Zero Knowledge Proofs
Theory and Applications

Giacomo Fenzi
University of St. Andrews*

September 2019

*Thanks in particular to Prof. Stephen Linton and Dr. Louis Theran

Abstract

In this paper we will survey the theory underlying Zero Knowledge Proofs, developing the idea of interactive proving protocols, which will then be formalized first in Interactive Proofs and then in more complex knowledge withholding schemes, such as Perfect and Computational Zero Knowledge Proofs. In the process, we will have shown interactive proofs for Graph Non Isomorphism (a co-NP problem), and zero knowledge proofs for both Graph Isomorphism and Graph 3-Colorability. Finally, we will detail some applications of the theory, showing various "flavours" of ZKP for the Discrete Logarithm problem that will culminate in Schnorr Signatures. We will build on this to obtain the Secure Remote Protocol, a protocol that uses a similar ZKP in order to authenticate without communicating your password to the server. Finally, we show a very simple Zero Knowledge Range Proof, an instance of a very general class that has a wide range of applications and is currently one of the most active avenues of research. The paper will also introduce Sagemath worksheets that demonstrate nearly all of the protocols here detailed.

Declaration

Contents

1 Introduction

The concept of proof is one that resonates quite strongly in all of society. Having positive verification that an untrusted party is actually making an affirmative claim opens up to a variety of avenues for collaboration and enquiry. This paper will be a survey of the theory of a particular variety of proofs, namely Zero Knowledge Proofs. We will build an understanding of Interactive Proofs, which are processes in which a verifier is successively convinced by a prover. The fundamental idea is that the kind of proofs that we will consider are not fixed proofs as in Mathematics[1], but instead they are interactive processes, in which a verifier asks questions (or challenges) to the prover, and is progressively convinced by correct answers. The general problem that we look at is the following. Suppose that we have some object x such that the proposition $P(x)$ holds. How do we convince a skeptic of this fact? One way would be to just reveal x, but of course this might not be desirable for a variety of reasons. For example, it might be that x is a password, or contains other sensitive information. However, if $P(\cdot)$ has enough structure, we are able to devise a protocol in which the verifier can ask questions, that the prover can only consistently answer if it indeed has x, and from which the verifier learns nothing about such an x, except the fact that it exists. We will formalize such process, investigate the structure of the resulting class, and show some examples of protocols that satisfy said properties. Finally, we will survey some applications of Zero Knowledge Proofs, in the fields of digital signatures and authentication.

2 Structure

The structure of the dissertation will be as following. Section 3 will provide some motivation that show why the effort of developing Zero Knowledge Proofs is justified. Section 4 will cover the preliminaries, that includes both the mathematical machinery needed and the definition of the computational settings that we will be extensively using, namely that of Turing Machines. This Section will also introduce complexity classes that are relevant to our analysis, such a $\mathsf{P}, \mathsf{NP}, \mathsf{BPP}$, and introduce the concept of a proof in a computational sense. Section 5 will explore this concept in more depth, exploring via examples the characterization of Zero Knowledge Proofs, such as the general setup, soundness, completeness and the zero knowledge property. Section 6 will formalize the notion of proofs by introducing Interactive Turing Machine, and introduce the wide class of interactive proofs, of which Zero Knowledge Proofs are a subset. We will then give an example of an interactive proof for the Graph Non Isomorphism problem, complete with a proof of its correctness. In the following Section 7 we will finally introduce Zero Knowledge Proofs, in both the perfect and computational formulation. We will continue by providing an example of a Perfect Zero Knowledge Proof for deciding Graph Isomorphism, and conclude the section by introducing bit commitments, and showing that if one way permutations[2] exist then every language in NP has a Zero Knowledge Proof. This is done in Subsection 7.4 by providing a computational Zero Knowledge Proof for the NP-complete problem of Graph 3-colouring. In Section 8 we will then look at some applications, in three different contexts. First we will show how we can use a ZKP for the Discrete Logarithm in order to digitally sign a message (Schnorr signatures). Secondly, we will show how fundamentally similar proofs can be used in order to implement an authentication scheme in which the server

[1]They resemble mathematical proofs more closely when we consider Non Interactive Zero Knowledge Proofs

[2]In fact this holds for one way functions as well, but the permutation proof is simpler

never receives the user's password. Finally, we will show an extremely simple example of a Zero Knowledge Range proof, which is a strategy by which a party can prove that a certain value that it knows is within a suitable range, while maintaining anonymity to what that value actually is. This family of proofs have a far-reaching implication, as they allow to implement complex systems upholding invariants while maintaining anonymity, such as payment systems or government sanctioned age verification. Finally, Section 9 will refer the reader to the Sage implementation of the proofs described in the paper.

3 Motivation

Zero Knowledge Proofs have applications in a number of fields outside of theory. Examples vary from classical cryptography settings such as message signature and authentication to more practical ones such as online age verification and electronic voting. The development[3] of online blockchain based transaction systems has given indications of another area in which ZKP applications could emerge. In this section we will be focusing on this application, as it gives a good indication of a complex problem that ZKP can solve. Consider a blockchain system, such as Bitcoin[4]. As is well known, Bitcoin accounts (also known as wallets) are uniquely identified by an address, and this address is visible in every transaction that the account is involved in. In fact, knowing the ownership of an account allows anyone to reconstruct from the public ledger the entire transaction history, complete with transaction amounts. Furthermore, the balance of any address is publicly available. One direction of development is to allow for a blockchain system to be completely private, so that transactions can be done between actors while maintaining anonymity. Of course, many technical problems arise when trying to do so, but let us focus on one in particular. Suppose account A wants to transfer an amount X to an account B. What party B would like to verify is that A actually has enough balance to transfer such amount. However, since the system that we design is private, B does not have the ability to peer into A account or its balance. Instead, to solve this problem, we make it so that when A initiates the transaction, it produces a "proof" that its balance is greater than X. This proof has to have three properties. First of all, it must be convincing (which we refer to as complete), so that if A does indeed have enough funds, then B will be confident that the proof provided is correct. Secondly, it must be sound, so that if A does not have enough funds B will never be convinced that it does. Thirdly, it must be zero knowledge, ensuring it will not leak to B any information about A (like balance, account number et cetera) other than the fact that A has more than X in its balance. This can be done via what is called a Zero Knowledge Range Proof, of which we will see a very simple example in Section 8. One project implementing this in practice is Zcash [1].

[3] And the "hype" associated with it

[4] But not limited to, in fact, as far as I know, every major blockchain system works in essentially the same way

4 Preliminaries

4.1 Mathematical Notation

We start by defining some crucial sets of strings[5].

Definition 1. $\{0,1\}^n = \{a_1a_2\dots a_n \mid a_i \in \{0,1\}\}$

Definition 2. $\{0,1\}^* = \{a_1a_2\dots a_n \mid a_i \in \{0,1\} \text{ and } n \in \mathbb{N}\}$

It is important to stress that $\{0,1\}^*$ is the set of all *finite* strings, and also that the empty string, denoted ϵ.

Definition 3. *A **language** L is a subset of* $\{0,1\}^*$*, i.e.* $L \subseteq \{0,1\}^*$

Examples of languages include

$$\{x \in \{0,1\}^* \mid |x| = 2\}, \left\{\langle(x,y,z)\rangle \mid x^2 + y^2 = z^2\right\}$$

Note in particular the use of $\langle\cdot\rangle$ to denote the binary representation of the inner element. In general, as long as the element can be encoded in binary in polynomial space, we will not concern ourselves too much with said representation, and sometime even omit the brackets. In particular note that, unless specified otherwise, integers are encoded in binary and without brackets.

Now, we introduce a couple of utilities for talking about the asymptotic complexity of various functions.

Definition 4. *We define the following sets:*

$$\mathcal{O}(f) = \left\{g \mid \exists r \in \mathbb{R}^+, N \in \mathbb{N} \text{ s.t } \forall n \geq N : g(n) \leq rf(n)\right\}$$
$$\Omega(f) = \left\{g \mid \exists r \in \mathbb{R}^+, N \in \mathbb{N} \text{ s.t } \forall n \geq N : rg(n) \geq f(n)\right\}$$
$$\Theta(f) = \left\{g \mid \exists r_1, r_2 \in \mathbb{R}^+, N \in \mathbb{N} \text{ s.t } \forall n \geq N : r_1g(n) \leq f(n) \leq r_2g(n)\right\}$$

We also allow for two slight relaxation of notation, writing for example $\mathcal{O}(f)$ as $\mathcal{O}(f(n))$ and $f \in \mathcal{O}(g)$ as $f = \mathcal{O}(g)$. Furthermore, we define the following useful set:

Definition 5. *We define*

$$\mathsf{poly} = \bigcup_{i=0}^{\infty} \mathcal{O}\left(n^i\right)$$

as the set of all functions bounded above by a polynomial.

As before, we will allow a slight relaxation and use $\mathsf{poly}(\cdot)$ to specify some fixed but unspecified polynomial.

[5]In general, the use of the $\cdot^*$ operator, also known as the *Kleene Star* on a finite set of elements (also known as *alphabet*) generates the set of all finite sequences of elements of the original set.

Definition 6. *Let $\mu : \mathbb{N} \to \mathbb{R}$. We say μ is* ***negligible*** *if for every positive* $\mathsf{poly}(\cdot)$ *there exists a $N \in \mathbb{N}$ s.t for every $n > N$ we have:*

$$\mu(n) < \frac{1}{\mathsf{poly}(n)}$$

As an example, functions such as 2^{-n} or n^{-n} are negligible. In particular the set of negligible functions is closed under multiplication by elements of $\mathsf{poly}(n)$. This is significant, as we usually aim to have the attacker's success probability to be negligible, and this means that even repeating the attack polynomially many time will not create a non negligible threat.

There a couple of notions of probability that we will most likely need. First of all, we will use the notation $x \leftarrow\!\!\$\, A$ to say that x is uniformly selected from the set A[6]

Definition 7. *Suppose we have a series of n independent events X_i, each one of them occurring with probability θ. Then the total number of successes, which we denote as $X = \sum_i X_i$, follows a binomial distribution which means:*

$$\Pr[X = k] = \binom{n}{k} \theta^k (1-\theta)^{n-k}$$

In particular, we have that in those n trials we expect $\mathbb{E}[X] = n\theta$ events to occur. This is useful, especially since we have that the probability of an event occurring every time is $\Pr[X = n] = \theta^n$. This will be useful especially to estimate the probability of some algorithm succeeding multiple times when it is not supposed to. Note in particular that if an algorithm spuriously succeeds with probability $\theta \in [0, 1)$ then $\Pr[X = n]$ is a negligible function of n.

The following definition will be used for having a compact notation for selecting permutations.

Definition 8. *The group of permutations acting on X:*

$$S_X = \{f : X \to X| \text{ s.t. } f \text{ is a bijection}\}$$

Finally, a lot of our examples will deal with graphs and isomorphisms on graphs.

Definition 9. *A* ***graph*** *$G = (V, E)$ is a pair of sets, where V is referred to as the vertex set, and $E \subseteq \binom{V}{2}$ is the set of edges.*

Definition 10. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. A bijection $\phi : V_1 \to V_2$ is an* ***isomorphism*** *between G_1, G_2 iff*

$$(v, v') \in E_1 \implies (\phi(v), \phi(v')) \in E_2$$

In the case such an isomorphism exists, the two graphs are said to be isomorphic, and we write this as $G_1 \cong G_2$.

[6]Equivalently, when A finite, we have defined a random variable X such that for every $a \in A$ we have $\Pr[X = a] = \frac{1}{|A|}$

Finally, we introduce the following useful notation, for the graph obtained by permuting all vertexes of another graph.

Definition 11. *Let $G = (V, E)$, and $\pi \in S_V$, i.e. a permutation on the set V. Then we define $\pi(G) = (V, E')$ where:*

$$E' = \{(\pi(v), \pi(v'))|(v, v') \in E\}$$

Clearly, $G \cong \pi(G)$.

4.2 Computational Setting

The computational model used in this paper is that of Turing Machines (TMs)[2]. We will not discuss the inner definition of deterministic and non deterministic TMs, as, within the category, changes in definition do not create more than polynomial slowdowns, and as such for all intents and purposes can be considered the same[3].

In particular, we will be providing a uniform probability based definition for the complexity classes. The traditional (and equivalent) definitions are in Appendix C.

The Turing machine that we will be using have a slightly relaxed transition function, which allows for two different actions to be taken on the same step (as in a non deterministic machine). Also, for a polynomial time Turing Machine[7] M we will write $M_r(x)$ for the output of M on x, where r is a binary string, whose i-th character specifies which action the machine should take on step i. We can then let l be the maximum length that r can take and define the probability of M outputting y on input x to be:

Definition 12. *The probability of machine M outputting y on input x*

$$\Pr[M(x) = y] = \frac{\left|\{r \in \{0,1\}^l \mid M_r(x) = y\}\right|}{2^l}$$

Using this definition, we can then define our complexity classes.

Definition 13. *We say that a language $L \in \mathsf{P}$ if there exists a probabilistic polynomial time TM M such that:*

$$x \in L \iff \Pr[M(x) = 1] = 1$$

The complexity class P is the one we most closely associate with "efficient" computation. Notable examples of languages which belong in P are finite languages, regular languages, context free languages, and more. For example, the following are all in P :

$$\{(a_1, ...a_n) \mid n \in \mathbb{N} \text{ and } a_i \leq a_{i+1}\}$$

[7]The polynomial bound is required since in theory decisions taken early in the execution can effect the running time and such the length of the string. If the machine M is always polynomial time we can then find a bound p s.t. $\forall r, T(M_r(x))) \leq p(|x|)$, and as such having $r \in \{0,1\}^{p(|x|)}$ will be enough

$$\{p \,|\, p \in \mathbb{N} \text{ and } p \text{ prime } \}^{8}$$
$$\left\{(a,b,c) \,|\, a^2+b^2=c^2\right\}$$

In general, a language being in P implies there exists a machine such that all the computation paths recognize the input if the input is indeed in the language.

Compare this to the definition of NP.

Definition 14. *We say that a language $L \in \mathsf{NP}$ if there exists a probabilistic polynomial time TM M such that:*

$$x \in L \iff \Pr[M(x)=1] > 0$$

The definition implies that there is at least one path for which the input is accepted. It is easy to see that $\mathsf{P} \subseteq \mathsf{NP}$, but the question of $\mathsf{P} =_? \mathsf{NP}$ is still open [5] [6]. There are many NP problems that we believe do not belong in P, such as the following:

$$\texttt{INT\_FACT} = \{(n,k) \,|\, \exists f : 1 < f < k \text{ and } f \text{ divides } n\}$$
$$\texttt{SUB\_SUM} = \left\{ G \subset \mathbb{Z} \,|\, |G| < \infty \text{ and } \exists S \subseteq G : \sum_{i \in S} i = 0 \right\}$$
$$\texttt{SAT} = \{\phi : \{0,1\}^n \to \{0,1\} \,|\, n \in \mathbb{N} \text{ and } \exists a_1 \dots a_n : \phi(a_1, \dots a_n) = 1\}$$
$$\texttt{GRAPH\_HOM} = \{(G,H) \,|\, \exists \phi : G \to H \text{ s.t. } \phi \text{ is a graph homomorphism}\}$$

In particular, the last three languages are instances of NP-complete problems, i.e. problems that are conceptually as hard as any problem in NP (more on this in Appendix C). It is interesting to see that each of these problems involve some sort of search over an exponentially large possibility space, and the correctness can be easily verified by the result of such search. We associate thus NP class with problems that can be efficiently verified.

