ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ

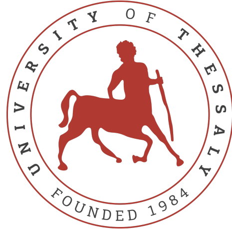
Μία Μελέτη των Αποδείξεων Μηδενικής Γνώσης και οι Εφαρμογές τους στις Βάσεις Δεδομένων

Σταύρος Κασαράς

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
Γεώργιος Δημητρίου

Λαμία, 31 Μαΐου 2022

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14



UNIVERSITY OF THESSALY

SCHOOL OF SCIENCE

INFORMATICS AND COMPUTATIONAL BIOMEDICINE

A Study on Zero-knowledge Proofs and their Applications on Databases

Stavros KASSARAS

Master Thesis

Supervisor
Dr. Dimitriou GEORGIOS

Lamia, May 31, 2022

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ

ΚΑΤΕΥΘΥΝΣΗ:
«ΠΛΗΡΟΦΟΡΙΚΗ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΑΣΦΑΛΕΙΑ, ΔΙΑΧΕΙΡΙΣΗ ΜΕΓΑΛΟΥ ΟΓΚΟΥ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ»

Μία Μελέτη των Αποδείξεων Μηδενικής Γνώσης και οι Εφαρμογές τους στις Βάσεις Δεδομένων

Σταύρος Κασαράς

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
Γεώργιος Δημητρίου

Λαμία, 31 Μαΐου 2022

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

«Υπεύθυνη Δήλωση μη λογοκλοπής και ανάληψης προσωπικής ευθύνης»

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, και γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα και ενυπογράφως ότι η παρούσα εργασία με τίτλο «Μία Μελέτη των Αποδείξεων Μηδενικής Γνώσης και οι Εφαρμογές τους στις Βάσεις Δεδομένων» αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές από τις οποίες χρησιμοποίησα δεδομένα, ιδέες, φράσεις, προτάσεις ή λέξεις, είτε επακριβώς (όπως υπάρχουν στο πρωτότυπο ή μεταφρασμένες) είτε με παράφραση, έχουν δηλωθεί κατάλληλα και ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο ΔΗΛΩΝ
Σταύρος Κασαράς

Ημερομηνία
31 Μαΐου 2022

Υπογραφή

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Σταύρος Κασαράς

Τριμελής Επιτροπή:

1. Γεώργιος Δημητρίου
2. Αντώνιος Δαδαλιάρης
3. Νικόλαος Τζιρίτας

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

UNIVERSITY OF THESSALY

Abstract

Informatics and Computational Biomedicine
School of Science

Master of Science

A Study on Zero-knowledge Proofs and their Applications on Databases

by Stavros KASSARAS

In 1985 Goldwasser, Micali and Rackoff conceived first the notion of zero-knowledge proofs in their paper *"The Knowledge Complexity of Interactive-Proof Systems"* [GMR85]. Since then, there have been important advances but, anyone coming around the ideas of ZKPs meets a view of nebulousness and abstractness. The reason is, the topic of ZKPs is orders of magnitude more complex than usual cryptographic primitives. The purpose of this thesis is twofold. One, to introduce in a graduate (and undergraduate) level the student to the notions of ZKPs. The present treatment here is best described as informal in the language used but nonetheless the math notation (in definitions, theorems and proofs) is formal. Secondly, we use zero-knowledge databases as a long example of presenting in practice the concepts treated in the previous chapters. There are of course modern protocols (such as zk-SNARKs and Bulletproofs), but we think such complex examples can discourage the student's task of learning a new cryptographic primitive. We further contribute in this thesis the zero-knowledge aspect of a graph database in contrast to an elementary database.

This work is organized in 5 chapters. The first Chapter is introductory. In this chapter the style is conversational, communicating the motivation behind the formal definitions of ZKPs. In Chapter 2, we present the definitions of interactive proof systems and zero-knowledge property along with the example protocol of Quadratic Residuosity. In Chapter 3, we treat an important class of zero-knowledge protocols called Sigma Protocols which offer a smooth introduction to the notion of Proofs of Knowledge. In Chapter 4, we discuss additional topics such as composition of proofs, commitment schemes and non interactivity for constructing ZKPs, concluding the chapter with the famous result that any $\mathcal{NP}$-statement can be proven in zero-knowledge. In Chapter 5, we couple the notions learned in the foregoing chapters with the implementation of elementary databases. In Chapter 6 we present our contribution of zero-knowledge graph databases. Appendix A presents an unrestricted model of ZKPs allowing individual local inputs. We placed the proof of Schnorr's protocol in Appendix B so these technicalities do not cause discontinuity with the theory. Last, in Appendix C, we included relevant number theory facts and pseudo algorithms to support the examples presented in the thesis.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Contents

Abstract vii

Preface 1

1 Introduction to Zero-knowledge Proofs 3
- 1.1 Graphical examples . . . 4
  - 1.1.1 Ali Baba's Cave . . . 4
  - 1.1.2 Where's Waldo? . . . 6
- 1.2 Deeper in the Ali Baba's Cave . . . 6
  - 1.2.1 Not a Math Proof, Yet a Proof . . . 6
  - 1.2.2 Completeness and Soundness . . . 7
  - 1.2.3 Proofs and Complexity . . . 7
  - 1.2.4 What about Zero-knowledgeness? . . . 7
  - 1.2.5 Putting it all Together . . . 8
- 1.3 An Example Protocol . . . 8
  - 1.3.1 Quadratic Residue . . . 8
  - 1.3.2 Modular Arithmetic . . . 8
  - 1.3.3 Protocol Description . . . 9

2 Formal Definition of Zero Knowledge Proofs 11
- 2.1 Preliminaries . . . 11
  - 2.1.1 Elementary Algebra . . . 11
  - 2.1.2 Mathematical Notation . . . 12
  - 2.1.3 Big-O notation . . . 12
- 2.2 Interactive Proof Systems . . . 12
  - 2.2.1 Formal Definition . . . 13
  - 2.2.2 Quadratic Residue Revisited . . . 16
- 2.3 Zero-knowledge Definitions . . . 17
  - 2.3.1 Perfect Zero-knowledge . . . 17
  - 2.3.2 Computational Zero-knowledge . . . 19
  - 2.3.3 Statistical Zero-knowledge . . . 20

3 Σ-Protocols 25
- 3.1 The Discrete Logarithm problem in Cryptography . . . 25
- 3.2 DLP for ZKPs . . . 26
  - 3.2.1 Definitions . . . 27
  - 3.2.2 Proofs of knowledge . . . 28
- 3.3 *AND* and *OR* Compositions . . . 34
  - 3.3.1 Conjunction Composition $\mathcal{P}_{AND}$ . . . 34
  - 3.3.2 Disjunction Composition $\mathcal{P}_{OR}$ . . . 34

