ZER-KNOWLEDGE AND INTERACTIVE PROOFS

JOSHUA KOO

Contents

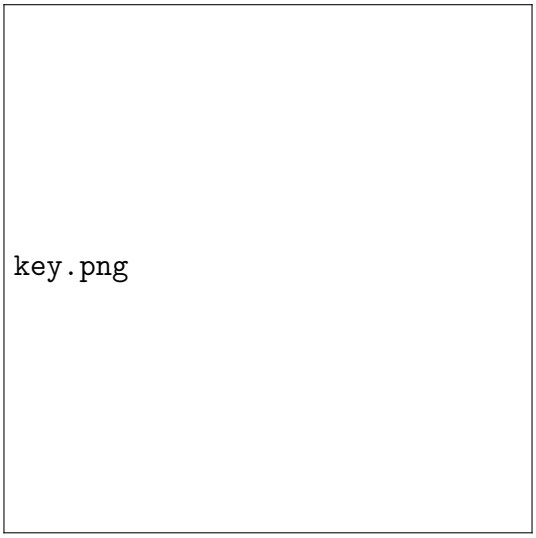
1. Introduction

In the topics of computational complexity theory and cryptography, interactive and zero-knowledge proof systems represent computation as a communication process between two parties: a prover and a verifier. These parties engage in message exchanges to determine whether a particular string belongs to a language or not. The prover is equipped with unrestricted computational capabilities but lacks trustworthiness, while the verifier possesses limited computational resources but is consistently assumed to be honest. The verifier and prover exchange messages until the verifier obtains a solution and becomes sufficiently persuaded of its correctness.

2. The Diffie-Hellman Key Exchange

One of the most basic but important concepts in cryptography is the Diffie-Hellman Key Exchange, a mathematical method of creating a cryptographic key to exchange information between two parties in a secure manner. The goal of the exchange is so that even if you know

all the public info, not knowing any of the private info makes it near impossible to construct the key, while knowing at least one piece of private info allows you to do so. To see how the exchange works, let us label the two parties as Alice and Bob. Together, they will first pick some prime number p. They then pick a base g such that g is a primitive root modulo p. Then, in secret, Alice picks a number a, and similarly, Bob picks a number b. Alice then computes $g^a \pmod p$ and sends it out to the public including Bob (this is public information) and Bob similarly computes $g^b \pmod p$ and sends it to Alice. Alice now knows the number $g^b \pmod p$ so she can now compute $(g^b)^a = g^{ab} \pmod p$. In a similar manner, Bob can also compute $g^{ab} \pmod p$. Here, $g^{ab} \pmod p$ is the private key that can be used to encrypt messages that the public does not know. This is because even if you know $g^a \pmod p$ or $g^b \pmod p$, it is very difficult to figure out $g^{ab} \pmod p$. The figure below shows a conceptual idea of how the cryptographic aspect of how the system works.



3. The Discrete Logarithm Problem

Instead of modulo, we will now look at integers.

4. RSA

RSA (Rivest-Shamir-Adleman) is a widely used public-key cryptography system that was invented by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. It is named after the initials of its inventors. RSA is widely used in various applications, including secure communication, digital signatures, and secure data transmission.

RSA is based on the mathematical problem of factoring large composite numbers into their prime factors. The security of RSA relies on the difficulty of factoring large numbers, which is currently believed to be a computationally difficult problem for classical computers. This makes RSA a popular choice for secure communication.

The RSA system involves three main steps: key generation, encryption, and decryption. Let's go through each step in detail:

(1) Key Generation:
   - Choose two distinct prime numbers, p and q. These primes should be large, typically several hundred digits long, to ensure security. Calculate the modulus, n, by multiplying p and q: $n = p*q$. Calculate Euler's totient function, $\phi(n)$, where $\phi(n) = (p-1)*(q-1)$. Euler's totient function counts the number of positive integers less than n that are coprime (have no common factors) with n.
   - Choose an encryption exponent, e, which is a positive integer greater than 1 and less than $\phi(n)$. e should also be coprime with $\phi(n)$.
   - Calculate the decryption exponent, d, which is the modular multiplicative inverse of e modulo $\phi(n)$. In other words, d is the value that satisfies the equation: $(d*e) \pmod{\phi(n)} = 1$.
   - The public key consists of the modulus, n, and the encryption exponent, e. The private key consists of the modulus, n, and the decryption exponent, d.