Definition 15. *We say that a language $L \in \mathsf{BPP}$ if there exists a probabilistic polynomial time TM M such that the following hold*

1. $x \in L \iff \Pr[M(x)=1] \geq \frac{2}{3}$
2. $x \notin L \iff \Pr[M(x)=0] \geq \frac{2}{3}$

In this case we say that L is ***recognizable in probabilistic polynomial time***

Note in particular that we can replace[9] the $\frac{2}{3}$ constant with $\frac{1}{2}+\epsilon$ for any $0 < \epsilon < \frac{1}{2}$.

BPP is referred to as the class of bounded polynomial time languages. It is evident that $\mathsf{P} \subseteq \mathsf{BPP}$ but, while it is believed that the classes are equal [7], no proof was yet found. This class is significant, as it is intrinsically associated with efficient computation. This is because of a few factors, first of all because

[8]This can be done via AKS Primality Testing [4]

[9]By running the machine multiples times and taking majority vote

the probabilistic computation closely mimics the capabilities of general purpose computer, which can use the external environment as a source of randomness. Secondly, polynomial time computation is feasible, and, as long as Moore's law holds, modern computer can every year improve faster than one can increase the problem size[10]. Furthermore, the error probability of a probabilistic algorithm can be made exponentially small by repeating the algorithm polynomially many times. This means that if an algorithm has a $\frac{1}{c}$ probability of failing, repeating it n times will yield an algorithm that fails with probability c^{-n}. Let us have an example to clarify how a probabilistic algorithm works:

```
Matrix Multiplication Probabilistic Verification
Inputs: $A \in \mathbb{F}^{n \times m}, B \in \mathbb{F}^{n \times k}, C \in \mathbb{F}^{k \times m}$
$x \leftarrow_{\$} \mathbb{F}^m$
return $Ax =_? BCx$
```

Figure 1: Algorithm for checking Matrix Multiplication probabilistically

Example 1. *Consider three matrices, A, B, C where $A \in \mathbb{F}^{n \times m}$, $B \in \mathbb{F}^{n \times k}$, $C \in \mathbb{F}^{k \times m}$. We want to solve the question $A =_? BC$. Of course, in a deterministic setting, we can simply multiply the two matrices (taking $\mathcal{O}(nkm)$ steps) and compare it with the first (in $\mathcal{O}(nm)$ steps), in a total of $\mathcal{O}(nmk + nm)$ steps. We can do better if we allow for some probabilistic error. Take the following procedure (also in Figure 1) :*

1. *Take a random vector $x \in \mathbb{F}^m$*
2. *Compute Ax and $B(Cx)$.*
3. *Compare the two resulting vectors (in $\mathbb{F}^n$). If they are the same return true, else return false.*

Computationally wise, this might require less calculations. Let's assume you can select a random k-vector in $\mathcal{O}(k)$ steps, then the first step is $\mathcal{O}(m)$, the two computations take respectively $\mathcal{O}(nm)$ and $\mathcal{O}(k(n+m))$[11] and the comparison will be $\mathcal{O}(n)$. So in total the algorithm is $\mathcal{O}(nm + kn + mk + n)$. We claim that, trivially, if $A = BC$ then the algorithm always answer correctly. Furthermore if $A \neq BC$ then the algorithm answers correctly in the majority of cases. Let $D = BC$ Note that if the algorithm is to output incorrectly, it

[10] What I mean is, if an algorithm takes $\mathsf{poly}(|x|)$ steps, and on year n a computer can perform $\mathcal{O}(2^n)$ steps, then by waiting a suitable and polynomial number of years the time needed decreases exponentially. E.g., suppose that a given problem takes $|x|^3$ steps to be solved, and on year 0 a computer can perform 1000 steps per year. Then in year 0 we can solve instances of size at most $|x| = 10$. However, suppose in year n the power of the computer is given by $1000 \cdot 2^n$. This implies that in year n we can handle instances of size $|x| = 10 \cdot 2^{n/3}$, which is exponential in n

[11] This is optimized computing $B(Cx)$

must be that $Ax = Dx$. Then it must be that $(A - D)x = 0$, and as such $x \in \ker(A - D)$. Note that $A - D \in \mathbb{F}^{n \times m}$ and as such $A - D$ can be seen as a linear transformation from $\mathbb{F}^m \to \mathbb{F}^n$. The whole problem reduces to finding the probability that a vector picked at random belongs to the kernel of a linear transformation. Let $|\mathbb{F}| = q$, the size of the underlying field[12]*. First of all note that since $A \neq D$ at least one of the of the row of the matrix is non zero, and as such $\dim \ker(A - D) < m$. Since the size of any vector space V over $\mathbb{F}$ is $|V| = q^{\dim V}$, then the size of the kernel is at most q^{m-1}. As such the probability of a random vector belonging to the kernel is at most $\frac{q^{m-1}}{q^m} = \frac{1}{q}$. We can then calculate that, repeating the algorithm l times the probability of getting at least one vector not in the kernel is $1 - \left(\frac{1}{q}\right)^l$ and if we want to ensure that this probability is greater than $\frac{2}{3}$ we can then set:*

$$l > \frac{\log \frac{1}{3}}{\log \frac{1}{q}} = \frac{\log 3}{\log q}$$

Finally, we can see that repeating the procedure l times[13] *(call this algorithm M) will have the following result:*

- $A = BC \implies \Pr[M(A, B, C) = 1] = 1$
- $A \neq BC \implies \Pr[M(A, B, C) = 0] > \frac{2}{3}$

Note that that brings the final complexity to $\mathcal{O}(l(nm + kn + mk + n))$.

4.3 Adversarial Settings

In cryptography, in order to achieve sound schemes, it is common to work in what is commonly referred to as an adversarial setting. Intuitively, while aiming to prove the security of a cryptography protocol, it is not sufficient to prove the construction secure from a particular subset of attacks. Instead it is necessary for it to be secure in the wake of *any* strategy that a sufficiently capable attacker might take.

The capability of the attacker will vary depending on the level of security that the protocol needs to be held to. For example, the computational power that the adversary has will affect the class of problems that it can solve. As an instance, if a construction relies on (hard) instances of an NP-complete problem (and if $\mathsf{P} \neq \mathsf{NP}$) then a deterministic polynomial time attacker will not be able to break the scheme, while one with non-deterministic polynomial resources will be able to[14].

[12]Note that in this case we assume the underlying field to be finite, the infinite case can also be handled but requires more thought

[13]In fact, since $l < 1$ for $q > 3$, repetition is only really needed for the trivial field $\mathbb{Z}_2$

[14]For the sake of the argument, say this problem is a decision problem, say L. Then, since $L \in \mathsf{NP}$ there exists a deterministically recognizable $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$ s.t. $x \in L \iff \exists y$ s.t. $(x, y) \in R_L$ and $|y| = \mathsf{poly}(|x|)$. So a NDTM can, to solve $x \in_? L$ guess y in a polynomial number of non deterministic steps, and check if (x, y) belongs in R_L

In general, the most strict level of security that we might require is that of information theoretic security, which holds even in the case that an attacker has infinite computational power. We will see that Perfect Zero Knowledge Proofs have an information theoretic flavour, even though saying that they are secure against an infinitely powerful attacker might be a stretch[15]. After that level, we usually require that the attacker has computational capabilities that are bounded by a polynomial. This quite closely mirrors the real world requirements for most schemes [8]. We will see that Computational Zero Knowledge Proofs have this kind of protection, as they rely on the concept of computational indistinguishability.

4.4 Notion of Proof

The notion of a proof has extremely overloaded meaning, as it used in many disparate fields. In particular, the notion of a proof in Mathematics most closely comes to mind and, while mathematical proofs and ZKP ultimately aim to ascertain the validity of a statement, the way that they achieve that result varies. In formal logic, a proof is a fixed sequence of steps (derivations) that, from commonly agreed statement (axioms), reach the statement that we are wanting to prove. In contrast, in a ZKP the process is a dynamic interaction between multiple parties, more akin to the notion of proof in law or debating.

5 Proof Requirements

In order to start visualizing the concept of ZKP, we can start with a couple of toy examples that are helpful. Here and in the rest of the text we will refer to the prover as Peggy and to the verifier as Victor.

Example 2. *Assume that Victor is completely colourblind*[16]*. Peggy wants to prove to Victor that she is not colourblind, so she devises an experiment. Peggy takes two balls completely identical, save for their colour, and gives them to Victor, one in his left and one in his right hand. Peggy asks Victor to put the balls behind his back, and to decide whether to swap them or not, without telling her. She then asks Victor to show the two balls. Peggy concludes by telling Victor whether he swapped the balls or not.*
This is a zero knowledge proof in the sense that:

[15]What I mean is that, in perfect ZKP the output of the interaction can be perfectly simulated by polynomial time prover, which is an information theoretic level of accuracy. However, any ZKP usually deals with an underlying language that is only interesting when it is not decidable in polynomial time. For example, Graph Isomorphism is one of such problems. While we can devise a perfect ZKP for this problem, this does not mean that an unbounded computational attacker cannot decide the problem. It merely means that even a perfect attacker cannot understand whether it is in a simulation or a real interaction. On a less formal note though, we can see that the zero knowledge property still holds, as an unbounded attacker in a sense already has all of the knowledge, and as such it cannot gain any more!

[16]i.e. that he cannot distinguish object based on colour alone

1. *If Peggy were colourblind, she would not be able to tell whether Victor switched the balls or not, and so the best she could do is guess, getting it right $\frac{1}{2}$ of the time*[17]*.*

2. *If Peggy is not colourblind, she will be able to answer correctly every time, and thus convince Victor*

3. *Victor does not learn anything else about his environment, apart from the fact that Peggy is not colourblind.*

Example 3. *This example is due to Jean-Jacques Quisquater [9]. Consider a cave shaped like a ring, with a single entrance. On the side opposite to the entrance, there is a gate that divides the cave into two. The gate can only be opened by a secret word, that only Peggy knows. Peggy wants to prove to Victor that she possesses this secret word, without of course letting him in on the secret. She devises an experiment as follows. She tells Victor to wait at the entrance of the cave, and she goes by one of the two possible paths (without letting Victor know which one she took) to the gate, where she can utter the secret word without being heard. Then she asks Victor to name one of the two paths, and she goes back trough the named path. As before:*

1. *If Peggy did not know the secret word, she would not be able to go back on a path different from the one she initially took, and as such has only a $\frac{1}{2}$ chance of getting it right.*

2. *Instead, if Peggy does indeed know the secret word, she will be able to always take the path that Victor names, and as such convince him*

3. *Furthermore, Victor does not gain any knowledge of the secret key, apart from the fact that Peggy knows it.*

This illustrates the general idea of Zero Knowledge Proofs. The proof part is devised as an experiment or challenge from the verifier to the prover, that the prover can only answer by having the necessary piece of information or "power". The prover cannot fool the verifier consistently and an honest prover and honest verifier will agree on the fact. Furthermore no new information is learned by the verifier. Also, if external observers were to have witnessed the exchange they would not be convinced of the fact (as Peggy and Victor could have coordinated their answers). Note however that in this case a simpler protocol can actually fulfil the requirement. For example, Peggy could go into the cave, showing Victor which entrance she took, and come out from the other entrance. This in effect amounts to removing the need of an interaction between Peggy and Victor, which is something that is quite desirable in real world applications. In particular we will see this in Section 8, with regards to Schnorr signatures. Note that this transformation also removes the probability aspect, which instead is something that in general we cannot do for non trivial languages, but I digress.

We seek to formalize the ideas from these two examples next.

[17]And as such repeating the experiment n times will bring her probability of always guessing right to $\frac{1}{2^n}$

5.1 Parties in play

In the sequel, we will have two parties interacting, the *prover* and the *verifier* that, as before, will be colloquially referred to as Peggy and Victor. In mathematical notation we will have P, V always referring, respectively, to prover and verifier. As in maths, we aim to have the verification procedure be as efficient as possible, while most of the computational burden will be placed on the prover. This mirrors the definition of NP we used above, as the class of all problems whose solution can be efficiently verified. Furthermore, it is crucial to understand that the verifier inherently does not trust the prover (otherwise there would not be need for the proof to be provided), and as such Victor will be skeptical of everything that Peggy says. The general situation will be Peggy and Victor having access to some common input, plus each one possibly having access to some additional private input, which will often be related from the shared input.

5.2 Soundness

The condition of soundness asserts that an honest verifier cannot be tricked into accepting a false statement by a possibly cheating prover. In the first example, this is equivalent to stating that, if Peggy is indeed colourblind, then she cannot convince Victor that she is not. In the second one, if Peggy were not to know the password to the gate, then she would not be able to convince Victor who will then not be fooled. It is important to note that in the above example we actually allow for a dishonest prover to be able to successfully fool Victor with a bounded above probability[18].

5.3 Completeness

Conversely, the condition of completeness states that an honest prover will be able to convince an honest verifier of the veracity of the claim, if the claim is indeed true. So, moving back to the first example, Peggy will be able to prove she is not colourblind if that is indeed true and Victor acts honestly satisfying the protocol. While in the examples Victor is always positively convinced at the end of the interaction, this is not a strict requirement, instead (similarly as in Soundness) we just require the interaction to succeed with a strictly bounded below probability[19]. As an example of when this might be the case, consider the following

Example 4. *Peggy claims that coffee tastes differently when brewed by an espresso machine compared to one made in a cup. Victor wants to verify that claim, so he brews a coffee choosing randomly between the two options (in way that Peggy does not see which is which). Peggy then tastes the coffee, and gives*

[18]And this probability can be made negligible thanks to repeating the experiment multiple times

[19]And again this probability can be increased by repeating the experiment and taking the majority answer

her opinion on how it was brewed. The experiment is repeated n times[20]*, and Victor is convinced if Peggy is right more than c times, where c depends on how certain we want to be of Peggy's ability.*

This is the exact subject of hypothesis testing, in particular it reduces to the exact problem of finding whether a coin is biased. This can be modelled by a binomial distribution with mean $\mu = n/2$ and standard deviation $\sigma = \frac{\sqrt{n}}{2}$. In particular, if we apply the central limit theorem (and as such have $n > 30$) we can model it as a normal distribution, then using the 3σ rule[21] we can see that if $|c - \frac{n}{2}| \geq 3\frac{\sqrt{n}}{2}$ then we can affirm that with a more than 99% chance Peggy can distinguish the two. For example, if we set $n = 100$ and Peggy guesses right more than $c = 65 = \frac{100}{2} + 3\frac{\sqrt{100}}{2}$ times (or less than 35 times) then we can be almost certain that Peggy is truthful (and in the other case that she can distinguish it but for some reasons she is convinced the espresso machine coffee is brewed in cup, and vice versa).