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

4 Additional Topics on Zero-knowledge Proofs **37**
4.1 Composition of Zero-knowledge Proof Systems 37
4.1.1 Sequential Composition 37
4.1.2 Parallel Composition 38
4.1.3 Witness Indistinguishability (WI) 39
4.1.4 Concurrent Composition 40
4.2 Commitment Schemes 40
4.2.1 The 3-Coloring Problem 40
4.2.2 Defining Commitment Schemes 42
4.3 Non-Interactive proofs 43
4.3.1 Walkthrough of an Illustrated NIZK (Sudoku Protocol) 43
4.3.2 From Examples to Practice 44
4.4 $\mathcal{NP}$ Problems and $\mathcal{ZKP}$ Results 46

5 Zero-knowledge Databases **49**
5.1 Key-value Databases 49
5.2 Building Blocks 49
5.2.1 Elementary Databases 50
5.2.2 Zero-knowledge EDBs (ZK-EDBs) 50
5.2.3 Pedersen's Commitment Scheme 51
5.2.4 Binary Trees 51
5.3 Construction 52
5.3.1 Merkle Tree (Committing steps) 53
5.4 Proving Database Values 54
5.4.1 Proof Construction of $D(x) = y$ 54
5.4.2 Proof Verification of $D(x) = y$ 55
5.4.3 The case $D(x) = \perp$ 56
5.4.4 Dealing with Collisions 56
5.5 On Soundness and Zero-knowledge 57
5.6 The Necessity for Zero-knowledge Databases 57
5.7 Recent Advances in ZK-EDBs 58

6 Zero-knowledge Graph Databases **61**
6.1 Graph Databases 61
6.1.1 Comparison with Relational Databases 62
6.1.2 The Necessity for Zero-knowledge Graph Databases 62
6.2 Towards Zero-knowledge Graphs 62
6.2.1 Querying Graph Databases 63
6.2.2 Directed and Undirected Graphs 63
6.2.3 Preliminaries 63
6.3 Construction - Undirected Case 64
6.3.1 Complete Graphs 65
6.3.2 Edge Commitment Schemes 66
6.3.3 Committing up to an Edge 67
6.4 Zero-Knowledge Graph Databases 68
6.5 An example of ZK-GD 69
6.5.1 Contraction and Committing Steps 69
6.5.2 Proving Graph Relations 73
6.6 The Case of Bipartite Graphs 74
6.6.1 Bipartite Committing 74
6.7 On Soundness and Zero-Knowledge 76

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

6.8 Communication Complexity 76
6.9 Directed Case 76

7 Conclusions and Open Problems **77**
7.1 Open Questions 77

A Augmented Model **79**

B Schnorr's Protocol: Proof of Sigma Protocol **81**

C Computational Number Theory and Algorithms **83**
C.1 Prime Numbers 83
C.2 On Quadratic Residues 83
C.3 On Discrete Logarithm 84
C.3.1 Computing $\phi(n)$ 85
C.3.2 Computing Primitive Roots 86

Bibliography **89**

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

List of Figures

1.1 Peggy chooses randomly a path between A and B, while Victor is standing outside. . . . 4
1.2 Victor chooses randomly an exit path A or B. . . . 5
1.3 Peggy appears at Victor's requested path. . . . 5

4.1 Information leakage from parallel composition. . . . 38
4.2 An example of a 3-colored graph. . . . 41
4.3 An instance of a standard Sudoku problem and its solution. . . . 43

5.1 An example of a key-value pair. . . . 50
5.2 An example of a $\mathcal{T}_3$ complete labeled binary tree. . . . 51
5.3 Nodes of $H[D]$. . . . 52
5.4 The light shaded vertices compromise the subtree T'. . . . 52
5.5 The light and darkly shaded vertices together compromise the subtree T. . . . 53
5.6 Subtree T with associated values m_x. . . . 53
5.7 Subtree T with associated values h_u. . . . 54
5.8 Merkle Tree of T. . . . 54
5.9 Authentication path of $H(x)$. . . . 55
5.10 The recursive verification of proof π_x. . . . 55
5.11 Anti-pradigm of u betraying D's support. . . . 56

6.1 An example of a Graph Database. . . . 61
6.14 An example of a Bipartite Graph. . . . 74

C.1 Non-interactive proof of discrete logarithm. . . . 88

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

List of Algorithms

1 Prime Root Test . . . 84
2 Extended Euclidean Algorithm (Recursive) . . . 85
3 Euler's Totient Function . . . 86
4 Primitive Root Algorithm . . . 87

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Preface

In cryptography we are concerned with the construction of security schemes that can withstand any kind of abuse. When building a security system we should neither limit it by its environment of operation nor make it impenetrable to a specific class of attacks. There are no utopian cryptographic schemes that can ever satisfy both untypical environment states and hold up against the strategies of an adversary. Therefore, the only assumptions we allowed to have concern the computational abilities of the adversary. More precisely, we care whether or not an adversary can *efficiently* extract information over insecure media.

Aside from adversaries, security protocols are based on trusted users. Modern cryptography has past the point of encryption schemes operating with equal encryption and decryption keys to the abstract point which in every security problem we wish to limit the effects of dishonest users. Simultaneity problems, that is, the simultaneous exchange of secrets, require the participation of an external party or the existence of a fully trusted party. The problem of secure implementation of functionalities transforms to the problem of implementing a secure party. The foregoing fault-tolerant protocols make sense only with honest parties, or in other words, with parties that follow exactly a protocol.

Here is where Zero-knowledge proofs come forth. Zero-knowledge proofs (ZKPs for short) is a cryptographic tool that *forces* the users to follow a given protocol properly. Almost everywhere in bibliography ZKPs are presented as a security scheme in which one party, called the Prover, can convince another party, called the Verifier, that some assertion is true without revealing nothing else than the validity of the assertion. Even if both the parties are distrustful to one another, ZKPs provide the means to disclosing specific pieces of information.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Chapter 1

Introduction to Zero-knowledge Proofs

Zero-knowledge proofs definition is seemingly contradictory in nature; they have the bizarre property of convincing about the validity of an assertion and at the same time yielding nothing more than the fact of its validity. In general, ZKPs fall under a cryptographic domain of *privacy-preserving* communication protocols; protocols in which the exchange of information between distressful parties is guaranteed to be computed in conformity with a predetermined operation while concealing any secret knowledge.

