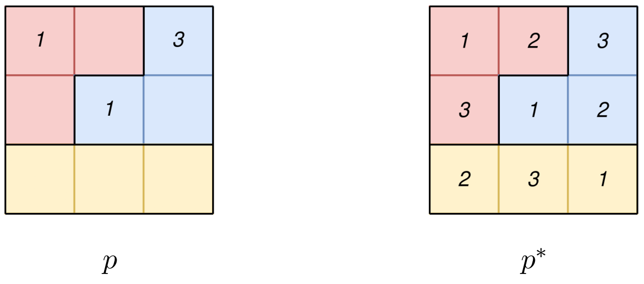
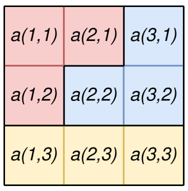


Figure A.1: Jigsaw Sudoku Example

$\mathcal{P}$	$\mathcal{V}'$
JS p, JS solution p^*	JS p,

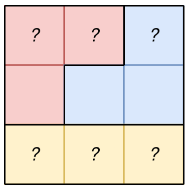
$\sigma : \{1,2,3\} \mapsto \{3,1,2\}$

$a \leftarrow$



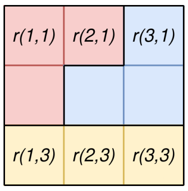
$\xrightarrow{a}$

$c \leftarrow$



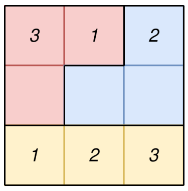
$\xleftarrow{c}$

$r \leftarrow$



$\xrightarrow{r}$

$\sigma(p^*)$ of r :=



$\sigma^{-1} : \{3,1,2\} \mapsto \{1,2,3\}$, as

$r(1,1) = 3 \mapsto p(1,1) = 1$

$r(3,1) = 1 \mapsto p(3,1) = 2$

fill in p :=

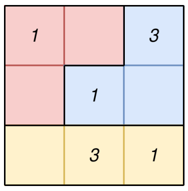


Figure A.2: Larger Challenge Space Leaks Knowledge

A.2 Protocol 1 - Example Protocol Runs

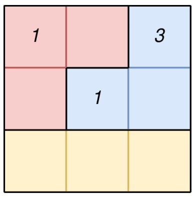
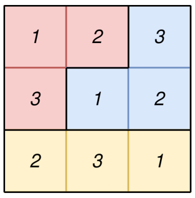
$\mathcal{P}$		$\mathcal{V}$
$''Hi\ I'm\ Prover'' \leftarrow$ "Hi I'm $Prover$. I have the solution to this 'really hard-to-solve' Jigsaw Sudoku and I want to prove that I know without revealing the solution."		
$p :=$ 		
$p^* :=$ 		
	$\xrightarrow{''Hi\ I'm\ Prover'',\ p}$	
		$''Hi\ I'm\ Verifier'' \leftarrow$ "Hi I'm $Verifier$. I would like for you to try. How are you planning to do this?"
	$\xleftarrow{''Hi,\ I'm\ Verifier''}$	
$''Explanation'' \leftarrow$ "$Prover$ explains the ZKP-protocol(**??**) and in particular proposes an agreement on a (arbitrary, but) widely trusted hash function h"		
	$\xrightarrow{''Explanation'',\ h}$	
	$\xleftarrow{''I Accept''}$	

Figure A.3: Σ - Protocol 1 for JS, Example Run, Initial Agreement

$\mathcal{P}$		$\mathcal{V}$
JS p, JS solution p^*, hash function h		JS p, hash function h
$\sigma : \{1,2,3\} \mapsto \{2,3,1\}$		
$s_{(1,1)} \overset{\$}{\leftarrow} s$		
$a(1,1) = h(\sigma(p^*(1,1))\|\|s_{(1,1)})$		
$= h(\sigma(1)\|\|s_{(1,1)})$		
$= h(2\|\|s_{(1,1)})$		
$s_{(1,2)} \overset{\$}{\leftarrow} s$		
$a(1,2) = h(\sigma(p^*(1,2))\|\|s_{(1,2)})$		
...		
$s_{(3,3)} \overset{\$}{\leftarrow} s$		
$a(3,3) = h(\sigma(p^*(3,3))\|\|s_{(3,3)})$		
$a = \{a(1,1), a(1,2), ..., a(3,3)\}$		
	$\xrightarrow{a}$	
		$c = \{(1,1),(1,2),(1,3)\}$
	$\xleftarrow{c}$	
$r \leftarrow \{(\sigma(p^*(1,1)), s_{(1,1)}),$		
$(\sigma(p^*(1,2)), s_{(1,2)}),$		
$(\sigma(p^*(1,3)), s_{(1,3)})\}$		
	$\xrightarrow{r}$	
		$a(1,1) \overset{?}{=} h(\sigma(p^*(1,1)))\|\|s_{(1,1)})$
		$a(1,2) \overset{?}{=} h(\sigma(p^*(1,2)))\|\|s_{(1,2)})$
		$a(1,3) \overset{?}{=} h(\sigma(p^*(1,3)))\|\|s_{(1,3)})$
		$\sigma(p^*(1,1)) \overset{?}{\neq} \sigma(p^*(1,2))$
		$\sigma(p^*(1,1)) \overset{?}{\neq} \sigma(p^*(1,3))$
		$\sigma(p^*(1,2)) \overset{?}{\neq} \sigma(p^*(1,3))$