5.4 Zero Knowledge

Formalizing the notion of zero knowledge requires a bit more work. We would like to be able to say that in any interaction with the prover the verifier does not gain any knowledge that it did not posses already. That shifts the conversation to what it means to gain knowledge. Let us consider for example the interaction of a prover and a verifier with a common input of a suitably large graph. If the prover reveals to the verifier whether the graph is connected or Eulerian or the average degree of its vertexes then in a sense the verifier does not gain any knowledge, as these are all easily (polynomial time) computable without any additional information. Instead if the prover reveals whether the graph is Hamiltonian, or the chromatic number of the graph, or a k-colouring of it[22] then the verifier gains knowledge, as it would have not been able to answer that question itself using an efficient procedure. This gives us a hint of what gaining knowledge can entail. In particular, we say that the verifier gains knowledge from an interaction with the prover if, after the interaction, it can efficiently compute something which it would have not been able to do before. How to rigorously express this is non trivial, and we defer the discussion to the following sections.

6 Proof Systems

In our drive to formalize the intuition that we gained in the previous sections, we will start by extending the model of a Turing Machine to allow for two Turing

[20]According to [10], and assuming Peggy weighs around 70kg, we would not recommend for n to be greater than 75-100

[21]i.e. that three standard deviations include 99.7% of the data

[22]And of course assuming $\mathsf{P} \neq \mathsf{NP}$ and that the corresponding problems cannot be solved in polynomial time

Machines to interact. Then we will define formally interactive proofs and give examples.

6.1 Interactive Turing Machine

First of all, we can start by extending the traditional randomized Turing Machine. It is of note that in interactive Turing Machines (ITMs) we are almost never interested in the execution of a single machine, but rather in the interconnected operation of the two machines. As such, it makes it easier to describe the behaviour of a pair of ITMs rather than of a single one. First of all, with this model we aim to preserve the following notions:

- Exclusivity, so that, before termination, exactly one of the two machines is executing at any point in time
- Privacy, so each of the two machines can withhold some information from the other[23]
- Communication, each of the two machines can choose to let some information (often related to their private information) to the other, using communication tapes

To fulfil those goals, we define a pair of interactive machines as follows:

Definition 16. *A pair of interactive Turing Machines A, B have the following characteristics:*

- *Each machine is a randomized Turing machine in its own right i.e. it has an input tape, an output tape, and a random tape*[24].
- *Neither machine can access the other's aforementioned tapes.*
- *The system has two additional tapes, called communications tapes, T_1, T_2. A can read from T_1 and write to T_2, while B can write to T_1 and read from T_2.*
- *Each machine has an access to a number of additional switch states, δ_S, where S is a state of the machine in question. If a transition sets the state to one of these δ-states, then the current machine becomes idle, and execution is resumed on the other machine. When execution is resumed on a machine, the state it is in depends on the last suspension. If it was suspended in state δ_S, then it will resume execution in state S. Of course, if no suspension occurred, the machine will start from the initial state.*

[23]In particular we require that each machine is able to keep the contents of its random tape private. There are some cases in which this requirement might be lifted, in so called public coin protocols, but we will not discuss them here

[24]This is an alternate equivalent definition of randomized Turing Machines, it just makes the explanation easier

- *The interconnected randomized Turing machine pair terminates when either of the two machines terminates.*
- *An interactive proof system runs on some common input, which is initially written on both machines' input tape.*
- *By convention, A is initially running, and B is idle.*

It is to be noted that there is an alternative and ultimately equivalent model that allows each machine to have an additional private input tape, which allows for both machines to be polynomial time. For simplicity here we work with the definition with less tapes, but it is good to be aware of it.

Using the above definition, we can go and define notation for the output of an execution of a pair of ITMs.

Definition 17. *Let A, B be interconnected randomized Turing machines, such that both A and B always terminate in finitely many steps. Then we define $\langle A, B\rangle(x)$ to be a random variable modelling the output of machine B after interacting with machine A on common input x (with the computation path string uniformly and independently selected).*

Note that the definition is asymmetric, as it only accounts for the output of B, however, for all our intents or purposes, this should suffice. Furthermore, we need to accurately represent time complexity in this model, and we say that:

Definition 18. *An ITM A has time complexity $t : \mathbb{N} \to \mathbb{N}$ if, for any string x and every linked ITM B, and any computation path, it halts within $t(|x|)$ steps.*

In a nutshell, the above implies that, regardless of the messages that machine B sends, machine A always terminates quickly enough.

6.2 Interactive Proof Systems

With the above machinery, we can now start introducing Interactive Proofs[11].

Definition 19. *A pair of ITMs (P, V) are an **interactive proof system** for a language L if V is polynomial time and the following hold:*

- ***Soundness** if $x \notin L$, for every ITM E:*

$$\Pr[\langle E, V\rangle(x) = 1] \leq \frac{1}{3}$$

- ***Completeness** if $x \in L$ then:*

$$\Pr[\langle P, V\rangle(x) = 1] \geq \frac{2}{3}$$

There are some things to note in the above definition. First of all, note that only the verifier is required to be computationally bound, while the prover has no such limit. This is required in the general case, as we will see in the next example, but often in the practical application we can make the prover polynomially bound by resorting to a similar model which makes use of auxiliary input. Secondly, as in many other examples, the bounds of $\frac{1}{3}$, $\frac{2}{3}$ can be replaced[25] by $\frac{1}{2} \mp \epsilon$ for $\frac{1}{2} > \epsilon > 0$. From the above definition, we can prove that any language in NP has an interactive proof system. Letting IP be the set of all languages with an interactive proof system:

Theorem 1. $\mathsf{NP} \subseteq \mathsf{IP}$

Proof. If L is in NP then[26] there exists a polynomially recognizable relation R_L such that $x \in L \iff \exists y : (x, y) \in R_L$. Also, such a y satisfies $|y| < p(|x|)$ for some polynomial p.
We design the following interactive system, which has as common input x. First of all, the prover finds such y. This can be done by searching for any string s in $\bigcup_{i=0}^{p(|x|)} \{0, 1\}^i$ that satisfies $(x, s) \in R_L$. Since the prover has no computational bound this can be done without issues. The prover then sends this s to the verifier. On a message s the verifier accepts if $(x, s) \in R_L$, rejects otherwise. Since R_L is recognizable in polynomial time, the verifier runs in polynomial time as required.
Completeness is easily verified, as in fact the verifier will accept with probability 1. Soundness requires a little more thought, but essentially since $x \notin L \implies \forall y : (x, y) \notin R_L$ then the verifier will never accept (i.e. accept with probability 0). This implies that the above is an interactive proof system for L. Since L was an arbitrary element of NP, then $\mathsf{NP} \subseteq \mathsf{IP}$. ☐

In fact, as we will see next, IP actually contains languages that are not believed to be in NP. We will not prove this, but it turns out that $\mathsf{IP} = \mathsf{PSPACE}$[12], i.e. the set of all languages that use polynomial space[27]. Also note that in the case of NP problems we can even make the prover machine polynomial time, if we use the additional input model and give the prover the certificate.

6.3 An Example: GNI

Let us consider the Graph Non Isomorphism problem, a decision problem which is not known to be in BPP or in NP. This quite aptly named problem concerns deciding whether two graphs are not isomorphic. Before presenting the system, let us try to give some intuition onto how we will be approaching the problem. As in the coffee problem that we examined in the completeness section, we will have the verifier issuing challenges to the prover, which it will use its superior computational abilities to solve. In particular, we rely on the fact if the two graphs are not isomorphic then it will always be possible to distinguish the two. Here is a interactive proof system that decides this problem [13].

[25]By essentially repeating the procedure and taking majority votes
[26]Here we are using the certificate definition of NP, see Appendix C for details
[27]And this class is considered to be much bigger than NP

Proof that G_1, G_2 are not isomorphic

Prover		**Verifier**
	$G_1 = (V_1, E_1), G_2 = (V_2, E_2)$	
		$i \leftarrow_{\$} \{1, 2\}$
		$\pi \leftarrow_{\$} S_{V_i}$
		$F = \pi(G_i)$
	$\xleftarrow{\quad F \quad}$	
Find j s.t. $G_j \cong F$		
	$\xrightarrow{\quad j \quad}$	
		return $i =_? j$

Figure 2: An interactive proof for Graph Non Isomorphism

Example 5. *Let the common input be a pair of graphs, G_1, G_2 such that $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.*

- *The verifier starts the interaction, uniformly selecting at random $i \in \{1, 2\}$ and then randomly selects a permutation $\pi \in S_{V_i}$. It then sends the graph $F = \pi(G_i)$ to the prover.*
- *The prover finds which of the two input graphs is isomorphic to F, and sends the answer j to the verifier*
- *If $i = j$ the verifier accepts (concludes that the two graphs are not isomorphic), else it rejects (is not convinced that the graphs are not isomorphic).*

We now set to prove that the above is an interactive system proof. First of all note that the verifier program can be implemented in probabilistic polynomial time. We do not know of any algorithm allowing the prover to be implemented in probabilistic polynomial time, but luckily that is not needed[28]. Now let us check the two conditions of soundness and completeness.

Theorem 2. *If G_1 and G_2 are not isomorphic, the above procedure always accepts.*

Proof. If $(G_1, G_2) \in$ `GNI` then there does not exist a graph F s.t. $G_1 \cong F \cong G_2$, (since otherwise $G_1 \cong G_2$ by transitivity). By construction then the graph F constructed by the verifier will always be isomorphic to only one of the two

[28]The prover can do this by alternatively trying each of the $n!$ possible permutations on V_1 (if $n = |V_1|$) and checking for isomorphism, outputting 1 if one found and 2 if none (of course this relies on the verifier being honest which for completeness and soundness is always the case).

input graphs, and as such the prover always has an unique choice of j, which must equal the i that the verifier chose. As such the verifier always accepts and the completeness bound is 1. $\square$

Intuitively, for the soundness bound, we claim that a possibly malicious prover cannot convince the verifier that the graphs are isomorphic while indeed they are not. In particular, we claim that in case they are isomorphic, the best the prover can do is guess what the verifier chose, with probability $\frac{1}{2}$.

We start by proving this preliminary claim:

Theorem 3. *Let I be a random variable uniformly selected from $\{1, 2\}$. Let G_1, G_2 be two graphs such that $G_1 \cong G_2$. We also let Π be a random variable uniformly distributed over the set of permutations of the vertex set of G_1 (which without loss of generality we assume being equal to the vertex set of G_2). Then we want to show, for every graph $F \cong G_1 \cong G_2$ the following hold:*

$$\Pr[I = 1 \mid \Pi(G_I) = F] = \Pr[I = 2 \mid \Pi(G_I) = F] = \frac{1}{2}$$

Proof. First of all let F be a graph isomorphic to G_1, G_2. Then consider the sets S_1, S_2 defined by $S_\delta = \{\pi : \pi(G_\delta) = F\}$. We first claim that $|S_1| = |S_2|$. To see this, note that $G_1 \cong G_2 \implies \exists \phi : G_1 = \phi(G_2)$. Then $\pi \in S_1 \implies \pi(G_1) = F \implies \pi(\phi(G_2)) = F \implies \pi \circ \phi \in S_2$, where the last step is justified since $\alpha(\beta(G)) = (\alpha \circ \beta)(G)$. So we can see that $f : S_1 \to S_2$ defined as $f(\sigma) = \sigma \circ \phi$ is well defined, and we can prove that it is a bijection[29]. and as such $|S_1| = |S_2|$. Now using this fact we have that:

$$\begin{aligned}
&\Pr[I = 1 \mid \Pi(G_I) = F] \\
&= \Pr[\Pi(G_1) = F] \\
&= \Pr[\Pi \in S_1] \\
&= \Pr[\Pi \in S_2] \\
&= \Pr[\Pi(G_2) = F] \\
&= \Pr[I = 2 \mid \Pi(G_I) = F]
\end{aligned}$$

Now using Bayes' rule[30]:

$$\Pr[I = 1 \mid \Pi(G_I) = F] = \frac{\Pr[\Pi(G_I) = F \mid I = 1] \Pr[I = 1]}{\Pr[\Pi(G_I) = F]} = \Pr[I = 1] = \frac{1}{2}$$

$\square$

Using this, proving the soundness property is not too hard.

[29]Injectivity: $f(\sigma) = f(\sigma') \implies \sigma\phi = \sigma'\phi \implies \sigma = \sigma'$. Surjectivity: $\pi \in S_2 \implies \pi(G_2) = F \implies \pi(\phi^{-1}(G_1)) = F \implies \pi \circ \phi^{-1} \in S_1$. Then $f(\pi \circ \phi^{-1}) = \pi \circ \phi^{-1} \circ \phi = \pi$

[30]And using the facts that $\Pr[\Pi(G_I) = F \mid I = 1] = \Pr[\Pi(G_1) = F] = \Pr[\Pi \in S_1]$ and that $\Pr[\Pi(G_I) = F] = \frac{\Pr[\Pi(G_I) = F \mid I = 1] + \Pr[\Pi(G_I) = F \mid I = 2]}{2} = \frac{1}{2}(\Pr[\Pi \in S_1] + \Pr[\Pi \in S_2])) = \Pr[\Pi \in S_1]$

Theorem 4. *Let V be the verifier in the above protocol, then for any prover E and any $G_1 \cong G_2$ we have*

$$\Pr[\langle E, V\rangle (G_1, G_2) = 1] \leq \frac{1}{2}$$

Proof. We see that, in the protocol, the verifier only ever accepts if the prover successfully figures out which graph was permuted and sent to it. Using the notation of the above discussion, we have that the permuted graph is represented by the random variable $\Pi(G_I)$. Letting E be a random process, we see that the verifier accepts if $E(\Pi(G_I)) = I$. Then:

$$\Pr[E(\Pi(G_i)) = I] = \sum_{G'} \Pr[\Pi(G_I) = G'] \Pr[E(G') = I \mid \Pi(G_I) = G']$$

We now can use the proof from before and conclude that, for any G'

$$\begin{aligned}\Pr[E(G') = I \mid \Pi(G_I) = G'] &= \sum_i \Pr[E(G') = i \,\wedge\, I = i \mid \Pi(G_I) = G'] \\ &= \sum_i \Pr[E(G') = i] \Pr[I = i \mid \Pi(G_I) = G']\end{aligned}$$

Here we can use the fact proved before, and for $i \in \{1, 2\}$ we have that the expression on the right is equal to $\frac{1}{2}$ and as such

$$\begin{aligned}&\sum_i \Pr[E(G') = i] \Pr[I = i] \,\Pi(G_I) = G' \\ &= \frac{\Pr[E(G') = 1] + \Pr[E(G') = 2]}{2} \\ &= \frac{\Pr[E(G') \in \{1, 2\}]}{2} \\ &\leq \frac{1}{2}\end{aligned}$$

Where the last inequality is since the probability is maximized when E always outputs an element of $\{1, 2\}$. Going back to the original equation:

$$\Pr[E(\Pi(G_i)) = I] \leq \frac{1}{2} \sum_{G'} \Pr[\Pi(G_I) = G'] = \frac{1}{2}$$

Since E was arbitrary, this implies that any possible prover can fool the verifier with maximum probability $\frac{1}{2}$. □

This shows that the soundness bound of the protocol is $\frac{1}{2}$, and repeating the protocol twice yields an interactive proof with completeness bound 1 and soundness bound $\frac{1}{4}$.