There are several privacy-preserving protocols that have been studied. Amongst them, the most well known is Shamir's threshold scheme [Sha79]. In this scheme we partition the data D into n pieces so any k pieces form the initial data D, but fewer than k pieces reveal totally no information about D. This scheme is called a (k, n) threshold scheme, since k minimum pieces are required to reconstruct D. This permits a mutual sharing of ownership of a singe secret to different parties, each one of them owning a different piece (or share) of the secret. Each individual's share is useless on its own but to reconstruct the original message, even if all parties have conflicted interests, they must cooperate. Additionally, the threshold scheme is *information-theoretic secure*, even an adversary with unlimited computational power would not be able to break the encrypted secret.

The concept of a fuzzy vault was developed by Juels and Sudan [JS02] to conceal a secret value. A party lays a secret value k in a fuzzy vault and secures it in terms of elements of a set A chosen from a public universe U (e.g., generating a cyphertext C_A representing an encryption of κ under the set A). Another party can "unlock" the vault by using a set B of the same length and obtain k only if A and B have a narrow gap, i.e., B covers in great extend the set A. This scheme is secure against a computationally unbounded adversary.

Another privacy-preserving scheme is k-anonymity. k-anonymity, likewise (k,n) threshold scheme, averts re-identification of data to fewer than k data items and additionally the data produced by two sources are not exposed to each other. In [JC06] is presented a two party framework that maintains the benefits of partitioning data while integrates k-anonymous data that does not violates privacy through disclosure to the data holders.

In [McL90] is developed a security model that is based purely on information flow rather than an information sharing based theory. In the information based-model the information flows from low-level objects to high-level object and not vice-versa (where "object" is used in its broadest sense including e.g., files, programs, etc.) - an upward only flow of information.

We choose in introduction not to procced directly to the formal definition of ZKPs,

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

but rather we adopt a constructive way. We start with the abstract idea of a Zero-knowledge proof and we will meet an "informal" definition (the formal presentation will follow in Chapter 2). The reason for this naive approach is that ZKPs combine a number of notions together - regarded as a body of notions - some of which justify their use but others do not make obvious their working. The informal style points on being more conversational, to familiarize the reader with several notions until the rigorous presentation.

1.1 Graphical examples

Although the formal definition of a Zero-knowledge Proof requires a lot of mathematical toil, before we procced we will introduce some illustrative examples for a smooth transition from a simple presentation of the ideas behind ZKPs to the rigorous formal foundations.

1.1.1 Ali Baba's Cave

The following example is useful in presenting the fundamental ideas of zero-knowledge proofs and will be used as a first analysis and introduction [Qui+90].

Imagine a cave whose entry is separated into two paths, one to the left and the other one to the right. Both passages are connected with a door which opens with a secret word that you enter in a numerical pad. A party named Peggy knows the secret word, but Victor, another party, being suspicious wants to devise an experiment that proves whether Peggy admittedly knows the word or not (Peggy and Victor are characters used respectively for the Prover and Verifier of the statement in a zero-knowledge proof). At the same time Peggy wishes not to disclose the secret word to Victor neither the fact of her knowledge of such secret word to anyone else.

Victor thinks of the following strategy: Initially, both Victor and Peggy are outside of the cave. Victor tells Peggy to go alone in the cave and randomly down to one of the passages A or B while Victor's view is restrained from the path she took.

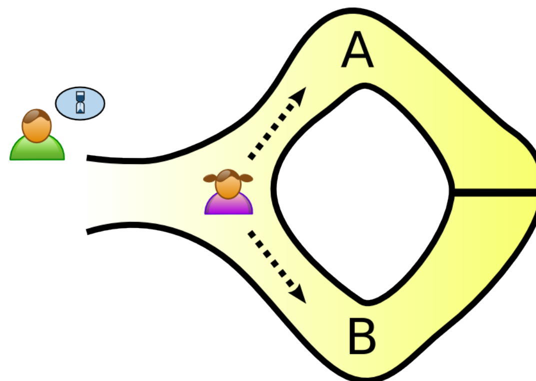


FIGURE 1.1: Peggy chooses randomly a path between A and B, while Victor is standing outside.

Then, Victor enters the cave only as far as the two paths start and he flips a coin. If the coin comes up with heads, Peggy is instructed to come out from the right path otherwise she comes out from the left path.

Peggy could get lucky if Victor call her to come out from the same passage she entered. So Victor decides to repeat the experiment several times, say 20 times in a row. With each new test the chances of success for someone to come out from the

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

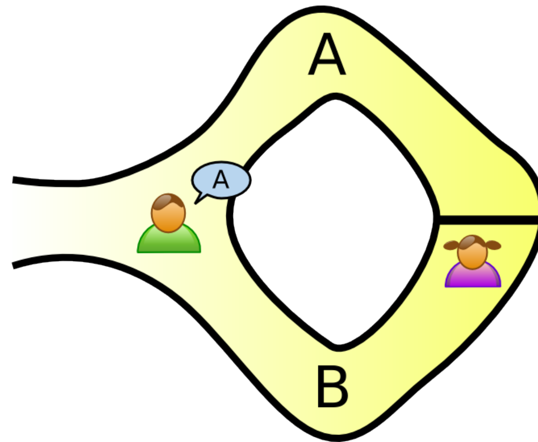


FIGURE 1.2: Victor chooses randomly an exit path A or B.

passage Victor calls, are divided by two. In other words, the chances for someone to come out each time by the required exit are becoming vanishingly small, in our case about one in a million!

Hence, if Peggy constantly appears at the required exit after all Victor's requests, then, with high probability, Peggy really does in fact knows the hidden word and Victor accepts her claim.

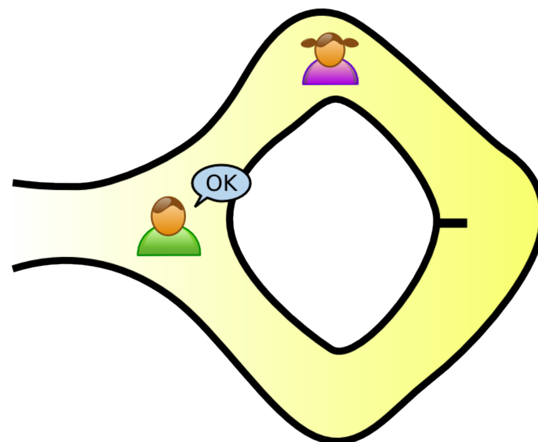


FIGURE 1.3: Peggy appears at Victor's requested path.

So far Peggy has proved to Victor that she knows the secret word, but as we stated in the beginning she wishes not to reveal the fact of her knowledge to the world.