(2) Encryption: To encrypt a message, represent it as a number, m, such that $0 \leq m < n$. The message should be smaller than the modulus. Apply the encryption function: $c = (m^e) \pmod{n}$. The ciphertext, c, is the result of the encryption process. It can be transmitted over an insecure channel.

(3) Decryption: To decrypt the ciphertext, c, apply the decryption function:$m = (c^d) \pmod{n}$. The resulting value, m, is the original plaintext message. The security of RSA lies in the difficulty of factoring the modulus, n, into its prime factors. It is computationally infeasible to determine the private key, d, without knowing the prime factors of n. As long as the private key remains secret, the encryption is considered secure.

It is worth noting that the size of the modulus, n, and the choice of prime numbers are crucial for the security of RSA. If the primes are too small, the system becomes vulnerable to attacks that exploit

the factorization process. Therefore, the selection of large primes is essential to ensure the strength of the RSA system.

5. Zero Knowledge Proofs

Zero-knowledge proofs (ZKPs) are a revolutionary cryptographic concept that allows one party, the prover, to convince another party, the verifier, of the truth of a statement without revealing any additional information beyond the validity of the statement. In other words, zero-knowledge proofs enable the prover to demonstrate knowledge of a secret without disclosing any details about the secret itself.

The concept of zero-knowledge proofs was first introduced by Shafi Goldwasser, Silvio Micali, and Charles Rackoff in 1985. Since then, they have become a fundamental tool in cryptography and have found applications in various domains, including secure authentication, digital signatures, privacy-preserving protocols, and blockchain technology.

The fundamental idea behind zero-knowledge proofs is to establish trust and verify the correctness of a statement or claim without revealing any sensitive information. This concept is particularly valuable in scenarios where privacy and confidentiality are of utmost importance. For example, consider a scenario where Alice wants to prove to Bob that she knows a password without actually disclosing the password itself. Zero-knowledge proofs enable Alice to convince Bob of her knowledge without revealing any information that could be exploited by a malicious party. To achieve zero-knowledge, the prover must demonstrate three crucial properties:

- Completeness: The prover should be able to convince the verifier that the statement is true. If the statement is indeed valid, the verifier should accept the proof with high probability.
- Soundness: If the statement is false, the prover should not be able to convince the verifier of its validity except with negligible probability. Soundness ensures that an untruthful prover cannot succeed in deceiving the verifier.
- Zero-knowledge: The proof should not reveal any additional information beyond the validity of the statement. Even after interacting with the prover, the verifier should gain no knowledge about any secret information.

Zero-knowledge proofs are typically interactive (which we will be going into more later), meaning that the prover and the verifier engage in a series of computational steps to establish trust. The interaction involves multiple rounds of communication, where the prover convinces

the verifier by responding to challenges or queries without revealing any sensitive information.

Various techniques and protocols have been developed to construct zero-knowledge proofs. These include simulation-based proofs, algebraic proofs, and non-interactive proofs using cryptographic hash functions. Simulation-based proofs involve the prover constructing a simulated proof, which is indistinguishable from a genuine proof, without revealing any secrets. Algebraic proofs rely on mathematical properties and transformations to demonstrate knowledge of a secret without revealing it explicitly. Non-interactive zero-knowledge proofs leverage cryptographic hash functions to create proofs that can be verified without requiring direct communication between the prover and the verifier. We will now go over an example of a zero knowledge proof.