7 Zero Knowledge Proof

Now that we have introduced interactive proofs, we can naturally extend them to encompass the idea of zero knowledge. In the previous sections we have given some intuition into what gaining knowledge does mean, and now we aim to formalize this. As before, we consider an interactive proof system, composed of a prover and a verifier, respectively P, V. The first observation is that in an interactive proof system the prover usually (either by being more powerful computationally or by having some additional information) has a "knowledge advantage" over the verifier. With zero knowledge we aim to make so that this advantage does not transfer to the verifier, no matter what clever things it may do. From this we can conclude that being zero knowledge is strictly a property of the prover in the interaction, in the same way that soundness is a property of verifier. Secondly, in a sense, we can express the zero knowledge requirement as the fact that any verifier V', when interacting with prover on input x, will not be able to compute anything that it wouldn't have been able to compute on input x alone. In order to formalize this, we turn to the idea of a simulation of the interaction of P, V'.
Let us consider why this might be an helpful notion. Our loose definition of Zero Knowledge as computational power gain in effect means that any prover that is polynomial cannot yield any knowledge to a verifier, as the verifier already has all the knowledge to start with. Because of this reason, every problem in BPP has a Zero Knowledge Proof, and we refer to these problems as the trivial ones. Now consider a non trivial language, and a corresponding prover that is more powerful than a polynomial one. How can we use a simulation to show that such a prover does not leak information? The key idea is that we make the simulator just as powerful as the verifier. Then if a simulator with limited computational capabilities is able to replicate the interaction, the verifier cannot learn anything from it, as the simulator has no knowledge advantage to start with. Then, since the simulator and the prover interact indistinguishably with the verifier, the verifier cannot learn anything from the prover as well. In essence, the existence of the simulator shows that all the verifier can learn is that which is given by a BPP machine, and since the verifier is BPP to start with, it gains Zero Knowledge. We formalize this by saying that a language L is zero knowledge if it has an interactive proof system P, V such that, for every possible verifier V', there exists a probabilistic polynomial time Turing machine $M_{V'}$ that for any $x \in L$ satisfies that $\langle P, V \rangle (x)$ and $M_{V'}(x)$ are similarly distributed[31]. Using this intuition and playing around with the loose parts of it, we can start defining the interesting classes of zero knowledge proofs.

7.1 Perfect Zero Knowledge

The first and stricter class of zero knowledge proofs are the so called perfect zero knowledge proofs. Ideally, according to the above definition we would like

[31]More on this later, essentially the level of similarity of the two distributions yields the definition of *perfect*, *statistical* and *computational* zero knowledge

the definition of zero knowledge proof to simply require that the ensembles $\langle P, V \rangle (x)$ and $M_{V'}(x)$ are identically distributed. However, as far as we know, no non trivial languages satisfy that requirement. To show why this needs to be case, if a simulator can perfectly and infallibly simulate an interaction between prover and verifier, despite having less computational power (and or information available) than this effectively implies that the prover is not using its superior capabilities to their best or at all! This in turns will imply that language being decided is in fact of the trivial class. So, in order to allow us to decide any harder language, the simulator will have at some point have to give up. We can formulate this by giving the following definition of Perfect Zero Knowledge Proof.

Definition 20. *Let L be a language. Then L has a **perfect zero knowledge proof system** if there exist P, V such that:*

- *P, V is an interactive proof system for L*
- *For any probabilistic polynomial time ITM V', there exists a probabilistic polynomial time TM $M_{V'}$ such that, for any $x \in L$ the following two hold:*
    1. *The machine $M_{V'}$ outputs the special symbol $\perp$ with probability:*

    $$\Pr[M_{V'}(x) = \perp] \leq \frac{1}{2}$$

    2. *Letting $m_{V'}$ be a variable describing the output of $M_{V'}$ conditioned[32] on the output of $M_{V'} \neq \perp$ we must have that $\langle P, V \rangle (x)$ and $m_{V'}(x)$ are identically distributed*

As always, the bound on how often the simulator is allowed to fail can be made negligible. It is interesting to note how the identical distribution requirement mentioned is a notion that is mostly used in information theoretic contexts, and as such this scheme is in a sense perfectly secure. To be more precise, this implies that even a possibly cheating computationally unbounded verifier cannot, quite amusingly, detect if it is in a simulation or in a real interaction. Furthermore, we can make the observation that, for any verifier V', its execution is completely determined by the contents of its random tape and the messages that are sent on the communication tape by P. Knowing this we can define a random variable $\text{view}_{V'}^{P}(x)$ to describe the transcript of the interaction between P and V', including the contents of V' random tape and all the messages sent from P to V'. Using this, we can replace the $\langle P, V \rangle (x)$ by $\text{view}_{V'}^{P}(x)$ and obtain an equivalent definition that is slightly easier to work with, as it allows to account for verifiers that arbitrarily deviate from the protocol (e.g. some that terminate between steps). With this definition, let us give now an example of a Perfect Zero Knowledge Proof for the language of Graph Isomorphism.

[32] $\Pr[m_{V'}(x) = \alpha] = \Pr[M_{V'}(x) = \alpha \mid M_{V'}(x) \neq \perp]$

Example 6. *First of all, let us define the language* $\mathtt{GI}$*.*

$$\mathtt{GI} = \{ \langle G, H \rangle \mid G, H \text{ graphs and } G \cong H \}$$

Note that $\mathtt{GI}$ *is*[33]*, as far as we know, not* NP*-complete nor in* BPP*. This is important since every language in* BPP *has a trivial proof, while known ZKP languages that are* NP*-complete require further assumptions. The following protocol [13] [14] is a Perfect Zero Knowledge Proof System for* $\mathtt{GI}$*.*

Proof that $G_1 \cong G_2$

Prover | **Verifier**

$$G_1 = (V_1, E_1), G_2 = (V_2, E_2)$$

Find ϕ s.t. $\phi(G_1) = G_2$

$\sigma \leftarrow_{\$} S_{V_2}$

$G' = \sigma(G_2)$

$$\xrightarrow{\quad G' \quad}$$

$i \leftarrow_{\$} \{1, 2\}$

$$\xleftarrow{\quad i \quad}$$

```
if $i =_? 2$
  $\pi := \sigma$
else
  $\pi := \sigma \circ \phi$
```

$$\xrightarrow{\quad \pi \quad}$$

return $\pi(G_i) =_? G'$

Figure 3: A perfect zero knowledge proof for Graph Isomorphism

1. *Let the common inputs of* P, V *be two graphs* G_1, G_2 *s.t.* $G_i = (V_i, E_i)$*. Also if* $(G_1, G_2) \in \mathtt{GI}$ *it must be that* $G_1 \cong G_2$ *and as such there exists an isomorphism* $\phi : V_1 \to V_2$*. We stress that this* ϕ *is not publicly known*[34]*.*

2. *On the first step,* P *uniformly randomly selects a permutation* $\sigma \in S_{V_2}$*, and computes the graph* $G' = \sigma(G_2)$*. Then* P *sends such graph to* V*.*

[33]In the most general case, as we actually do know polynomial time algorithm for special cases such as trees

[34]By this we mean that the prover either uses its superior computational ability to find ϕ, or that it is only known to the prover. Strictly speaking the second formulation requires a slightly different formulation of ZKP, but the proof follows similarly

3. *Once V receives the graph G' from P, it randomly selects an $i \in \{1, 2\}$, and sends it to P.*

4. *When P receives i, it acts as follows. If $i = 2$, then P sends σ, else it sends $\sigma \circ \phi$.*

5. *Finally, if the permutation received from P is an isomorphism between G_i and G', then V accepts, else it rejects.*

The rationale behind this is quite straightforward. On the first step, the prover generates a challenge graph, that it claims is isomorphic to both of the input graphs. Of course, if the two graphs are non isomorphic no graph can satisfy this claim. On receiving this graph, the verifier issues a challenge, asking to receive an isomorphism between the received graph and one of the two inputs. If the graphs are isomorphic, then the prover can always easily (if the isomorphism ϕ is know) answer the query. Else, it must fail with some probability.

Theorem 5. *The above is a Perfect Zero Knowledge Proof for* `GI`*.*

Proof. First of all, note that the verifier can easily be implemented in polynomial time. We do not know of a way to similarly implement the prover, but luckily this is not a requirement. If the permutation ϕ is known by P a priori then we can actually find such an implementation. First of all, we aim to show that P, V is an interactive proof system.

- Completeness. If $(G_1, G_2) \in \mathtt{GI}$, then P is always able to find ϕ such that $\phi(G_1) = G_2$. Then $G' = \sigma(G_2) = (\sigma \circ \phi)(G_1)$. Then the prover can always satisfy the verifier challenge, and as such the completeness bound is 1.

- Soundness. If $(G_1, G_2) \notin \mathtt{GI}$, then there is no isomorphism between the two input graphs. Therefore for any G' that a possibly cheating prover E might choose, there exists a $j \in \{1, 2\}$ such that $G_j \ncong G'$. So, since the verifier chooses $i \in_R \{1, 2\}$ after G' is fixed, $\Pr[i = j] = \frac{1}{2}$ and as such the prover will fail to convince the verifier at least $\frac{1}{2}$ of the times, and that is the soundness bound.

Now, for the arguably more challenging section of the proof let us prove that the above protocol is indeed zero knowledge. We show a simulator[35] that satisfies the zero knowledge condition. Let V' be an arbitrary polynomial time randomized ITM. We define the following simulator $M_{V'}$, that runs on input $x \equiv (G_1, G_2)$:

1. First of all, let $q(|x|)$ be a polynomial bound on the running time of V'. We will be simulating V' on M'_V, and to do so we will first allocate a string of $q(|x|)$ random bits to be used[36]. We call such bits r.

[35]In fact, a family of simulators dependant on verifiers

[36]An alternative would be interactively answering V' queries for random bits, but this is conceptually easier

2. The simulator randomly selects $j \in \{1, 2\}$, and permutation $\pi \in S_{V_j}$, computes the graph $H = \pi(G_j)$.

3. The simulator now starts simulating V', giving it x as common input, r as the random tape, H in the incoming message tape.

4. After polynomially many steps, V' will place a message i on its outgoing message tape. The simulator reads this value, normalizes it[37] by setting $i = 1$ if $i \neq 2$.

5. If $i = j$ then the simulator outputs (x, r, H, π).

6. Else (i.e. $i \neq j$), the simulator fails and outputs $\perp$.

A couple of notes are in order. First of all, as long as the verifier is polynomial time, the above simulator can be implemented in polynomial time. Secondly, the simulator does not follow exactly the steps of the prover, of course since it is required to run in polynomial time. Thirdly, the simulator is not able to perfectly answer the query every time, as it does not know ϕ as the prover does, and as such it is required to fail some times. Now onto proving that this simulator satisfies the requirement. Let $x \equiv (G_1, G_2) \in \texttt{GI}$. We first show that:

$$\Pr[M_{V'}(x) = \perp] \leq \frac{1}{2}$$

Recall that[38] we showed that, for any randomized algorithm E, any two isomorphic graph G_1, G_2, and with the variables $\Pi \in_R S_{V_1}, I \in_R \{1, 2\}$ we have that:

$$\Pr[E(\Pi(G_I)) = I] \leq \frac{1}{2}$$

Since we see that $\Pr[M_{V'}(x) = \perp] \leq \Pr[E(\Pi(G_I)) = I]$ (as the simulation only fails when V' is able to find out i from $H = \pi(G_i)$), the claim is proved.
Using now the alternative characterization of zero knowledge, we aim to show that $\text{view}^P_{V'}(x)$ and $m_{V'}(x)$ are identically distributed. First of all, note that both deal with quadruples of the form $(x, r, \cdot, \cdot)$, with r in both cases randomly and uniformly selected. As such, we can move on to consider only the next two components, namely the first and second message sent by the prover to the verifier. Let us define $s(x, r)$ be the last two elements of the quadruple outputted by the simulator (as long as the simulation succeeds), and similarly $p(x, r)$ the same elements of the quadruple of the verifier's view of the interaction. We aim to show these are identically distributed. First of all not that once x, r, H are fixed, the output of V', which we will call $v(x, r, H)$ is uniquely determined. It turns out that $s(x, r)$, $p(x, r)$ are identically distributed over the set:

$$C_{x,r} = \big\{(H, \pi) | H = \pi(G_{v(x,r,H)})\big\}$$

[37]This makes the discussion easier, also if V' terminated we can still apply the normalization and proceed as before

[38]In proving the soundness bound for the interactive proof of $\texttt{GNI}$

Unfortunately, the proof is rather tedious[39], and not particularly related to cryptography, so for now I have decided to skip it until I find a simpler one. This concludes out proof that the above is a valid Perfect Zero Knowledge simulator for the interactive proof system, and as such we are done. □

7.2 Computational Zero Knowledge

Perfect Zero Knowledge Proofs have the extremely strong requirement that the simulation and the actual interaction yields the same identical distribution (of course when the simulator is able to answer accurately i.e. does not output $\perp$). However, this requirement is quite stringent, and as such we can relax it by borrowing from pseudorandom generators. In particular, the key insight is that for a generator to be such, it is not necessary for it to be truly random, but only for it to be impossible for a polynomial time procedure to distinguish it from a truly random sequence. In particular, we can formalize this as follows:

Definition 21. *Two random variable ensembles*[40] $\{A_n\}_{n\in\mathbb{N}}$ *and* $\{B_n\}_{n\in\mathbb{N}}$ *are* ***computationally indistinguishable*** *if for any polynomial time algorithm D the function*

$$\delta(n) = |\Pr[D(A_n, 1^n) = 1] - \Pr[D(B_n, 1^n) = 1]|$$

is a negligible function of n.

A couple of notes are, as usual, in order. First of all, the 1^n parameter are used to rule out degenerate cases where the two distributions yields strings logarithmic in the size of n, and as such the D algorithm is unable to run in polynomial time of n. The δ function is often referred to as the attackers advantage. The idea is that this metric can measure how much better the attacker can do over the general strategy of always accepting, rejecting or guessing, which always yield an advantage of[41] 0. In order to further explain how this is useful let us consider the following two examples [15].