Figure A.4: Σ - Protocol 1 for JS, Example Run, First Column

$\mathcal{P}$		$\mathcal{V}$
JS p, JS solution p^*, hash function h		JS p, hash function h
$\sigma : \{1,2,3\} \mapsto \{1,3,2\}$		
$s_{(1,1)} \overset{\$}{\leftarrow} s$		
$a(1,1) = h(\sigma(p^*(1,1))\|\|s_{(1,1)})$		
$= h(\sigma(1)\|\|s_{(1,1)})$		
$= h(1\|\|s_{(1,1)})$		
$s_{(1,2)} \overset{\$}{\leftarrow} s$		
$a(1,2) = h(\sigma(p^*(1,2))\|\|s_{(1,2)})$		
...		
$s_{(3,3)} \overset{\$}{\leftarrow} s$		
$a(3,3) = h(\sigma(p^*(3,3))\|\|s_{(3,3)})$		
$a = \{a(1,1), a(1,2), ..., a(3,3)\}$		
	$\xrightarrow{a}$	
		$c = \{(1,1),(3,1),(2,2)\}$
	$\xleftarrow{c}$	
$r \leftarrow \{(\sigma(p^*(1,1)), s_{(1,1)}),$		
$(\sigma(p^*(3,1)), s_{(3,1)}),$		
$(\sigma(p^*(2,2)), s_{(2,2)})\}$		
	$\xrightarrow{r}$	
		$a(1,1) \overset{?}{=} h(\sigma(p^*(1,1)))\|\|s_{(1,1)})$
		$a(3,1) \overset{?}{=} h(\sigma(p^*(3,1)))\|\|s_{(3,1)})$
		$a(2,2) \overset{?}{=} h(\sigma(p^*(2,2)))\|\|s_{(2,2)})$
		$\sigma(p^*(1,1)) \overset{?}{=} \sigma(p^*(3,1))$
		$\sigma(p^*(1,1)) \overset{?}{\neq} \sigma(p^*(2,2))$
		$\sigma(p^*(3,1)) \overset{?}{\neq} \sigma(p^*(2,2))$

Figure A.5: Σ - Protocol 1 for JS, Example Run, Initial Values

A.3 Protocol 3 - Example Transformation of Tents Before Commit

	2	1	0	2	1	1
0					T	
2	T					
0	T					
2			T			
1	T				T	
2			T			

p

	2	1	0	2	1	1
0					T	
2	T	-			-	
0	T					
2	-		T	-		
1	T				T	-
2	-		T	-		

p^*

Figure A.6: Tents Game

	0	2	1	0	2	1	1	0
0								
0						T		
2		T	-			-		
0		T						
2		-		T	-			
1		T				T	-	
2		-		T	-			
0								

Figure A.7: p^* with 1-cell border

	0	2	1	0	2	1	1	0
0								
0								
2			-			-		
0								
2		-			-			
1							-	
2		-			-			
0								

Figure A.8: p^* with 1-cell border, no trees

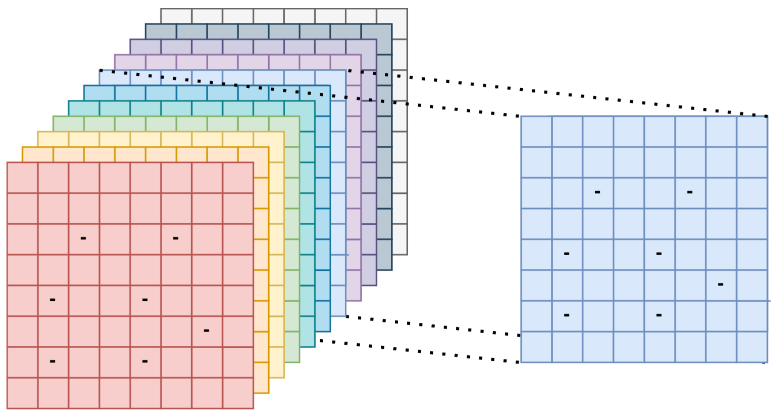


Figure A.9: p^*_{11}

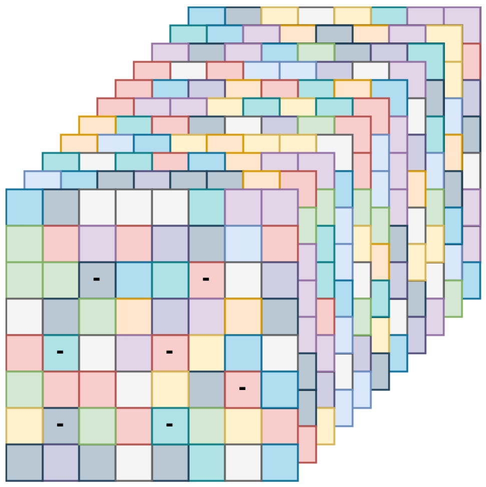


Figure A.10: $\sigma(p^*_{11})$

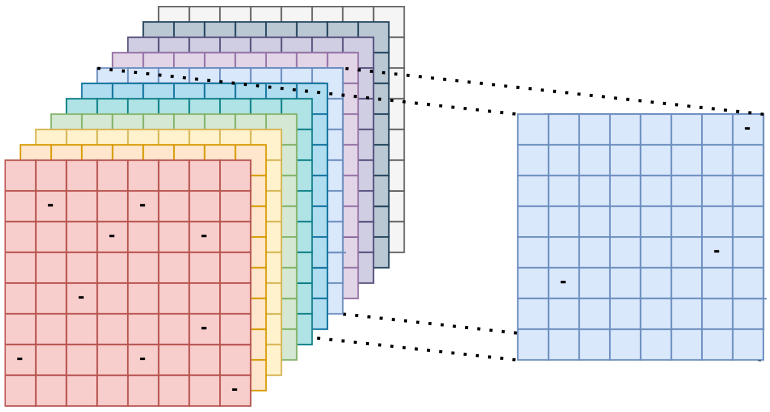


Figure A.11: reordered $\sigma(p^*_{11})$ on (permuted) indices