Peggy uses a camera to record the experiment. All the camera is recording is Victor being at the fork of the cave, flipping a coin, calling Peggy to come out from one of the passages according to the coin's outcome and Peggy appearing at the exit requested by Victor. Such recording is trivial for someone to fake. She could hire an actor that looks like her and record the same experiment. It is obvious that the actor does not know the secret word and so about half of the repetitions are failed since the actor does not make it to come out of the requested passage. This is not a problem since Peggy can edit the tape as many times are required and keep only the successful outcomes.

Notice now that you cannot tell apart the two recordings, that is, you cannot distinguish between the genuine tape and the simulated one. Since you do not convey

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

any knowledge of the secret from the simulated tape, the same is true for the genuine tape. So you can not pass the fact of Peggy's knowledge of the secret word, thus no one can be convinced of her knowledge.

1.1.2 Where's Waldo?

Imagine that you are a software developer and you have made an algorithm that takes as input a picture from a *Where's Waldo?* book and outputs Waldo's position in the picture.

A company being interested in a *Where's Waldo?* solver program comes to you with an offer to buy your program, but first they ask for a demonstration that proves your code actually works. At the same time you don't want necessary to do unpaid work for them. Both the company and you are willing to mutually benefit from each other but there is still mistrust between both parties. Fortunately, there is a way without doing free work to prove your algorithm works as it supposes to.

You take a picture from the *Where's Waldo?* book and tell the company's representatives to step out from the room. Then, you use a cardboard which in the center of it is a small cut and you place it on top of the picture so only Waldo's figure is visible from the cut and the cardboard is large enough so it doesn't give away the position of the book. You now ask the company's representatives to come back in the room so they can observe the Waldo's picture and at the same time they cannot determine the position of him in the picture.

1.2 Deeper in the Ali Baba's Cave

1.2.1 Not a Math Proof, Yet a Proof

In the cave example, Peggy was able to prove Victor she knows the secret word. When we hear about a *"proof"* the first thing that comes to our mind is the conventional mathematical proof, a statement that is inferred by previously established statements or original assumptions, called axioms, in a logical way. This view is somewhat strict, "static". The interaction between Peggy and Victor is anything but a math proof, yet Peggy was able to convince Victor about her knowledge of the secret word.

This is the same as proving someone's innocence in a court trial or proving a philosophical statement to someone or a political statement. All these proofs are interactive in nature, that is, they use a dynamic process to establish the truth of a statement via an interaction, an exchange of information. This dynamic interpretation of proof is more suitable for practical implementation in cryptographic protocols and it is central to the non-triviality of ZKPs. The interaction is a verification procedure that merely verifies the validity of a statement. Specifically in our example, Victor's verdict to accept or reject Peggy's claim is probabilistic. Victor can repeat the procedure many times so the probability error can be decreased to be negligible.

In cryptography, interactive proofs are presented in specific ways. One of them is descriptive like below:

- Initially Peggy goes into the cave to the branching point while Victor waits outside.
- Peggy randomly selects path A or B and goes to the door.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

- Victor goes into the cave to the branching point.
- Victor flips a coin and according to the result he calls the path where Peggy should come out.
- Victor accepts when Peggy appears at the exit he called, otherwise he rejects.

1.2.2 Completeness and Soundness

If Peggy did not have the secret word, she would have to try really hard by any means to convince Victor that she knows the secret. In other words, Victor the verifier has the ability to protect himself from being convinced of false statements. On the contrary, when Peggy knows the secret word she can convince Victor for her claim. These two properties, false statements are not accepted and the prover being able to convince the verifier for true statements, are two basic notions regarding every interactive proof and are respectively called *soundness* and *completeness*.

Completeness means the prover is able to generate proofs for valid statements. Soundness means there cannot exist interactive proofs for false statements, or in other words, if a statement can be proved then it is valid.

1.2.3 Proofs and Complexity

Completeness and soundness are two *"computational"* tasks that suggest the existence of two parties, the prover and the verifier. We must note there is an asymmetry in complexity between prover and verifier. The interactive proofs we encounter in practice are based on arithmetic problems where a sort of computational difficulty exists (e.g., is difficult to find for a large number N a factor smaller than a number k, but is relative easy to find it if we know beforehand the factorisation of N). The prover is able to generate a proof for a solution on a "hard" problem and the verifier is the party capable of verifying it. Since the burden of computation is placed on the prover, we assume he has unlimited complexity.

On the contrary, the verification task is easier. So, in every interactive proof we focus in the verification complexity. *"Easy"* in complexity tasks means *whatever can be efficiently verified*. In complexity theory we usually associate efficient computation with deterministic polynomial-time computation which is captured by the complexity class $\mathcal{NP}$. Since in an interactive process coin tosses are allowed and the verifier's verdict of the result is statistical in nature, either accepts or rejects, probabilistic polynomial-time ($\mathcal{PP}$) computations are reflecting (in interactive proofs) efficient computations as a formalization of *whatever can be efficiently verified*.

1.2.4 What about Zero-knowledgeness?

At the start we mentioned the verifier, Victor, does not gain any knowledge about the secret word. We will elaborate a bit more what we mean when we say *"gaining no knowledge"*. Imagine the secret word is very simple, lets say "Abc". If Victor wants to learn the secret word on his own, he could just go to the numerical pad and try different combinations until he finds the secret word. Because "Abc" is a very simple combination, he will find it in short time. If the word is more complicated in nature, lets say "2357OpenSesame1113@!", Victor will have a very difficult time to find it if he uses a brute-force technique.

In the first case, "Abc", Victor determined the word by himself. Even if he engage in a conversation with Peggy, he will gain nothing more that he could not compute

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

by himself, his computational power will not increase. In other words, if whatever the verifier can efficiently compute after interacting with the verifier he can also compute by himself we say that he has *gained no knowledge*. At the second case, Victor cannot efficiently learn anything about the word. In general, the verifier *gains information* if after interacting with the prover his computational ability has increased, e.g. if Victor in some way learns that the first part of the secret word is "2357OpenSesame", then he can easily compute the rest part of the word, whilst he could not compute it before.

1.2.5 Putting it all Together

We are ready now to present an informal definition of Zero-knowledge proof containing all the aforementioned properties.