6. Ali Baba Curve

There is a well-known story presenting the fundamental ideas of zero-knowledge proofs, first published in 1990 by Jean-Jacques Quisquater and others in their paper "How to Explain Zero-Knowledge Protocols to Your Children".[6] Using the common Alice and Bob anthropomorphic thought experiment placeholders, the two parties in a zero-knowledge proof are Peggy as the prover of the statement, and Victor, the verifier of the statement.

In this story, Peggy has uncovered the secret word used to open a magic door in a cave. The cave is shaped like a ring, with the entrance on one side and the magic door blocking the opposite side. Victor wants to know whether Peggy knows the secret word; but Peggy, being a very private person, does not want to reveal her knowledge (the secret word) to Victor or to reveal the fact of her knowledge to the world in general.

They label the left and right paths from the entrance A and B. First, Victor waits outside the cave as Peggy goes in. Peggy takes either path A or B; Victor is not allowed to see which path she takes. Then, Victor enters the cave and shouts the name of the path he wants her to use to return, either A or B, chosen at random. Providing she really does know the magic word, this is easy: she opens the door, if necessary, and returns along the desired path.

However, suppose she did not know the word. Then, she would only be able to return by the named path if Victor were to give the name of the same path by which she had entered. Since Victor would choose A or B at random, she would have a 50

Thus, if Peggy repeatedly appears at the exit Victor names, he can conclude that it is extremely probable that Peggy does, in fact, know the secret word.

One side note with respect to third-party observers: even if Victor is wearing a hidden camera that records the whole transaction, the only thing the camera will record is in one case Victor shouting "A!" and Peggy appearing at A or in the other case Victor shouting "B!" and Peggy appearing at B. A recording of this type would be trivial for any two people to fake (requiring only that Peggy and Victor agree beforehand on the sequence of A's and B's that Victor will shout). Such a recording will certainly never be convincing to anyone but the original participants. In fact, even a person who was present as an observer at the original experiment would be unconvinced, since Victor and Peggy might have orchestrated the whole "experiment" from start to finish.

Further, if Victor chooses his A's and B's by flipping a coin on-camera, this protocol loses its zero-knowledge property; the on-camera coin flip would probably be convincing to any person watching the recording later. Thus, although this does not reveal the secret word to Victor, it does make it possible for Victor to convince the world in general that Peggy has that knowledge—counter to Peggy's stated wishes. However, digital cryptography generally "flips coins" by relying on a pseudo-random number generator, which is akin to a coin with a fixed pattern of heads and tails known only to the coin's owner. If Victor's coin behaved this way, then again it would be possible for Victor and Peggy to have faked the experiment, so using a pseudo-random number generator would not reveal Peggy's knowledge to the world in the same way that using a flipped coin would.

Notice that Peggy could prove to Victor that she knows the magic word, without revealing it to him, in a single trial. If both Victor and Peggy go together to the mouth of the cave, Victor can watch Peggy go in through A and come out through B. This would prove with certainty that Peggy knows the magic word, without revealing the magic word to Victor. However, such a proof could be observed by a third party, or recorded by Victor and such a proof would be convincing to anybody. In other words, Peggy could not refute such proof by claiming she colluded with Victor, and she is therefore no longer in control of who is aware of her knowledge.

7. Hamiltonian Cycles

The following scheme is due to Manuel Blum.[10]

In this scenario, Peggy knows a Hamiltonian cycle for a large graph G. Victor knows G but not the cycle (e.g., Peggy has generated G and revealed it to him.) Finding a Hamiltonian cycle given a large graph is believed to be computationally infeasible, since its corresponding decision version is known to be NP-complete. Peggy will prove that she knows the cycle without simply revealing it (perhaps Victor is interested in buying it but wants verification first, or maybe Peggy is the only one who knows this information and is proving her identity to Victor).

To show that Peggy knows this Hamiltonian cycle, she and Victor play several rounds of a game.