Example 7. *Consider the two ensembles $\{A_n\}_{n\in\mathbb{N}}$ and $\{B_n\}_{n\in\mathbb{N}}$ where we set $\Pr[A_n = n] = \Pr[A_n = n+1] = \frac{1}{2}$, and $\Pr[B_n = n] = 1$. So A_n is either n or $n+1$, while B_n always equals n. Then $\{A_n\}_{n\in\mathbb{N}}$ and $\{B_n\}_{n\in\mathbb{N}}$ are* ***not*** *computationally indistinguishable*

Proof. Let D be the algorithm that accepts $(x, 1^n)$ if $x = n$. Then

$$\begin{aligned}\delta(n) &= |\Pr[D(A_n, 1^n) = 1] - \Pr[D(B_n, 1^n) = 1]| \\ &= \left|\frac{1}{2} - 1\right| = \frac{1}{2}\end{aligned}$$

[39]The first element of the pair can be shown to be identically distributed easily, but the second part requires apparently at least a discussion of the authomorphisms of a graph, which is beyond our scope

[40]An ensemble $\{A_n\}_{n\in\mathbb{N}}$ is just an infinite set of random variables over $\{0,1\}^*$ indexed by either an integer n, or by some string in some language

[41]For example, if an algorithm D always accepts then $\delta(n) = |1-1| = 0$

Since $\delta(n)$ is a constant, it is non negligible and as such the two ensembles are distinguishable. □

Example 8. *Consider the two ensembles $\{A_n\}_{n\in\mathbb{N}}$ and $\{B_n\}_{n\in\mathbb{N}}$ where we define $\Pr[A_n = 0^n] = 2^{-n}$, $\Pr[A_n = 1^n] = 1 - 2^{-n}$ and $\Pr[B_n = 1^n] = 1$. In order to make intuitive, A_n can be generated by an algorithm that flips n coins and outputs 0^n if all landed tails, and 1^n otherwise, and B_n always outputs 1^n. Then $\{A_n\}_{n\in\mathbb{N}}$ and $\{B_n\}_{n\in\mathbb{N}}$ are* ***computationally indistinguishable***

Proof. First of all note that the only case in which the distribution of the two differ is when $A_n = 0^n$. Intuitively, this happens only with negligible probability, and as such the overall advantage that any D could gain happens scarcely enough that it does not matter. Let us give a formal proof of this fact. Let D be any polynomial time algorithm. Then let:

$$\begin{aligned}\phi(n) &= \Pr[D(B_n, 1^n) = 1] - \Pr[D(A_n, 1^n) = 1]\\ \delta(n) &= |\phi(n)|\end{aligned}$$

Where ϕ is used so not to have to deal with the absolute value.

First of all note that if $A_n \neq 0^n$ then the two distributions are identical (both equal to 1^n). So:

$$\Pr[D(A_n, 1^n) = 1 \mid A_n \neq 0^n] = \Pr[D(B_n, 1^n)]$$

Then we can split the case of D accepting A_n into two branches:

$$\begin{aligned}\Pr[D(A_n, 1^n) = 1] =& \Pr[D(A_n, 1^n) = 1 \wedge A_n \neq 0^n]\\ &+ \Pr[D(A_n, 1^n) = 1 \wedge A_n = 0^n]\end{aligned}$$

Now note that the second term is greater than 0, and as such:

$$\begin{aligned}\Pr[D(A_n, 1^n) = 1] &\geq \Pr[D(A_n, 1^n) = 1 \wedge A_n \neq 0^n]\\ &= \Pr[D(A_n, 1^n) = 1 \mid A_n \neq 0^n] \Pr[A_n \neq 0^n]\\ &= \Pr[D(B_n, 1^n) = 1] \Pr[A_n \neq 0^n]\\ &= \Pr[D(B_n, 1^n) = 1] (1 - 2^{-n})\end{aligned}$$

Using this we can bound the ϕ from above:

$$\begin{aligned}\phi(n) &= \Pr[D(B_n, 1^n) = 1] - \Pr[D(A_n, 1^n) = 1]\\ &\leq \Pr[D(B_n, 1^n) = 1] - \Pr[D(B_n, 1^n) = 1] (1 - 2^{-n})\\ &= \Pr[D(B_n, 1^n) = 1] (1 + 2^{-n} - 1)\\ &= 2^{-n} \Pr[D(B_n, 1^n) = 1]\\ &\leq 2^{-n}\end{aligned}$$

Similarly we can use the fact that $\Pr[A \wedge B] \leq \Pr[A]$ to conclude that:

$$\begin{aligned}
&\Pr[D(A_n, 1^n) = 1] \\
&= \Pr[D(A_n, 1^n) = 1 \wedge A_n \neq 0^n] + \Pr[D(A_n, 1^n) = 1 \wedge A_n = 0^n] \\
&\leq \Pr[A_n \neq 0^n] + \Pr[D(A_n, 1^n) = 1 \wedge A_n = 0^n] \\
&= 2^{-n} + \Pr[D(A_n, 1^n) = 1 \mid A_n \neq 0^n] \Pr[A_n \neq 0^n] \\
&= 2^{-n} + \Pr[D(B_n, 1^n) = 1] (1 - 2^{-n})
\end{aligned}$$

And conclude that ϕ is bounded below by:

$$\begin{aligned}
\phi(n) &= \Pr[D(B_n, 1^n) = 1] - \Pr[D(A_n, 1^n) = 1] \\
&\geq \Pr[D(B_n, 1^n) = 1] - 2^{-n} - \Pr[D(B_n, 1^n) = 1] (1 - 2^{-n}) \\
&= -2^{-n} + 2^{-n} \Pr[D(B_n, 1^n) = 1] \\
&\geq -2^{-n}
\end{aligned}$$

Where the last inequality follows since $\Pr[D(B_n, 1^n) = 1] \geq 0$. Now, since $\delta(n) = |\phi(n)| \leq 2^{-n}$, it follows that δ is a negligible function of n, and as such A_n, B_n are computationally indistinguishable.

□

Using this we can define the idea of a Computational Zero Knowledge proof.

Definition 22. *Let L be a language. Then L has a* ***computational zero knowledge proof system*** *if there exist P, V such that:*

- *P, V is an interactive proof system for L*
- *For any probabilistic polynomial time ITM V', there exists a probabilistic polynomial time TM $M_{V'}$ such that the following ensembles are computationally indistinguishable:*
  1. $\{\langle P, V \rangle (x)\}_{x \in L}$
  2. $\{M_{V'}(x)\}_{x \in L}$

As you can see, other than the different level of "closeness" of the two distributions, the main other difference is that we do not require bounded failure of the simulation by the $\perp$ symbol. The rationale between this is quite simple. First of all, remember that in perfect zero knowledge we can repeat the interaction multiple times[42] to reduce the probability of outputting $\perp$ to a negligible one (in the size of the input). In particular this implies that the simulator only differs from the interaction a negligible percentage of times, and the above example shows how two ensembles that only differ with negligible probability are computationally indistinguishable. Computational Zero Knowledge Proofs are

[42] In order for this to be formal we would actually need to prove that sequential composition of zero knowledge proofs does not yield any knowledge, but for the sake of this discussion just know this holds

in a sense more efficient then Perfect Zero Knowledge proofs, as in practical applications there is no need to perfectly simulate the interaction to guarantee zero knowledge. From a practical standpoint as well, computational Zero Knowledge Proofs are those that are generally referred to as simply ZKP, and those around which applications mostly focus.

7.3 Zero Knowledge Proofs for NP

After having defined Zero Knowledge Proofs, it is just natural to consider what types of languages have a ZKP. In this section we show a potentially surprising result, that, under some reasonable assumptions, every language in NP has a Zero Knowledge Proof.

7.3.1 Bit Commitment

The machinery that we introduce is that of Bit Commitment. In a nutshell, the intuition behind this is to digitally simulate the action of receiving a sealed letter from someone, and knowing that its content will not have changed since it was written (while of course being unable to learn its contents until the seal is broken). Loosely speaking, a bit commitment algorithm works between two parties that do not trust each other. The first commits itself to a value, and sends to the second some sort of transcript. Once the second requests the opening of the seal, the first party sends a certificate that, together with the transcript, reveals the committed value and proves that it was indeed committed. Formally, we define a bit commitment scheme as follows:

Definition 23. *A **bit commitment scheme** is a pair of probabilistic polynomial time interactive machines (S, R) that, on common security parameter n, satisfy the following requirements.*

- ***Secrecy**. For any probabilistic polynomial time interactive machine E, the ensembles $\{\langle S(0), E\rangle (1^n)\}_{n\in\mathbb{N}}$ and $\{\langle S(1), E\rangle (1^n)\}_{n\in\mathbb{N}}$ are computationally indistinguishable.*
- ***Unambiguity** We let $(r, \bar{m})$ be, respectively, the random bits used by the receiver and the messages received by the sender. Then we say that $(r, \bar{m})$ is a i-commitment if for some random string s the machine[43] $S_s(i, 1^n)$ produces the sequence of messages $\bar{m}$ when interacting with R with local coins r. If $(r, \bar{m})$ is both a 0-commitment and a 1-commitment, then it is ambiguous. We require that for all but a negligible fraction of $r \in \{0,1\}^{poly(n)}$ there exists no $\bar{m}$ such that $(r, \bar{m})$ is ambiguous.*

As you can see, the unambiguity requirement is much more involved than the secrecy one, and it is that requirement that ensures the commitment property of the scheme. In fact, something interesting to note is that the secrecy requirement is computational, while the unambiguity one has an information theoretic

[43]Here we use the notation $M_r(x)$ to denote the machine M running with local coins r on input x

flavour[44]. In particular, note that once S has committed a value, in order to provide verification, it can send its private input and the random coin it used to R, which can then run S's algorithm to verify the certificate in polynomial time.

As far as we know, the existence of bit commitment schemes requires some assumptions, namely the existence of one way functions, functions that are easy to compute but hard to invert. For a more through description, look no further than Appendix A. While a proof that uses solely those kind of function exists [18], it requires some discussion of pseudorandom generators, which are not particularly insightful nor too related. Instead, we can use the stronger assumption of the existence of one way permutations (i.e. one way functions that are also bijections), which results in a simpler proof [19].

Let us first discuss the intuition that makes the following construction useful. Let us assume that we have access to a one way permutation f. We want to commit to a bit b. What we can do is select a random string. We can make use of a property of one way functions, that is the property that they have hard cores. This is some property of the input of a one way function that cannot be guessed from the output (discussed extensively in Appendix A). Let us denote $h(\cdot)$ as one of those hard cores. Then let us select some random string s that we will call the certificate. We will be sending to the other party the pair $(f(s), h(s) \oplus b)$. This has a double effect. First of all, since the function is one way, we won't be able to efficiently find a s' such that $f(s) = f(s')$, and so effectively we won't be able to find another certificate with which to trick the other party. On the other hand, since h is a hard core, the other party will not be able to guess it effectively from $f(s)$, and such $h(s) \oplus b$ and consequently b cannot be recovered by it. Of course, revealing s, b will make the other party be able to efficiently verify that we did not cheat.

Theorem 6. *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a one-way permutation. Let $h : \{0,1\}^* \to \{0,1\}$ be a hard-core of f. Then the following is a bit commitment scheme.*

- ***Commitment.*** *To commit the value i, uniformly select $s \in \{0,1\}^n$. Send $(f(s), h(s) \oplus i)$ to the receiver.*
- ***Reveal.*** *To reveal, the sender reveals s, i. The receiver, which had received commit (α, j), accepts if $f(s) = \alpha$ and $h(s) \oplus i = j$.*

Proof. The secrecy requirement is a direct consequence of f being one way, and b being one of its hard cores. The unambiguity requirement is a consequence of f being injective, as no ambiguous view exists. $\square$

[44]This commitment scheme is referred to as perfectly binding. There is also a corresponding, mutually exclusive, "dual"-like notion of perfectly hiding schemes [16], in which the secrecy requirement is information theoretic and the unambiguity is computational. This can be used to provide Perfect ZK Arguments for languages in NP [17], but they are relatively more advanced and as such will not be discussed here

Furthermore, the above scheme has one additional desirable feature: the fact that it only requires to interact one way (from the prover to the verifier). As such we define:

Definition 24. *A **one-way-interaction bit commitment scheme** is a polynomial time algorithm F such that.*

- *For s uniformly drawn from $\{0,1\}^n$, the ensembles $\{F_s(0,1^n)\}_{n\in\mathbb{N}}$ and $\{F_s(1,1^n)\}_{n\in\mathbb{N}}$ are computationally indistinguishable.*
- *There does not exist s such that $F_s(0,1^n) = F_s(1,1^n)$.*
- *For such a scheme, the (canonical) reveal of $F_s(i,1^n)$ is (i,s).*

With this we can see that the one-way permutation construction yields a one-way-interaction bit commitment scheme where $F_s(i,1^n) = (f(s), h(s) \oplus i)$.

7.4 ZKP for NP-complete problems

Now that we have defined our machinery, we can make use of the structure of problems in the NP space. In particular, as detailed in Appendix C, there is a class of problems in NP known as the NP-complete problems. These are problems that are efficiently reducible to any other problem in NP, i.e. such that the existence of an efficient solution for any of them will imply efficient solutions to each problem in NP. It follows that by finding a ZKP for a single one of these problems, we can show that any problem in the class has a Zero Knowledge Proof. In particular, the language that we will tackle is that of Graph Three Colouring.

Definition 25. *Let $G = (V,E)$ be a finite graph. A **3-colouring** for the graph is a function $\pi : V \to \{1,2,3\}$ such that for all $(u,v) \in E \implies \pi(u) \neq \pi(v)$*

Definition 26.

$$\mathtt{3COL} = \{G \mid G \text{ has a valid 3-coloring}\}$$

The language 3COL as defined is NP-complete [20]. Let us now present a computational ZKP [13] [21] for this language. We use a one-way-interaction bit commitment scheme which denotes by $C_s(i)$ the commitment of value $i \in \{1,2,3\}$ using random coins s (That is $C_s(i) = F_s(i,1^n)$). Note how the commitment schemes presented before only allow us to commit a single bit, while this one is able to commit a ternary bit of sorts. Of course, the transformation from one scheme to the other is trivial and only requires using the bit commitment scheme twice for every value (e.g. committing 00 for 1, 01 for 2, 10 for 3).

Proof that G is 3-colorable

Prover | **Verifier**

$G = (V, E)$, bit commitment scheme $C_r(s)$

π a 3-colouring of G

$\sigma \leftarrow\!\$\, S_3$

$\phi \coloneqq \sigma \circ \pi$

$s_i \leftarrow\!\$\, \{0,1\}^n, \forall i \in V$

$c_i \coloneqq C_{s_i}(\phi(i))$

$$\xrightarrow{c_1, \dots, c_n}$$

$(u, v) \leftarrow\!\$\, E$

$$\xleftarrow{u, v}$$

$\gamma_u \coloneqq \phi(u)$

$\gamma_v \coloneqq \phi(v)$

$$\xrightarrow{s_u, \gamma_u, s_v, \gamma_v}$$

fail if $C_{s_u}(\gamma_u) \neq_? c_u$

fail if $C_{s_v}(\gamma_v) \neq_? c_v$

fail if $\gamma_u =_? \gamma_v$

return true

Figure 4: A zero knowledge proof for Graph 3-Colouring

Example 9. *Let the common input be $G = (V, E)$, where $V = \{1, \dots, n\}$. This graph is 3-colourable, with colouring π, which is unknown to the verifier*[45].