Definition 1.2.1. A zero-knowledge proof is an interactive proof of a statement involving a Prover and Verifier referring explicitly to the following three computational tasks

- ***Completeness:*** *if the statement is true, an honest Verifier (that is, one following the protocol properly) will be convinced of the truth of the statement by an honest Prover.*
- ***Soundness:*** *if the statement is false, no cheating Prover can convince an honest Verifier that it is true, except with some small probability.*
- ***Zero-knowledge:*** *if the statement is true, no Verifier gains any additional knowledge other than the fact that the statement is true.*

So far we view ZKPs as an interactive proof system - completeness and soundness together, stacking next to it the zero-knowledge property. Another view of ZKPs is pushing aside the notion of interactive proof system and look at the above three properties all together independently from other notions. Completeness states if both the Prover and the Verifier are honest, the protocol will result in success. On the other hand, soundness and zero-knowledge together state if at least one of them is dishonest the protocol fails. We can say that if both parties want to result in success, even if they are distrustful to one another, they have no other way than to follow exactly the protocol or in other words, the protocol "forces" them to follow it properly in order to convince one another.

1.3 An Example Protocol

1.3.1 Quadratic Residue

At this point we will present a famous and simple zero-knowledge proof protocol, also known as Fiat-Shamir identification protocol [FFS88], which uses quadratic residuosity.

Before describing the protocol we will review the necessary mathematical background.

1.3.2 Modular Arithmetic

Definition 1.3.1. Let a, b and $n > 1$ be integers such that they differ by an integer multiple of n, i.e. $a - b = kn$ for some $k \in \mathbb{Z}$. Then, we write

$$a \equiv b \pmod{n}$$

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

and we say that a, b are congruent modulo n (n is called the modulus).

For any pair of integers a and $n > 1$ there is a unique integer r in $\{0, ..., r-1\}$, called the *residue*, such that $a \equiv r \bmod n$. Obviously, if $a \equiv b \pmod{m}$ and $b \equiv c \pmod{m}$ then $a \equiv c \pmod{m}$.

Many of the rules of arithmetic apply to modular arithmetic as well. In particular, if $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $a + c \equiv b + d \pmod{n}$ and $ac \equiv bd \pmod{n}$.

Definition 1.3.2. $\mathbb{Z}_m := \{0, 1, \ldots, m-1\}$. For $a, b \in \mathbb{Z}_m$ we define $a \oplus b$ to be the residue of $(a + b)$ modulo m. Similarly, we define $a \otimes b$ to be the residue of $(a \cdot b)$ modulo m.

Definition 1.3.3. An integer q is called *quadratic residue* modulo n if it is congruent to a perfect square modulo n i.e., if there exists an integer x such that: $x^2 \equiv q \pmod{n}$.

The quadratic residue protocol is: given the product of two primes, prove that an integer is a quadratic residue modulo this number. We select two prime numbers p and q and calculate their product $n = p \cdot q$, this n is used as modulo. The protocol assumes that the extracting of square root of a quadratic residue modulo n is a hard problem without the knowledge of its prime factors.

1.3.3 Protocol Description

A secret number s is selected and we compute its square $v = s^2$, v represents the public key. The prover keeps the secret key s while the verifier gets only the public key v. The prover does not want to reveal his secret number but only prove its possession.

Protocol conversation:

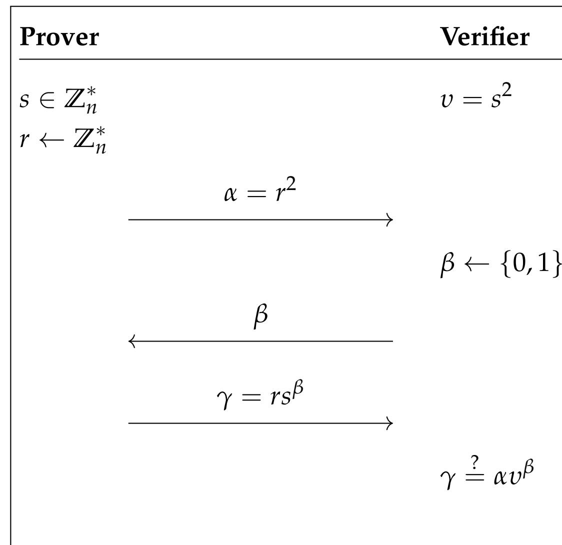
Statement: v is a quadratic residue mod n. **Public input**: v, n. **Prover's private input**: s.

- Prover randomly chooses $r \leftarrow \mathbb{Z}_n^*$ and sends $\alpha = r^2$ to the verifier.
- Verifier chooses random $\beta \leftarrow \{0, 1\}$ and sends it to the prover.
- Prover computes $\gamma = rs^\beta$ and sends it back.

Verification: The verifier accepts by checking if the equality $\gamma^2 = \alpha v^\beta$ holds.

Another way to demonstrate the protocol other than descriptive as above, is using a code block like the one below:

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14



Before we go on to the next chapter try to prove that the described protocol has all three necessary properties, namely completeness, soundness and zero-knowledge.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Chapter 2

Formal Definition of Zero Knowledge Proofs

So far we have derived an informal definition of ZKPs but reaching a definition it does not mean that it can be met. The way to demonstrate that our definition is viable is to construct an actual example. The fact that we can convince a party for the validity of a statement and at the same time gain nothing, is counter-intuitive and thus is not clear that such proofs even exist.

In this chapter we will demonstrate the existence of such proof, the Quadratic Residuosity, which is based on the assumption that is difficult to extract squared roots modulo a positive integer.

2.1 Preliminaries

2.1.1 Elementary Algebra

These notes cover basic notions in algebra which we will be needed for discussing several topics of this thesis.

Groups

Given a set G and a binary operation $*$, if each element in the set obeys the following 4 properties, then the set and its operation $(G, *)$ is called a *group*:

- *Closure*. If $a, b \in G$, then $a * b \in G$.
- *Associativity*. $(a * b) * c = a * (b * c)$ for all $a, b, c \in G$.
- *Existence of identity element*. There is an element $e \in G$, such that $a * e = e * a$ for all $a \in G$.
- *Inverse*. For every $a \in G$, there exists an $a^{-1} \in G$ such that $a * a^{-1} = a^{-1} * a = e$.

If, in addition each pair of elements $a, b \in G$ satisfies the commutative property, $a * b = b * a$, then the group $(G, *)$ is called a *Abelian*.

Fields

Given a set F and binary operations $*$ and $+$, $(F, *, +)$ is called a *field* if the following properties hold:

- $(F, +)$ is an Abelian group with identity element 0.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

- $(F - \{0\}, *)$ is an Abelian group with identity element $1 \neq 0$.
- *Distributive property*. For all $a, b, c \in F, a * (b + c) = (a * b) + (a * c)$.

According to the above definitions we have that $(\mathbb{Z}_m, \oplus)$ is a group, and when m is a *prime number*, $(\mathbb{Z}_m - \{0\}, \otimes)$ is also a group.