At the beginning of each round, Peggy creates H, a graph which is isomorphic to G (i.e. H is just like G except that all the vertices have different names). Since it is trivial to translate a Hamiltonian cycle between isomorphic graphs with known isomorphism, if Peggy knows a Hamiltonian cycle for G she also must know one for H. Peggy commits to H. She could do so by using a cryptographic commitment scheme. Alternatively, she could number the vertices of H. Next, for each edge of H, on a small piece of paper, she writes down the two vertices that the edge joins. Then she puts all these pieces of paper face down on a table. The purpose of this commitment is that Peggy is not able to change H while, at the same time, Victor has no information about H. Victor then randomly chooses one of two questions to ask Peggy. He can either ask her to show the isomorphism between H and G (see graph isomorphism problem), or he can ask her to show a Hamiltonian cycle in H. If Peggy is asked to show that the two graphs are isomorphic, she first uncovers all of H (e.g. by turning over all pieces of papers that she put on the table) and then provides the vertex translations that map G to H. Victor can verify that they are indeed isomorphic. If Peggy is asked to prove that she knows a Hamiltonian cycle in H, she translates her Hamiltonian cycle in G onto H and only uncovers the edges on the Hamiltonian cycle. This is enough for Victor to check that H does indeed contain a Hamiltonian cycle. It is important that the commitment to the graph be such that Victor can verify, in the second case, that the cycle is really made of edges from H. This can be done by, for example, committing to every edge (or lack thereof) separately.

Completeness

If Peggy does know a Hamiltonian cycle in G, she can easily satisfy Victor's demand for either the graph isomorphism producing H from G (which she had committed to in the first step) or a Hamiltonian

cycle in H (which she can construct by applying the isomorphism to the cycle in G).

Zero-knowledge

Peggy's answers do not reveal the original Hamiltonian cycle in G. Each round, Victor will learn only H's isomorphism to G or a Hamiltonian cycle in H. He would need both answers for a single H to discover the cycle in G, so the information remains unknown as long as Peggy can generate a distinct H every round. If Peggy does not know of a Hamiltonian cycle in G, but somehow knew in advance what Victor would ask to see each round then she could cheat. For example, if Peggy knew ahead of time that Victor would ask to see the Hamiltonian cycle in H then she could generate a Hamiltonian cycle for an unrelated graph. Similarly, if Peggy knew in advance that Victor would ask to see the isomorphism then she could simply generate an isomorphic graph H (in which she also does not know a Hamiltonian cycle). Victor could simulate the protocol by himself (without Peggy) because he knows what he will ask to see. Therefore, Victor gains no information about the Hamiltonian cycle in G from the information revealed in each round.

Soundness

If Peggy does not know the information, she can guess which question Victor will ask and generate either a graph isomorphic to G or a Hamiltonian cycle for an unrelated graph, but since she does not know a Hamiltonian cycle for G she cannot do both. With this guesswork, her chance of fooling Victor is 2n, where n is the number of rounds. For all realistic purposes, it is infeasibly difficult to defeat a zero-knowledge proof with a reasonable number of rounds in this way.

8. Interactive Proofs

NP is the class of languages decidable by a nondeterministic Turing machine in polynomial time. It is also equivalent to the class of languages verifiable in polynomial time. For the rest of this paper, it will be useful to think of NP languages not in terms of nondeterminism, but in terms of verification. We can view NP as the set of languages for which a supercomputer can provide a certificate to a deterministic polynomial time machine. No languages are added to NP if we extend this definition to allow the deterministic polynomial time machine to talk back and forth. This is because the supercomputer is capable of simulating the polynomial machine, anticipating its questions, and providing its answers all at the beginning. Thus NP is the language obtained by permitting interaction with a supercomputer whose responses

must be verifiable. Similarly, we may view BPP as the result of augmenting a deterministic polynomial time machine with an additional capability: randomness. Consider the following diagram, where right arrows denote the result of introducing randomness, and up arrows denote the result of introducing interaction:

Thus NP results from adding interaction to P, and similarly, BPP results from adding randomness to P. What happens when we add both interaction and randomness to P? We will define the result to be IP, the class of languages decidable by an interactive proof system.