1. *The prover randomly uniformly selects $\sigma \in S_3$. It sets $\phi \equiv \sigma \circ \pi$. For each $i \in V$, it generates a random string $s_i \in \{0,1\}^n$ and computes $c_i \equiv C_{s_i}(\phi(i))$. The prover then sends $c_1, \dots, c_n$ to the verifier.*

2. *The verifier uniformly selects an edge $(u, v) \in E$, and sends it to the prover*

3. *The prover sends $(s_u, \phi(u))$ and $(s_v, \phi(v))$ to the verifier*

4. *On the messages (α, i), (β, j), with $\alpha, \beta \in \{0,1\}^n$ and $i, j \in \{1, 2, 3\}$, the verifier checks that $c_u = C_\alpha(i)$ and that $c_v = C_\beta(j)$. If not it rejects. Finally, it accepts if $i \neq j$, and if as such the edges are differently coloured.*

A small discussion is, of course, in order. The fundamental idea is that the prover creates n boxes, each containing a colour. It locks these boxes and sends

[45]As always, such π could be either computed by a superior prover, or given as auxiliary input

them to the verifier. The verifier then chooses two of these boxes, such that they are connected by an edge in the original graph. It asks the prover to open the boxes, and after having verified that they haven't been tampered with, the verifier concludes that the colouring is valid if the two boxes it chose have different colours. Intuitively, the verifier only ever learns that two vertexes have different colours, that of course is a requirement for any colouring. In particular, note the role of the permutation σ. The idea is that, on every run, the prover will select a different sigma, and as such repeating the protocol will possibly yield different colourings for two vertexes. This way, a cheating verifier cannot alternatively ask for the colouring of each vertex in the graph, and so it cannot piece back the original colouring.

Theorem 7. *The above protocol is a computational zero knowledge proof for* `3COL`*.*

Proof. First of all, we will verify that the above protocol is an interactive proof system for `3COL`.

- Completeness: If G is indeed 3-colourable with colouring π, then the above protocol always accepts.
- Soundness: Let us suppose that G is not 3-colourable. Then the n committed values cannot correspond to a 3-colouring, and as such there must be at least one edge (v, v') such that $c_v, c_{v'}$ are commitment of the same value $k \in \{1, 2, 3\}$. So no matter what the prover commits, the verifier will reject with probability at least $\frac{1}{|E|}$. Since we can bound the number of edges with $|E| \leq \binom{n}{2}$, we have a polynomial bound of $\frac{2}{n(n+1)}$.

And so the protocol is an interactive proof system for `3COL`, with completeness bound 1 and with a weak soundness bound $1/|E|$. Repeating the protocol yields an interactive proof as required[46]. For any verifier V', we show a simulator $M_{V'}$ that shows that the prover has the zero knowledge property.

- Let the input be $G = (V, E)$, where $V = \{1, \dots, n\}$.
- Again, let $q(|G|)$ be a polynomial bound on the running time of V'. Allocate a uniformly randomly selected string $r \in \{0, 1\}^{q(|G|)}$ random bits to be used.
- Uniformly select $e_i \in \{1, 2, 3\}$, and $s_i \in \{0, 1\}^n$, for $i \in V$. Compute $d_i \equiv C_{s_i}(e_i)$,
- Simulate V' with G as input, and r as the random bits. Also, put the sequence of commitments $d_1, \dots d_n$ in V' incoming messages tapes.

[46]The situation is a bit subtle as we would want to ensure that repeating the protocol does not yield any knowledge. Luckily, the sequential composition lemma that we assumed earlier and here ensures that this is the case

- After V' has been simulated, we can assume that it will have some message m on its out message tape. We can also assume that $m \in E$, as if it is not we can adapt $M_{V'}$ so that is selected some edge in E to be used.
- Let $(u, v) = m$. Then if $e_u \neq e_v$ the simulator halts with output

$$(G, r, (d_1, \dots, d_n), (s_u, e_u, s_v, e_v))$$

- Else we output $\perp$

First of all, you might be puzzled by the presence of the $\perp$ symbol in this simulator. However, we noted before that computational proofs do not need such help, as repeating the simulation often will result it in it failing negligibly often, and as such the interaction will still be computationally indistinguishable. So, adding the capability of outputting $\perp$ does not "add" any extra power to the definition, and as such we can use it here to simplify our life (requiring as always that it is not output too often). We first show that. We note that the $\perp$ symbol is only output when the two edges that were selected by the verifier V' have been assigned the same colour. Intuitively, we would like to use the fact that commitments are computationally indistinguishable in order to show that the verifier cannot do much better than to randomly pick two edges (in which case the probability of getting two edges with the colour is $\frac{1}{3}$). This is easy to show, as any V' that is able to distinguish commitments will directly yield an algorithm that violates the secrecy property of the scheme[47]. Now, for the next step we aim to show that the view of the interaction of the prover and the verifier is computationally indistinguishable to the output of the simulator. In particular, let $m_{V'}(G)$ be the random variable denoting the output of $M_{V'}$ on input G conditioned on it being different from $\perp$. Let also A be any probabilistic time algorithm. We let:

$$\epsilon_A(G) \equiv \left|\Pr[A(m_{V'}(G)) = 1] - \Pr\left[A(\text{view}^P_{V'}(x))\right]\right|$$

We would like to show that $\epsilon_A(G)$ is a negligible function in n, where n is the number of vertices in G. First of all, recall that both the view of the interaction of the machine P with V and the output of the simulator are distributed over tuples of the form $(r, (\alpha_1, \dots, \alpha_n), (s_u, e_u, s_v, e_v))$ where (at least in the view of the actual interaction) we have r as the random tape of the verifier, α_i as the commitment of the vertex colours, e_i the colour of the i vertex and s_i the random string used for committing it. First of all note that r is in both cases uniformly distributed over $\{0, 1\}^q(n)$ where again we use q as a polynomial bound on the runtime of V'. It suffices now to show that if the two distributions are not computationally indistinguishable, then we will reach a contradiction with the properties of the commitment scheme. While the actual proof is quite technical, the main idea is that for any verifier V' we can define $p_{u,v}(G), q_{u,v}(G)$ as the probability of the verifier choosing edge (u, v) on step 2 of the protocol. Since the

[47]The full argument is not too complex, but long and not insightful, using a classical reduction proof as many of those seen in cryptography

verifier gets computationally indistinguishable ensembles, the two probability cannot differ by a non negligible factor (as then V' could yield an algorithm that contradicts the secrecy requirement of the scheme). One can refer to [14] for a complete proof.

□

A couple of things are worth noting. First of all, the proof above has a weak soundness bound, and as such a one round protocol has only a moderate chance of convincing the verifier. However, as always repeating the algorithm $k * |E|$ times yields a proof with error probability bounded by e^{-k}, and the sequential composition lemma will also guarantee that this is zero knowledge. Secondly, the main significance of this proof relies on the fact that `3COL` is a `NP`-complete language, and as such it is polynomial time reducible from any other problem in `NP`. This implies that, if one way functions exists, then every language in `NP` has a computational zero knowledge proof.

8 Applications

In the above sections, we discussed some of the mathematical definition of zero knowledge proofs, their properties and some basic proofs for languages such as `GI` and `3COL`. Of course, all would be for naught if Zero Knowledge Proofs did not have practical applications, and were not a capable tool for solving various problems. Taking inspiration from [22], we hope to show a variety of proofs that are applicable with real life problems.

8.1 Schnorr Signatures

The main problem that Schnorr signatures (proposed firstly in [23]) aim to solve is that of signing a message, so that the receiver can ensure that it was not tampered with. They are particularly helpful in practice, since there are ways to make the signatures attractively small. Schnorr algorithm is based on the hardness of the discrete logarithm problem, and in particular on the following zero knowledge protocol of knowledge of the discrete logarithm, due to [24].

Proof of knowledge of a discrete log β in base α

Prover **Verifier**

α, β, N

Find x s.t. $\alpha^x \equiv \beta \pmod{N}$
$r \leftarrow_{\$} \{1, \ldots, \phi(N)\}$
$\gamma \equiv \alpha^r \pmod{N}$

$$\xrightarrow{\quad\gamma\quad}$$

$b \leftarrow_{\$} \{0, 1\}$

$$\xleftarrow{\quad b\quad}$$

$y \equiv r + bx \pmod{\phi(N)}$

$$\xrightarrow{\quad y\quad}$$

return $\alpha^y \equiv_? \gamma\beta^b \pmod{N}$

Figure 5: A zero knowledge proof for Discrete Log

- The prover wants to show that it possesses an x such that $\alpha^x \equiv \beta \pmod{N}$, where α, β, N are chosen in advance and known to both parties[48].
- The prover chooses an r uniformly and randomly from the set $1, \ldots, \phi(N)$, where $\phi(\cdot)$ is Euler's totient function. It then computes the value of $\gamma \equiv \alpha^r$, and sends it to the prover.
- The verifier selects a bit $b \in \{0, 1\}$ uniformly and randomly, and sends it to the prover
- The prover computes the exponent $y \equiv r + bx \pmod{\phi(N)}$ and sends it to the verifier
- The verifier accepts if $\alpha^y =_? \gamma\beta^b$

One can easily verify that, if both parties follow the protocol, then the equation $\alpha^y = \alpha^{r+bx} = \alpha^r\alpha^{bx} = \gamma(\alpha^x)^b = \gamma\beta^b$ holds. The main idea is that the prover commits itself to a value r, that is kept hidden from the verifier thanks to the hardness of the discrete log. Then the verifier issues a challenge, in this case the bit b. If a cheating prover did not know x, then he would not be able to

[48]In particular N is chosen so that the discrete log is hard in the group $\mathbb{Z}_N^\times$. α is chosen to be a primitive root modulo N

find a y that satisfies the above equation. Furthermore, the random parameter r ensures that the verifier does not learn anything from y about x, ensuring the scheme is zero knowledge. Now, the idea is that, using something called the Fiat Shamir Heuristic [25], we can turn an interactive proof into a non interactive one. In summary, since the only contribution of the verifier to the interaction is to provide a challenge, we can replace this interaction with the use of a cryptographic hash function. We make the verifier compute a hash of the parameters, and use the hash as the challenge. By the fact that the hash is determined from public parameters the verifier can make sure that the prover is not cheating selecting one that would advantage it. Also, by the fact that the hash function is indistinguishable from a random function, in order to fool the verifier the prover will have to find a collision in the hash function output, which is computationally at least comparably hard to solving the original problem!. A transformation of the above protocol to a non interactive one is as follows:

Non Interactive Proof of knowledge of a discrete log β in base α

Prover **Verifier**

α, β, N. A Hash function $H(\cdot)$

Find x s.t. $\alpha^x \equiv \beta \pmod{N}$

$r \leftarrow_{\$} \{1, \ldots, \phi(N)\}$

$\gamma \equiv \alpha^r \pmod{N}$

$b \coloneqq H(\alpha, \beta, \gamma)$

$y \equiv r + bx \pmod{\phi(N)}$

$$\xrightarrow{\gamma, y}$$

$b' \coloneqq H(\alpha, \beta, \gamma)$

return $\alpha^y \equiv_? \gamma\beta^{b'} \pmod{N}$

Figure 6: A non interactive proof of knowledge for Discrete Log

- The prover has an x such that $\alpha^x \equiv \beta \pmod{N}$. The conditions are as previously. Furthermore, both the prover and the verifier have access to a hash function $H(\cdot)$
- The prover chooses an r uniformly randomly as before, and computes $\gamma \equiv \alpha^r$
- The prover now computes $b \coloneqq H(\alpha, \beta, \gamma)$, which will be his challenge
- Finally, the prover computes $y \equiv r + bx \pmod{\phi(N)}$, and sends the pair (γ, y) to the verifier

- The verifier, or anyone can ensure that $\alpha^y =_? \gamma\beta^b$. Also note how the verifier can compute b from the public parameters easily.

The reasoning for why this would work is exactly the same as the interactive version, but the hash function $H(\cdot)$ allows us to fairly select a challenge without the possibility of cheating. In particular, a further modification of the protocol allows us to get Schnorr signatures, a way to sign a message using $4t$-bit signatures, where t is a security parameter. In fact, it was later shown [26] that only $3t$-bits are required. The scheme works as follows:

Schnorr Signature, for message M

Prover		**Verifier**
	α, β, N. Hash function $H(\cdot)$	
Find x s.t. $\alpha^x \equiv \beta \pmod{N}$		
$r \leftarrow_\$ \{1, \dots, \phi(N)\}$		
$\gamma \equiv \alpha^r \pmod{N}$		
$e := H(\gamma, M)$		
$y \equiv r - ex \pmod{\phi(N)}$		
	$\xrightarrow{e, y}$	
		$\gamma' \equiv \alpha^y \beta^e \pmod{N}$
		return $e =_? H(\gamma', M)$

Figure 7: Schnorr Signatures

- The prover has a private x such that $\alpha^x \equiv \beta \pmod{N}$. α, β, N are published so that both parties know it. Also H is a known hash function such that: $H : \{0,1\}^n \to \{0, \dots, 2^t - 1\}$
- Let us suppose that the prover has a message $M \in \{0,1\}^*$ that he wants to sign.
- The prover chooses a random r uniformly as before. Set $\gamma \equiv \alpha^r$.
- The prover computes $e := H(\gamma, M)$, and $y \equiv r - ex$.
- Finally, the provers sends (e, y) as the signature
- To verify, the verifier computes $\gamma' \equiv \alpha^y \beta^e \pmod{N}$, and finally verifies that $e =_? H(\gamma', M)$

It is easy to check that if both parties follow the procedure then $\gamma' = \alpha^y \beta^e = \alpha^y \alpha^{ex} = \alpha^{r-ex} \alpha ex = \alpha^r = \gamma$ and as such $H(\gamma, M) = H(\gamma', M)$.

8.2 Secure Remote Password

The Secure Remote Password protocol [27], or SRP for short, is the most common example of zero knowledge proofs used for authentication. Common schemes usually use a plaintext approach, in which the user communicates a username and a password to a (trusted) server, that then has the burden of securely maintaining said information. On the other end, SRP only requires that the user itself maintains a secret key, is resistant to dictionary attacks, and crucially requires no trusted party. In particular, one of its most attractive features is that it can work as a drop in replacement that does not require users to change their workflow. SRP is based on Diffie Hellman, and as such it has some of the same cryptography requirement, namely the intractability of the discrete logarithm and the existence of one way functions. The protocol consists of two steps: the registration step and the authentication step.