This way of writing groups and other structures using symbols like $\oplus$ and $\otimes$ is getting cumbersome. To make math notation lighter, we will prefer $(\mathbb{Z}_m, +)$ instead of $(\mathbb{Z}_m, \oplus)$ and $(\mathbb{Z}_m - \{0\}, \cdot)$ instead of $(\mathbb{Z}_m - \{0\}, \otimes)$. Expressions like $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ and $(a \otimes b) \otimes c = a \otimes (b \otimes c)$ will become $(a + b) + c = a + (b + c)$ and $(ab)c = a(bc)$ respectively and will mean the former ones. We will keep this convention and whenever there is a confusion in notation we will state it differently.

2.1.2 Mathematical Notation

We will use the letter Σ to denote an alphabet, i.e. a finite set of symbols. Commonly used alphabets are the decimal $\{0, 1, \ldots, 9\}$ and the binary alphabet $\{0, 1\}$; especially for the binary alphabet an element $x \in \{0, 1\}$ is named as a bit. An ordered sequence of elements from Σ is a called a *string* (or *word*), and the empty word is often denoted with e. For a string σ, $|\sigma|$ denotes the length of the string and with Σ^k the set of all strings of length k. The union of all sets Σ^*, $\bigcup_{k \geq 0} \Sigma^k$ denotes all finite length strings over the specified alphabet. With L we denote a subset of strings and call it a *language*.

2.1.3 Big-O notation

Let $f, g : \mathbb{N} \to \mathbb{R}$ be functions. We say that:

- $f(n) = O(g(n))$ if there exist $c > 0$ and integer n_0 such that for all $n \geq n_0, f(n) \leq cg(n)$
- $f(n) = \Omega(g(n))$ if there exist $c > 0$ and integer n_0 such that for all $n \geq n_0, f(n) \geq cg(n)$
- $f(n) = \Theta(g(n))$ if there exist $c_1, c_2 > 0$ and integer n_0 such that for all $n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$
- $f(n) = o(g(n))$ if, for all $\epsilon > 0$, there exists integer n_0 such that for all $n \geq n_0, f(n) \leq \epsilon g(n)$
- $f(n) = \omega(g(n))$ if, for all $\epsilon > 0$, there exists integer n_0 such that for all $n \geq n_0, g(n) \leq \epsilon f(n)$

If $f(n)$ is a polynomial of degree k, we have $f(n) = \Theta(k^n)$, $f(n) = \omega(k^{n-1})$ and $f(n) = o(k^{n+1})$. We will write "$f(n) = \text{poly}(n)$" as a shorthand for "$f(n) = O(k^n)$ for some k".

2.2 Interactive Proof Systems

In this chapter we will dive deep into the nuts and bolds of Zero-knowledge proofs. The road to define ZKPs goes through interactive proof systems. We will note here that interactive proofs are interesting in their own sake but we will not extend more but only to some prerequisites.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

2.2.1 Formal Definition

As we described in the introduction every interactive proof system is related explicitly with two parties and their computational tasks respectively, these are the prover, the one who produces the proof, and the verifier, the one who verifies the validity of the proof. The two parties interaction is based on a *protocol*. A protocol will mean the exchange of messages between two *interactive algorithms* (will be defined below) according to a system of rules which describe the sequence of messages and the syntax of these messages. A move in a protocol is composed by the exchange of a single message from one algorithm to the other. Protocols that are executed within two or more moves are called *interactive* protocols, while those comprising only of one move are called *non-interactive* protocols.

To express mathematically the concepts of prover and verifier, we will use the Turing machine as a model of computation. The interaction between prover and verifier is parametrized by a common input, the statement to be proved (e.g., in the case of QR protocol this statement is: *"v is a quadratic residue mod n"*. Thus, both the Turing machines (prover and verifier) must have a tape from which they read the input. Since the input is common these tapes are shared. After the input, prover and verifier start interacting by means of exchanging messages. The messages sent by one machine need a tape to be written. Upon received by the second machine, this machine will only read them from the same tape. Thus, two more tapes are needed, both acting as write only and read only, both shared and used by alternating the roles of the machines, that is, every machine use one tape as the sender and the other tape as the receiver of messages.

Each machine individually needs a read and write tape for their inner computations. While one machine is busy with inner calculations, the other machine does not halt until receiving a new message, but becomes idle. An additional shared, read and write tape with a single cell, 0 or 1, can express the fact that while the active machine becomes idle, the content of this tape is switched and the other machine, having opposite content, becomes active.

In the cave example and the QR protocol a degree of randomness is involved. In QR the prover chooses randomly an element of $\mathbb{Z}_n^*$ and the verifier an element of $\{0,1\}$. So it is essential both the machines individually have a random tape. This tape contains a possible outcome for a sequence of internal coin tosses. We can picture this sequence as supplied to the machine by an external coin-tossing device, which corresponds to as having a machine containing a random tape with internal coin tosses. Finally, when the machines halt, they both need individually a tape to write their output (e.g., in QR the verifier at the end of the protocol needs to write if he accepts or rejects the validity of the proof) adding to one more write tape.

The above discussion leads naturally to the following definitions of Interactive Turing Machines and Joint Computation (interaction between two such machines):

Definition 2.2.1 (An Interactive Machine)**.**

- *An* ***interactive Turing machine*** *(ITM) is a (deterministic) multi-tape Turing machine. The tapes are a read-only input tape, a read-only random tape, a read-and-write work tape, a write-only output tape, a pair of communication tapes, and a read-and-write switch tape consisting of a single cell. One communication tape is read-only, and the other is write-only.*

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

- *Each ITM is associated a single bit $\sigma \in \{0,1\}$, called its **identity**. An ITM is said to be active, in a configuration, if the content of its switch tape equals the machine's identity. Otherwise the machine is said to be idle. While being idle, the state of the machine, the locations of its heads on the various tapes, and the contents of the writable tapes of the ITM are not modified.*
- *The content of the input tape is called **input**, the content of the random tape is called **random input**, and the content of the output tape at termination is called **output**. The content written on the write-only communication tape during a (time) period in which the machine is active is called the **message sent** at that period. Likewise, the content read from the read-only communication tape during an active period is called the **message received** (at that period).*

Definition 2.2.2 (Joint Computation of Two ITMs)**.**

- *Two interactive machines are said to be **linked** if they have opposite identities, their input tapes coincide, their switch tapes coincide, and the read-only communication tape of one machine coincides with the write-only communication tape of the other machine, and vice versa.*
- *The **joint computation** of a linked pair of ITMs, on a common input x, is a sequence of pairs representing the local configurations of both machines. That is, each pair consists of two strings, each representing the local configuration of one of the machines. In each such pair of local configurations, one machine (not necessarily the same one) is active, while the other machine is idle. The first pair in the sequence consists of initial configurations corresponding to the common input x, with the content of the switch tape set to zero.*
- *If one machine halts while the switch tape still holds its identity, then we say that both machines have halted. The outputs of both machines are determined at that time.*

Notation: We use the letters P and V - free of subscripts and superscripts - to denote honest Provers and Verifiers (i.e., not-cheating/following the protocol instructions). We indicate the private and public inputs as well as the random coin tosses used with subscripts. With an added asterisk, $\mathbf{P}^*$ ($\mathbf{V}^*$) we will mean that the prover (verifier) is potentially dishonest, without excluding the possibility that he is honest.