To be more precise, we must formally define what is meant by an interactive proof system. A message history is a string of the form $m_1\#m_2\#\ldots\#m_i$, where the m_i denote successive messages in a dialogue. A verifier V is a probabilistic, polynomial-time computable function such that given an input string w and a message history $m_1\#m_2\#\ldots\#m_i$, it computes $V(w, m_1\#m_2\ldots\#m_i) = m_{i+1}$ for even i. Also, V must accept or reject after a polynomial number of interactions. Note that m_{i+1} is not deterministic, but determined only up to a probability distribution. A prover P is an arbitrary function that computes $P(w, m_1\#m_2\#\ldots\#m_i) = m_{i+1}$ for odd i, with the only restriction that m_{i+1} is of polynomial length.

Given a prover P and a verifier V, they interact by alternately augmenting the message history until V accepts or rejects. We denote this interaction by $V \longleftrightarrow P$. Note that any interaction requires a polynomial number of steps of the verifying machine. Given an input string w, there is a probability associated with $V \longleftrightarrow P$ accepting w. We denote this probability by $Pr[V \longleftrightarrow P \text{ accepts } w]$. Formally, a language $L \in$ IP if there exists a verifier V such that:

9. Non-Isomorphism

10. PSPACE

PSPACE is the set of all decision problems that can be solved by a Turing machine using a polynomial amount of space. Formally, we will define PSPACE as the following:

$$\text{PSPACE} = \bigcup_{k\in\mathbb{N}} \text{SPACE}(n^k)$$

Let us go over an example of a complete PSPACE problem, namely the quantified Boolean formula problem (QBF).

11. QBF

12. IP = PSPACE

From here, one can try to prove that IP = PSPACE. The proof is far too long for this paper, but you can read more about it from Introduction the Theory of Computation by Michael Sipser.

Here is a general idea:

To prove that Interactive Proofs (IP) is equal to the complexity class PSPACE, we need to show two things: that any problem in IP can be solved by a PSPACE machine, and that any problem in PSPACE can be solved by an Interactive Proof System.

First, let's prove that any problem in IP can be solved by a PSPACE machine.

Proof: Suppose we have a problem P that belongs to the class IP. This means that there exists an Interactive Proof System (IPS) for problem P. The IPS consists of a prover P and a verifier V.

To show that P is in PSPACE, we need to design a PSPACE machine M that solves problem P. The PSPACE machine M will simulate the interaction between the prover P and the verifier V.

The simulation proceeds as follows:

The PSPACE machine M simulates the actions of the verifier V by making all the necessary queries to the prover P. The prover P responds to the queries made by M. M keeps track of the messages exchanged between P and V and uses this information to make further queries. The simulation continues until the verifier V reaches a decision. Since the verifier V operates in polynomial space in the original Interactive Proof System, the PSPACE machine M can simulate this process in polynomial space as well. Therefore, any problem in IP can be solved by a PSPACE machine, proving that IP is a subset of PSPACE.

Next, let's prove that any problem in PSPACE can be solved by an Interactive Proof System.

Proof: Suppose we have a problem P that belongs to the class PSPACE. This means that there exists a polynomial space machine M that solves problem P.

To show that P is in IP, we need to design an Interactive Proof System (IPS) for problem P. The IPS consists of a prover P and a verifier V.

The IPS proceeds as follows:

The verifier V generates a random string and sends it to the prover P. The prover P uses this random string to compute the response. V verifies the response by simulating the computation of the polynomial space machine M on the input, using the random string and the prover's

response. V reaches a decision based on the verification. Since the polynomial space machine M can be simulated by the verifier V in the Interactive Proof System, any problem in PSPACE can be solved by an Interactive Proof System. Thus, PSPACE is a subset of IP.

Combining both proofs, we have shown that IP is a subset of PSPACE and PSPACE is a subset of IP. Therefore, IP = PSPACE.

In conclusion, we have proven that Interactive Proofs (IP) is equal to the complexity class PSPACE.

Euler Circle, Mountain View, CA 94040