Secure Remote Password, with sec param k

User I		**Host**
	A group $\mathbb{G}$, generator g, Hash function $H(\cdot)$	
	.... Registration	
$p, s \leftarrow_{\$} \{0,1\}^*$		
$x := H(s, p)$		
$v := g^x$		
	$\xrightarrow{s, v}$	
		store I, s, v
	.... Authentication	
$a \leftarrow_{\$} \{1, \ldots, \lvert\mathbb{G}\rvert\}$		
$A := g^a$		
	$\xrightarrow{I, A}$	
		lookup s, v from I
		$b \leftarrow_{\$} \{1, \ldots, \lvert\mathbb{G}\rvert\}$
		$B := kv + g^b$
	$\xleftarrow{s, B}$	
	$u := H(A, B)$	
$x := H(s, p)$		$S := (Av^u)^b$
$S := (B - kg^x)^{a+ux}$		$K := H(S)$
$K := H(S)$		

Figure 8: Secure Remote Password protocol

- Preliminaries
- Both parties need to agree on a large safe prime N, and consequently on a group $\mathbb{G} \cong \mathbb{Z}_N$. Also, they agree to a generator of the group g, and to a parameter $k \in \mathbb{G}$.
- Both parties select a cryptographically secure hash function H.
- Finally, the parties agree to an identifier I for the user U.

Next, the registration step:

- The user selects a password p and a random (usually small) salt s.
- It computes $x \coloneqq H(s, p)$. Also, it computes $v \coloneqq g^x$.
- Finally, the user sends to the host the pair (s, v)
- The host stores the triple (I, s, v)

Once this is done, the user can authenticate using the following protocol:

- The user selects a random a, and computes $A \coloneqq g^a$, it then sends (I, A) to the host.
- The host looks up (s, v) using I, selects a random b and computes $B \coloneqq kv + g^b$. It then sends (s, B) to the user.
- Both parties set $u \coloneqq H(A, B)$
- The user computes $x \coloneqq H(s, p)$, and from it the session key $S \coloneqq (B - kg^x)^{a+ux}$. Finally it computes the key $K \coloneqq H(S)$.
- On its end the host computes the session key $S \coloneqq (Av^u)^b$. The final key is then $K \coloneqq H(S)$.

Finally, the proof that the authentication succeeded is shown by the two parties comparing the final key. Technically speaking, SRP is a Zero Knowledge Password Proof, which is a more narrow class of Zero Knowledge proof, however you can see how the ideas that were developed in the previous sections are still easily transferable. In particular, the prover is the user, that shows that it has knowledge of a value x such that the equation below holds for any random a, b, u, and for $v = g^x$:

$$(kv + g^b - kg^x)^{a+ux} =_? (g^a v^u)^b$$

Of course, the security lies in the fact that assuming the hardness of the discrete logarithm, computing x from g^x should be unfeasible.

8.3 Zero Knowledge Range Proofs

Let us consider an interesting class of problems. Imagine that a user has some sort of information associated with it, say x. The other party in play would like to verify that the value $x \in_? [a, b]$, for some interval $[a, b]$. How would the user be able to verify this, without leaking the value of x to the other party? Solving this kind of problem has many practical applications. For example, if x is to be understood as some sort of balance, a transaction can be executed without the receiving party knowing the actual balance of the sender[49]. An other idea

[49]In particular, this has many implications for crypto currency systems, as it allows systems such as Bitcoin to be truly anonymous and confidential. This is not the case currently as in every transaction the addresses of the parties in play are known, and the amount transferred as well

is that, if x is the date of birth of the user, he/she could prove to some sort of service that he/she is over 18, without revealing any private information. The kind of proofs that are used to solve this problem are referred to as Zero Knowledge Range Proofs. In particular, the proofs that we are most interested in this case are those referred to as Non Interactive. In this kind of proof, the challenge and answer method that we presented before is lost, as Peggy tends to compile a "transcript" of a proof, that Victor can check on its own, of course while not leaking any information as possible. These kind of proofs are particularly helpful in a network setting, in which communication is expensive. Let us look at a simple example. First of all we need to introduce the idea of Pedersen Commitment [28].

Definition 27. *Let $\mathbb{G}$ be a group in which the discrete log is hard, g, h be two generators. For a secret m, and random bits r we have the Pedersen Commitment of m :*

$$C_r(m) = g^m h^r$$

Pedersen Commitments have the desirable property that they are homomorphic i.e. $C_{r_1+r_2}(m_1 + m_2) = C_{r_1}(m_1)C_{r_2}(m_2)$. Now, suppose we want to show in one interaction the fact that $x \in [0, 2^N - 1]$. A way to do that is to show that we can encode $x = \sum_{i=1}^N b_i 2^i$ where each $b_i \in [0, 1]$. A non interactive proof [29] for the fact is as follows:

- Select a random value r, and let r_i be such that the bit decomposition of $r = \sum_{i=1}^N r_i 2^i$.
- Compute the commitment $c_i = C_{r_i}(b_i)$, and let $c = C_r(x)$.
- Send to the verifier the commitments c_i together with a proof π that each c_i is a commitment of a value which is either 0 or 1. This can be done by combining discrete log knowledge proofs such as [23] together with a proof of partial knowledge such as those described in [30]. Essentially, by the property of the commitment, the value of c_i is either h^{r_i} or gh^{r_i}. We prove that we either know the discrete log of c_i base h or that we know the discrete log of $g^{-1}c_i$ in base h.
- Finally, the verifier checks the proof π and that $c = \prod_{i=1}^N c_i^{2^i}$

Just to fill in the gap, the proof of partial knowledge in this case can be extracted and simplified, and it works as follows. Recall that in that step of the proof the prover has committed $c_i = g^{b_i} h^{r_i}$. We know that $b_i \in \{0, 1\}$. And $c_i = h^{r_i}$ or $c_i = gh^{r_i}$. Note that in the first case we know a discrete log of c_i, while in the second we know the discrete log of $g^{-1}c_i$ (in both cases in base h and in both cases the log is r_i). If we manage to prove either one of the facts, without letting the verifier know which actually holds, then we will be set. A protocol for this works as following (due to [29]):

- Let $E, f, g \in \mathbb{G}$ be common inputs. E is equal to either f^z or gf^z, where z is known by the prover only.

- The prover actions are conditional on what case holds
  - If $E = f^z$, the prover uniformly and independently selects integers w, r_1, c_1. Computes $a \equiv f^w$, and $b \equiv f^{r_1}(E/g)^{-c_i}$
  - If $E = gf^z$, the prover uniformly and independently selects integers w, r_2, c_2. Computes $a \equiv f^{r_2}E^{-c_2}$, $b \equiv f^w$

  The prover then sends a, b to the verifier
- The verifier sends a random integer c to the prover.
- Again, we act differently in both cases.
  - If $E = f^z$, the prover computes $c_2 \equiv c - c_1$, and $r_2 \equiv w + zc_2$
  - If $E = gf^z$, the prover computes $c_1 \equiv c - c_2$ and $r_1 \equiv w + zc_1$

  Finally, the prover sends r_1, r_2, c_1, c_2 to the verifier
- The verifier accepts if the following hold:

$$\begin{aligned} c &=_? c_1 + c_2 \\ f^{r_1} &= b(E/g)^{c_1} \\ f^{r_2} &= aE^{c_2} \end{aligned}$$

While this particular proof is quite simple, unfortunately it is not particularly efficient. In fact according to [31] proving that a particular x belongs in the interval $[0, 511] = [0, 2^9 - 1]$ while using an underlying group of size 2^{10} requires a proof transcript of size ≈ 196.9 kB. However, many examples in the literature improves this bound, and this paper gives an excellent overview of what is the state of the art [22].

9 Implementation

As supplemental material to this document, we produced a variety of Sagemath worksheets that allow the user to experiment with most of the Zero Knowledge Proofs presented in the text. In particular, the protocols presented are those for:

1. Probabilistic Matrix Multiplication checking (Figure 1)
2. Graph Non Isomorphism Interactive protocol (Figure 2)
3. Graph Isomorphism Perfect Zero Knowledge Proof (Figure 3)
4. Graph 3-Colouring Computational Zero Knowledge Proof (Figure 4)
5. Discrete Logarithm Interactive Zero Knowledge Proof (Figure 5)
6. Discrete Logarithm Non Interactive Zero Knowledge Proof (Figure 6)

7. Schnorr Signature protocol (Figure 7)

8. Secure Remote Password protocol (Figure 8)

We considered providing a worksheet for the Zero Knowledge Range Proof described in 8.3, but the branching that is needed for the implementation translated very poorly to worksheets, and as such it would have not been insightful. In the first four worksheets we also provided “experiments”, that is we set the worksheet up in such a way so that the corresponding protocol is executed multiple times serially, in order to show that the probabilities that we derived correspond to the experimental results.

9.1 How to run it

First of all, ensure to have Sagemath installed on your machine. Sagemath can be installed from `www.sagemath.org`, and make sure that the installed version is 9.0. The code written was tested using that version, and if works on any previous (and possibly later) version it is entirely by accident. The notebooks where designed to be used in a the context of a Jupyter Notebook with Sage kernel. All of this will be installed automatically in most Sage distributions. Once the notebook environment is running, navigate to the folder where the notebooks are stored, and select the one that you are interested in. The code can be run by either selecting “Run all Cells” or by individually stepping on each Cell and pressing Shift+Enter. Each notebook is complete with code generated figures[50] and a concise Markdown description of the protocol at hand. Note that the experimental section at the end of some of the notebooks might take a considerable time to run. The user should feel free to experiment with different parameters (that I conveniently grouped together) in order to get a more intuitive feel of the protocol’s behaviour.

9.2 Implementation Details

There are a couple of notes that I wanted to get out, in regards to implementation details, possible security holes[51], and general development experience. First of all, I would like to say that I was very much impressed with the capabilities of Sage. Not having used it before, and coming from a Python background, I found that the learning curve was quite flat, and it was very easy to get something up and running. The libraries are very well organized, the documentation is usually very helpful and it simplified the examples massively. However, as always in software engineering, there were some hurdles. First of all, whenever I was programming I found an impedance mismatch of sorts between the mathematics and the software. Often it is very small things, but translating the abstractions of Mathematics to Sage was not a frictionless effort. For example, as you can see in the Graph Isomorphism worksheet, the label set of a graph needed some

[50]That might be OS-dependent

[51]i.e. why you should not use the example worksheets of a dissertation as a base for any serious cryptographic work

relabelling in order to correctly behave with randomly selected permutations. In every example that uses a Hash Function (Non Interactive Discrete Log, Schnorr Signature, Secure Remote Password) I needed to essentially hack my way around in converting an hash back to an integer, which does not make me particularly proud.

Now, from a security standpoint there are a few things that could go wrong. First of all, the parameters that I used are chosen absolutely arbitrarily. They are very much too small for any kind of serious application[52]. Furthermore, in cryptography you should always use parameters that are battle-tested, such as the NIST [32] endorsed ones. This is because, even if in general we believe that a problem is intractable, this does not guarantee that all of its instances are, and there might[53] be some particular configuration of parameters that yield very easy to solve problems. Following up, sources of randomness. In these worksheets, I have used the standard random number generator that comes with Sage, which, as far as I know, is not cryptographically secure. This could have many implications. First of all, suitable attacks could fix the seed, and thus render the prover deterministic, which could then be used to break the Zero Knowledge Property. This can be done since, if the verifier knows the random bits of the prover, then it can simulate its following actions perfectly, and as such gain knowledge by predicting every branch and then comparing it with the prover's actual message. Secondly, let us consider the Graph 3 Colouring protocol. In this protocol we have to use bit commitments, and I emulated the construction based on one way permutations that we detailed before. In order to model a one way function, I simply used a standard RNG to generate a string of bits. There are two implications of this. Since I have used a one way function instead of a one way permutation, it might be the case that there are collisions, and as such the unambiguity requirement might be violated (which makes the proof system weaker). Also, there probably are attacks on such a generator, which would allow a verifier to peek the bit committed value before the reveal phase, allowing him to at least partially gather the three colouring .

A Appendix: One Way Functions

One way functions are the basis of a great deal of cryptography. Conceptually, a function is one way if it is easy to compute (i.e. computable in polynomial time), yet hard to invert. Formally, we have two definitions of one way functions:

Definition 28. *A function $f : \{0,1\}^* \to \{0,1\}^*$ is* ***strongly one way*** *if:*

1. *There exists a deterministic polynomial time algorithm A such that $A(x) = f(x)$*
2. *For every probabilistic polynomial time algorithm E, every sufficiently*

[52]In fact I use this in the Graph Non Isomorphism protocol, as finding isomorphisms for a graph which is too big is quite hard, but feasible for small ones
[53]There almost always is

large n, and U_n uniformly distributed over $\{0,1\}^n$ the following is negligible:

$$\Pr\left[E(f(U_n), 1^n) \in f^{-1}(f(U_n))\right]$$

Definition 29. *A function $f : \{0,1\}^* \to \{0,1\}^*$ is* ***weakly one way*** *if:*

1. *There exists a deterministic polynomial time algorithm A such that $A(x) = f(x)$*
2. *For every probabilistic polynomial time algorithm E, every sufficiently large n, and U_n uniformly distributed over $\{0,1\}^n$ the following is non-negligible:*

$$\Pr\left[E(f(U_n), 1^n) \notin f^{-1}(f(U_n))\right]$$

In a nutshell, an algorithm that tries to invert a strongly one way function will succeed only with negligible probability. Conversely, an algorithm that tries to invert a weakly one way function will fail with noticeable probability. Surprisingly, at the moment we do not know of any functions that are definitely one way, even though we have some candidates such as:

$$f_{mult}(x, y) = x \cdot y$$
$$f_{discrete}(x) = g^x$$
$$f_{ssum}(x_1, \ldots, x_N, I) = (x_1, \ldots, x_N, \sum_{i \in I} x_i)$$

Where f_{mult} is simply multiplying two integers together, and, assuming intractability of integer factorization, it is weakly one way. The second problem, $f_{discrete}$ is the discrete logarithm problem, where g^x is taken to be in some group in which the discrete log is assumed to be hard, and g is a generator for the group (more in the discrete log appendix). Instead, f_{ssum} is very closely linked to the NP-complete problem known as subset sum. While it is true that if $\mathsf{P} = \mathsf{NP}$ then f_{ssum} is not one way, the converse does not hold. To see this, note that subset sum could have *worst case* exponentially running time, yet be polynomially solvable in the average case, and as such making f_{ssum} not one way[54]. In fact, $\mathsf{P} \neq \mathsf{NP}$ is necessary but not sufficient condition for one way functions to exist. It turns out [14] that strongly one way functions exist if and only if weakly one way functions do. This justifies our assuming only the existence of vague one way functions, without specifying the type. While for one way functions it is hard to recover the complete input in general, there is nothing stopping a possible attacker from computing some partial information about the input. For example, if f is one way we can define the function[55] $g(x, r) = f(x)||r$ which will be one way, yet always leak half of its input to the attacker. On the other hand, there is some information that such g does not

[54]In fact, we report here f_{ssum} only for historical relevance, as originally it was thought to be a good candidate, while nowadays it is known that the `SSUM` problem is actually efficiently solved in many commonly arising cases [33], and a such we know that it is not

[55]We use || to denote concatenation

leak, for example (if f is strongly one way): $x \circ r$. Using this we can define the following:

Definition 30. *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a one way function. A predicate $b : \{0,1\}^* \to \{0,1\}$ is a **hard-core of** f if:*

- *b is computable in polynomial time.*
- *For every polynomial time probabilistic algorithm E, the following is negligible:*

$$\Pr[E(f(U_n)) = b(U_n)] - \frac{1}{2}$$

In other words, any algorithm that aims to compute $b(x)$ by only having access to $f(x)$ will succeed with probability only negligibly better than guessing. As you have seen in the bit commitment section, hard cores are really useful, and it turns out [14] that the existence of one way functions implies the existence of such hard cores.