To indicate that two interactive algorithms executing a protocol, we use angle brackets $\langle A, B \rangle$. We also use the following notations:

- $\text{out}_A \langle A, B \rangle$ - the output of A after this interaction is finished. Similarly we define $\text{out}_B \langle A, B \rangle$.
- $\text{view}_A \langle A, B \rangle$ - the view of A during the interaction: all the messages it received.

For example, in QR $\text{out}_V \langle P, V_{v,\beta} \rangle$ gives the output of the Verifier after interacting with the Prover with public input v and private input β.

Because this notation is somewhat cumbersome we will use, until specified differently, a lighter definition by dropping the prefix out_a and use $\langle A, B \rangle(x)$ to denote the local output of B on common input x when interacting with machine A. Hence, he random variable $\langle A, B \rangle(x)$ accounts for B's output.

As we see Turing machines are a basic tool needed in our formulation of interactive

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

proofs and ZKPs. It is natural to consider the time complexity of such interactive machines as a function of their input length. The following definition is needed:

Definition 2.2.3 (The Complexity of an Interactive Machine)**.** *We say that an interactive machine A has **time-complexity** $t : \mathbb{N} \to \mathbb{N}$ if for every interactive machine B and every string x, it holds that when interacting with machine B, on common input x, machine A always (i. e., regardless of the content of its random tape and B′s random tape) halts within $t(|x|)$ steps. In particular, we say that A is **polynomial-time** if there exists a polynomial p such that A has time-complexity p.*

We emphasize that $t(\cdot)$ is an upper bound for all the messages arriving at the interactive machine. That is, the machine's complexity does not depend on the content of the messages it receives.

We are now in position to give formally the definition of an interactive proof system. In the subsequent definitions, examples and protocols, the verifier's output is its decision on whether to accept or reject the proof on the common input. By convention we use as output 1 to represent "accept" and 0 for "reject".

Definition 2.2.4 (Interactive Proof System)**.** *Let P and V be interactive Turing machines with P (called the Prover) computationally unbounded and V (called the Verifier) having probabilistic polynomial-time complexity. Let L be a language, we call the pair (P, V) an **interactive proof system for language** L if the following two properties are satisfied:*

- ***Completeness**: For every $x \in L$,*

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

- ***Soundness**: For every $x \notin L$ and every interactive machine B,*

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

Comments: Soundness statement applies to all provers, honest and dishonest. This guarantees that the verifier is sound against all possible provers that may cheat or not. On the contrary, for completeness we require the honesty of both the Prover and the Verifier. Note that the Prover is not strictly computationally bounded as the Verifier. As we have mentioned, this is because the weight of the proof lies on the verification process.

In practice, an error probability of $1/3$ might not be acceptable, the choice $1/3$ in the definition is arbitrary, remember Victor in the cave example can decide how many times to repeat the experiment to reduce the error probability. We can modify the definition as in the case of $\mathcal{BPP}$ so the error probability can be made exponentially small between 0 and $1/2$ in place of $1/3$, by repeating the interaction (polynomially) many times.

In general, the completeness and soundness bounds do not even have to be constant. We can replace them with two functions $c, s : \mathbb{N} \to [0, 1]$ satisfying $c(n) < 1 - 2^{-\text{poly}(n)}$, $s(n) < 2^{-\text{poly}(n)}$ and $c(n) > s(n) + \frac{1}{\text{poly(n)}}$.

Namely, we consider the following generalization of Interactive Proof Systems:

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Definition 2.2.5 (Generalized Interactive Proof)**.** *Let $c, s : \mathbb{N} \to \mathbb{R}$ be functions satisfying $c(n) > s(n) + \frac{1}{p(n)}$ for some polynomial $p(\cdot)$. An interactive pair (P, V), as defined in 2.2.4, is called a (generalized) interactive proof system for the language L, with **completeness bound** $c(\cdot)$ and **soundness bound** $s(\cdot)$, if:*

- ***Completeness**: For every $x \in L$,*

$$\Pr[\langle P, V\rangle(x) = 1] \geq c(|x|)$$

- ***Soundness**: For every $x \notin L$ and every interactive machine B,*

$$\Pr[\langle B, V\rangle(x) = 1] \leq s(|x|)$$

2.2.2 Quadratic Residue Revisited

We will now treat the QR protocol formally and prove it is an Interactive Proof System. First, we find the language L in which our statement belongs. Since our statement is *"v is a quadratic residue modulo n"* the language we are referring to is:

$$L = QR(\mathbb{Z}_n^*) = \{x \in \mathbb{Z}_n^* : (\exists s \in \mathbb{Z}_n^*)(s^2 = x \bmod n)\}$$

the set of all quadratic residues modulo n. So our statement more formally becomes $v \in QR(\mathbb{Z}_n^*)$. We note that $QR(\mathbb{Z}_n^*)$ is a group with respect to multiplication of quadratic residues modulo n.

Completeness: This means if we have an honest prover, his proof we always be accepted by the Verifier, that is, Verifier checks if $\gamma^2 = \alpha v^\beta$ is equivalent with $\gamma^2 = r^2 s^{2\beta}$. Since P is honest, he sends $\gamma = rs^\beta$, so $\gamma^2 = r^2 s^{2\beta} = r^2 (s^2)^\beta = \alpha v^\beta \Rightarrow \gamma^2 = \alpha v^\beta$.

Soundness: This means if v is not a quadratic residue, then regardless of what the Prover does, the Verifier will reject the proof, specifically for our protocol, he will reject with probability at least 1/2.

We assume a dishonest prover P^*, we will prove $\Pr[\langle P^*, V\rangle(x) = 1] \leq \frac{1}{2}$. We distinguish two cases, $\alpha \in QR(\mathbb{Z}_n^*)$ and $\alpha \notin QR(\mathbb{Z}_n^*)$. First we prove the following lemma:

Lemma 2.2.1. Product of a quadratic residue and a quadratic non-residue is a quadratic non-residue.