B Appendix: Discrete Logarithm

The discrete logarithm problem is a generalization of the real valued logarithm to arbitrary groups. Recall that:

Definition 31. *Let $a, b > 0$. We define $\log_a b$ to be the unique real number such that:*

$$\log_a b = x \iff a^x = b$$

The discrete logarithm generalizes this to arbitrary groups. In particular:

Definition 32. *Let $\mathbb{G}$ be an arbitrary group. If it exists, the discrete logarithm or index of a in base g is denoted by $\log_g a$, it is an integer and has the property that:*

$$\log_g a = x \iff g^x = a$$

A couple of things to note. First of all, it is important to note that at times the discrete logarithm does not exist. For example, since the equation $2^k \equiv 3 \mod 15$ has no solution[56] then there is no $\log_2 3$ in $(\mathbb{Z}_{15})^\times$. In case $\mathbb{G}$ is a cyclic group, and g is a generator then every element $h \in \mathbb{G}$ has a $\log_g h$. While the real valued logarithm is generally easy to compute (for integers there are strategies using a method similar to binary search, for real values Newton Methods yield good approximations), there are no general techniques that, for every group $\mathbb{G}$, efficiently solve the problem. In fact, it is generally believed (and one of the core assumptions of a number of protocols such as Diffie Hellman), that there exist groups where the discrete log problem is not only hard to solve in the worst case, but actually seems to be extremely complicated to solve in the

[56] As $2^2 \equiv 4, 2^3 \equiv 8, 2^4 \equiv 1 \mod 15$, and the cycle repeats

average[57]. In particular, cyclic groups such as $\mathbb{Z}_p^\times$ are often good candidates, even though if $p-1$ has many factors then there are efficient algorithms such as the Pohlig–Hellman [34]. In practice, if modular arithmetic is used, the prime p is chosen so that $p = 2q + 1$ for some other prime q. Using this and using suitably large primes the discrete logarithm problem is considered intractable. The other alternative to modular arithmetic is using elliptic curves. Elliptic curves are defined over a field[58] $\mathbb{F}$. In particular, this field is in practice of the form $\mathbb{Z}_p$ for p prime, and it is defined by the set of points $(x, y) \in \mathbb{F}^2 \cup \{\infty\}$ such that[59]:

$$y^2 = x^3 + \alpha x + \beta$$

For suitable constant parameters α, β, we can define addition in a geometrical manner, having that the sum of some points P, Q is given by first taking the line between the points (or the tangent at P if $P = Q$), finding the other point in which it intersects the curve, and then reflecting it in the x-axis. In particular, this yields us a group which has the following properties:

- We denote $E(\mathbb{F})$ as the elliptic curve over $\mathbb{F}$ with parameters α, β.
- Then we have ∞ as the identity, and we let $P + \infty = \infty + P = P$ for all $P \in E(\mathbb{F})$
- For a given $P = (x, y) \in E(\mathbb{F})$ we define $-P = (x, -y)$, which is easy to check also is an element of $E(\mathbb{F})$. We also let $-\infty = \infty$.
- Letting $P = (x_1, y_1)$, and $Q = (x_2, y_2)$, if $P \neq \pm Q$ we have $P+Q = (x', y')$ where:

$$x' = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \text{ and } y' = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x') - y_1$$

- Instead we have that, if $(x, y) = P \neq -P$, $2P = (x', y')$ where:

$$x' = \left(\frac{3x^2 + \alpha}{2y}\right)^2 - 2x \text{ and } y' = \left(\frac{3x^2 + \alpha}{2y}\right)(x - x') - y$$

This definition is important, especially since the addition laws are easy to implement and optimize, yielding performance often superior to that of modular exponentiation. As an example we have $E(\mathbb{F}_{29})$, with $\alpha = 4, \beta = 20$. This group in particular has 37 points (including of course ∞), such as $(2, 6), (20, 26)$. We can also verify that $(5, 22) + (16, 27) = (13, 6)$, and that $2(5, 22) = (14, 6)$ (and

[57]It can be shown using random self reducibility, that if a hard instance exists than on the average the problem will also be hard

[58]In fact, it is required this field has characteristic not 2 or 3, otherwise the equations becomes more complicated

[59]In fact the one below is one of the possible equations for elliptic curves, there are many with different advantages, but all of the general form: $\alpha_0 y^2 + \alpha_1 xy = \beta_0 x^3 + \beta_1 x^2 + \beta_2 x + \gamma$

of course that closure holds). Using this kind of construction, and well tested[60] parameters such as the ones that NIST recommend [32] we can yield cryptosystems that are resistant to any non quantum attacker that we know of, and in particular we can get some pretty good performance as well.

C Appendix: Complexity Classes

Definition 33. *We say that a language $L \in \mathsf{P}$ if there exists a deterministic TM M and a polynomial $\mathsf{poly}(\cdot)$ such that both the following hold true:*

1. *On input $x \in \{0,1\}^*$, M halts within $\mathsf{poly}(|x|)$ steps.*
2. *$x \in L \iff M(x) = 1$*

Also, if $L \in \mathsf{P}$ we say that L is ***recognizable in polynomial time***

We can now prove the equivalence of our two definition of P, let us call the probabilistic definition as P_{prob} and the classical as simply P.

Proof. Let $L \in \mathsf{P}$, then there exists a Turing Machine M such that the machine outputs one for each $x \in L$ and always halts in polynomially many steps. Construct a probabilistic TM M_1 as follows: copy everything from M, except the transition function, replacing each transition of the form $\phi(s,i) = x$ with $\phi_1(s,i) = (x,x)$. Recall from the definition that on each step the machine M_1 will randomly choose between left and right transition. Since for each transition left and right are always equal, the choice is meaningless and for each transition it will take the same action as M (taking of course the same exact number of steps). Since M recognized L, then M_1 will also, and as such $L \in \mathsf{P}_{prob}$, which implies $\mathsf{P} \subseteq \mathsf{P}_{prob}$.

Conversely, let $L \in \mathsf{P}_{prob}$. Then there exists a Turing Machine M s.t. M halts in polynomial time, and that $x \in L \iff \Pr[M(x) = 1] = 1$. Let l be bound on the running time of M. This implies that, for any $x \in L$, we have that $\forall r \in \{0,1\}^l : M_r(x) = 1$. Construct a machine M_1 as follows, keep everything equal except from the transition function. Replace each transition of the form $\phi(s,i) = (x_0, x_1)$ to one $\phi(s,i) = x_0$. This is equivalent to picking r to be 0^l, and as such M_1 will also output 1 for each element in L. Of course the running time will still be bound by l, and as such $L \in \mathsf{P}$, which in turn implies $\mathsf{P}_{prob} \subseteq \mathsf{P}$

So, since $\mathsf{P} \subseteq \mathsf{P}_{prob}$ and $\mathsf{P}_{prob} \subseteq \mathsf{P}$, it must be that $\mathsf{P} = \mathsf{P}_{prob}$ ☐

Definition 34. *We say that a language $L \in \mathsf{NP}$ if there exists a relation $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$ and a polynomial $p(\cdot)$ such that the following hold:*

1. *R_L is recognizable in polynomial time*
2. *$x \in L \iff$ there exists y s.t. $|y| \le p(|x|)$ and $(x,y) \in R_L$*

[60] And please do not use the ones that I have used here unless you have a particular desire to design an insecure system!

Such a y is a ***witness for membership of*** $x \in L$

In particular, note the bound requirement on the witness y. This ensures that the certificate is not more than polynomially bigger than the problem statement. If this were not the case, then let us take for example the language

$$\texttt{TAUT} = \{\phi : \{0,1\}^n \to \{0,1\} | n \in \mathbb{N}, \forall(x_1, \dots x_n) : \phi(x_1, \dots, x_n) = 1\}$$

i.e. the language of all boolean formulas that return true for every input. If the bounds on the polynomial size of the certificate were to be lifted, then we could allow for a relation language $R_{\texttt{TAUT}} = \left\{(\phi, 1^{2^n}) | n \in \mathbb{N}, \phi \in \texttt{TAUT}\right\}$. Such language is recognizable in polynomial time in the certificate (as evaluating a boolean formula of n variable takes polynomial time in the formula length, and enumerating all possible n boolean variables takes $\mathcal{O}(2^n)$ which is polynomial in the length of the certificate), however, if the bound is upheld we have no evidence of the belonging of TAUT in NP (in fact, we know that $\texttt{TAUT} \in$ co-NP, and the relation between the two classes is still not clear).

We can similarly prove that our previous definition of NP (call it NP_{prob}) is the same as the classical one.

Proof. Let $L \in \mathsf{NP}$, then there exists a relation R_L s.t. that R_L is recognizable in polynomial time, $x \in L \iff \exists(x, y) \in R_L$ and that y is at most polynomial in the size of x. Consider a probabilistic time machine M as follows:

1. On each step randomly guess a bit of the certificate y
2. Randomly decide whether to generate another bit or to run a polynomial time algorithm to decide whether $(x, y) \in R_L$
3. If we recognized $(x, y) \in R_L$, accept, else reject.

Since the certificate exists, and it is polynomial in the size of x, there will always be at least one extremely lucky machine that will guess it, and as such $\Pr[M(x) = 1] > 0$ for $x \in L$. So $L \in \mathsf{NP}_{prob}$, which implies $\mathsf{NP} \subseteq \mathsf{NP}_{prob}$.

Conversely, let $L \in \mathsf{NP}_{prob}$. Then there exists a probabilistic time machine M s.t. $x \in L \iff \Pr[M(x) = y] > 0$. This implies that $\forall x \in L \exists r s.t. M_r(x) = y$. Construct $R_L = \{(x, r) | x \in L \text{ and } M_r(x) = 1\}$. First of all note the R_L is polynomial time recognizable, as one can run $M_r(x)$ and accept depending on the input (this only works since M is guaranteed to be polynomial time). The condition for existence is met by our above reasoning. Finally, since r is bound by the running time of the machine, which is polynomial, the condition for the size of the certificate is met. As such $L \in \mathsf{NP}$, which in turn implies $\mathsf{NP}_{prob} \subseteq \mathsf{NP}$.

So, since $\mathsf{NP} \subseteq \mathsf{NP}_{prob}$ and $\mathsf{NP}_{prob} \subseteq \mathsf{NP}$, it must be that $\mathsf{NP} = \mathsf{NP}_{prob}$

□

It is also to be noted that NP has also a different but equivalent definition as the set of all languages recognized in non deterministic polynomial time.

Next, we can move to NP completeness.

Definition 35. *A language L is polynomial time reducible to a language M if there exists a polynomial time computable function s.t. $x \in L \iff f(x) \in M$. We write in this case $L \leq M$.*

Definition 36. *We say a language L is* NP*-Hard, if* $\forall M \in \mathsf{NP}, M \leq L$.

Definition 37. *We say a language L is* NP*-Complete if $L \in \mathsf{NP}$ and L is* NP*-Hard.*

In particular, problems that are NP-Complete are considered to be the hardest to find solutions to. The classical NP-Complete problem is $\mathtt{SAT}$, i.e. the language of all satisfiable boolean formulae, but the class of such problems is constantly growing [20].

References

[1] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 2016.

[2] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

[4] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES Is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[5] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[6] Arthur M. Jaffe. The millennium grand challenge in mathematics. *Notices of the AMS*, 53(6), 2006.

[7] Oded Goldreich. In a world of P= BPP. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 191–232. Springer, 2011.

[8] Claude E. Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[9] 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 *Conference on the Theory and Application of Cryptology*, pages 628–631. Springer, 1989.

[10] Michael Chambers. ChemIDplus - 0000302272 - XFSBVAOIAHNAPC-NPVHKAFCSA-N - Aconitine [USP] - Similar structures search, synonyms, formulas, resource links, and other chemical information. https://chem.nlm.nih.gov/chemidplus/rn/302-27-2.

[11] Shafi. Goldwasser, Silvio. Micali, and Charles. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, 18(1):186–208, February 1989.

[12] Adi Shamir. IP = PSPACE, October 1992.

[13] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, July 1991.

[14] Oded Goldreich. *Foundations of Cryptography. Vol. 1: Basic Tools.* Cambridge Univ. Press, Cambridge, digitally print. 1. paperback version edition, 2007. OCLC: 255762567.

[15] Provable security - Computational indistinguishability with Example. https://crypto.stackexchange.com/questions/25678/computational-indistinguishability-with-example.

[16] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, October 1988.

[17] Moni Naor, Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Perfect Zero-Knowledge Arguments for NP Using Any One-Way Permutation. *Journal of Cryptology*, 11(2):87–108, March 1998.

[18] Moni Naor, Harry Road, and San-Jose Ca. Bit Commitment Using Pseudo-Randomness. page 10.

[19] BlumManuel. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, January 1983. PUB27 New York, NY, USA.

[20] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[21] Oded Goldreich. A Methodology of Cryptographic Protocol Design. page 15.

[22] Eduardo Morais, Tommy Koens, Cees van Wijk, and Aleksei Koren. A survey on zero knowledge range proofs and applications. *SN Applied Sciences*, 1(8):946, July 2019.

[23] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

[24] David Chaum, Jan-Hendrik Evertse, and Jeroen van de Graaf. An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology — EUROCRYPT' 87*, Lecture Notes in Computer Science, pages 127–141, Berlin, Heidelberg, 1988. Springer.

[25] 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*, Lecture Notes in Computer Science, pages 186–194, Berlin, Heidelberg, 1987. Springer.

[26] Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Hash function requirements for Schnorr signatures. *Journal of Mathematical Cryptology*, 3(1), January 2009.

[27] Thomas Wu. The Secure Remote Password Protocol. In *In Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1997.

[28] Torben Pryds Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, volume 576, pages 129–140. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992.

[29] Wenbo Mao. Guaranteed correct sharing of integer factorization with offline shareholders. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, Lecture Notes in Computer Science, pages 60–71, Berlin, Heidelberg, 1998. Springer.

[30] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, Lecture Notes in Computer Science, pages 174–187, Berlin, Heidelberg, 1994. Springer.

[31] Fabrice Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, Lecture Notes in Computer Science, pages 431–444, Berlin, Heidelberg, 2000. Springer.

[32] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, and Richard Davis. Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography. Technical Report NIST Special Publication (SP) 800-56A Rev. 3, National Institute of Standards and Technology, April 2018.

[33] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *Journal of the ACM (JACM)*, 32(1):229–246, January 1985.

[34] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms overGF(p)and its cryptographic significance (Corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, January 1978.