Proof. Let $x \in QR(\mathbb{Z}_n^*)$ and $y \notin QR(\mathbb{Z}_n^*)$ and assume in contrary that $z = xy \in QR(\mathbb{Z}_n^*)$. Since $QR(\mathbb{Z}_n^*)$ is a group we have: $xy = z \pmod n \Rightarrow y = zx^{-1} \pmod n$. Since z, x^{-1} are group elements, the right side product is quadratic residue but y is a quadratic non-residue, this is a contradiction, thus z is not a quadratic residue. □

Proof of soundness.
Case 1. $\alpha \in QR(\mathbb{Z}_n^*)$ We have the next two possible cases with same probabilities of 1/2 (since there are only two options for β with the same probability):

- Let assume the verifier sends $\beta = 1$. In this case the proof will fail since $\gamma^2 = \alpha v^1 = \alpha v$ and $\gamma^2 \in QR(\mathbb{Z}_n^*)$ but from the above lemma $\alpha v \notin QR(\mathbb{Z}_n^*)$.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

- Let assume the verifier sends $\beta = 0$. Then we have $\gamma^2 = \alpha v^0 = \alpha$. Obviously $\gamma^2 \in QR(\mathbb{Z}_n^*)$ and $\alpha \in QR(\mathbb{Z}_n^*)$. The prover can just send the r he posses for the γ and the verifier will find $\gamma^2 = r^2 = \alpha$ which holds.

Thus the verifier will reject with probability $\geq 1/2$.

Case 2. $\alpha \notin QR(\mathbb{Z}_n^*)$ We have the next two possible cases with same probabilities of $1/2$:

- If the verifier sends $\beta = 1$ the proof might succeed. We have $\gamma^2 = \alpha v^1 = \alpha v$, the left most side is in $QR(\mathbb{Z}_n^*)$ but the right most side it is not. The prover can choose γ and precompute γ^2 before sending α and then send γ^2/v for α. So the verifier might get convinced as on the both sides of equality $\gamma^2 = \alpha v$ are quadratic residues.
- If the verifier sends $\beta = 0$ the proof fails. $\gamma^2 = \alpha v^0$, but $\gamma^2 \in QR(\mathbb{Z}_n^*)$ and $\alpha \notin QR(\mathbb{Z}_n^*)$.

□

2.3 Zero-knowledge Definitions

We saw that soundness is a property of the prescribed verifier to protect himself against any malicious prover. Zero-knowledge property in contrast, favors the prescribed prover in case of a dishonest verifier, it captures prover's robustness against attempts to gain knowledge by interacting with him.

In this section we will introduce the most important aspect of ZKPs, the *zero-knowledge property*. We will go gradually from the simplest definition (perfect zero-knowledge) to more advanced definitions.

2.3.1 Perfect Zero-knowledge

Let us go back to our informal definition 1.2.1 of zero-knowledge. After our conversation in 1.2.4 we will choose the more precise but still informal definition:

Definition 2.3.1. *An interactive proof system (P, V) for a language L is zero-knowledge if whatever can be efficiently computed after interacting with P on input $x \in L$, can also be efficiently computed from x (without any interaction).*

We use the remarks we did in the cave example to motivate us for a rigorous approach to a formal definition. Recall that there Peggy made two tapes, the original one and a staged recording that no one could distinguish one from the other. Even if Peggy would show the original tape to a third person, she could be easily accused of faking the tape and could do nothing to convince anyone otherwise.

We call the staged tape a simulator. Imagine now an interaction of a prover P and a verifier V, and a simulator of their interaction. Note that when P interacting with V, he makes explicitly use of knowledge that he only posses. In the cave example is the secret word itself, in QR is the secret key s, while the simulator does not make use of this knowledge but is only given the public input. The simulator will create a transcript that *"looks"* like an interaction of P and V. Lets consider also a transcript produced by the real interaction of prover and verifier. This transcript must leak as much information as the simulated one, otherwise both would be distinguishable and we would not talk about simulation. Since the simulated protocol

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

transcript does not make use of any knowledge, like the key s, and only make use of the public input, it leaks no information at all about private knowledge (according to our informal definition so far). This means that the simulated transcript is in zero-knowledge. The conclusion is that the real protocol transcript, being indistinguishable from the simulated one or in other words being as good as the real one, also leaks no information as well.

The concept of simulation helps us define the zero-knowledge property in a practical way since our informal definition is somewhat abstract. All in all we have to do to assure that an interactive protocol is in zero-knowledge is to prove the existence of a simulator which generates fake transcripts, without access to the prover's secret knowledge, that are *"indistinguishable"* from a genuine proof.

Our new formal definition follows:

Definition 2.3.2. [Gol01b] *Let P be an interactive Turing machine and L a language. We say that P is **perfect zero-knowledge** if, for every probabilistic polynomial-time interactive verifier V^*, there exists a probabilistic polynomial-time algorithm M^* such that the following two random variables are identically distributed* [1]*:*

- $\{\langle P, V^* \rangle(x)\}_{x \in L}$ *(i.e., the output of the interactive machine V^* after interacting with P on common input x)*
- $\{M^*(x)\}_{x \in L}$ *(i.e., the output of machine M^* on input x)*

The machine M^* is called a *simulator* for the interaction of V^* and P. The first random variable represents an actual execution of the protocol with the pair (P, V^*). The second variable is a separate procedure, it does not interact with any algorithm.

We mention that the existence of a simulator is not obligatory to achieve zero knowledge. We could have a zero-knowledge interaction without the use of simulator. Conceptually, it is difficult to support the notion of *"no real gain"* with a well-formed definition, simulation is an indirect approach which nonetheless serves the purpose of zero-knowledge.

So far, the above definition is too tight for non-trivial cases. We can loosen up the definition and allow the simulator to fail with bounded probability and produce an interaction. This is the same as the staged tape Peggy made. There, almost half of the tries were unsuccessful since the actor came out from the wrong path, so Peggy had to edit out these tries.

Definition 2.3.3 (Perfect Zero-knowledge)**.** [Gol01b] *Let P be an interactive Turing machine and L a language. We say that P is **perfect zero-knowledge** if, for every probabilistic polynomial-time interactive verifier V^*, there exists a probabilistic polynomial-time algorithm M^* such that the following two conditions hold:*

1. *With probability at most $\frac{1}{2}$, on input x, machine M^* outputs a special symbol denoted $\perp$ (i.e., $\Pr[M^*(x) = \perp] \leq \frac{1}{2}$).*
2. *Let $m^*(x)$ be a random variable describing the distribution of $M^*(x)$ conditioned on $M^*(x) \neq \perp$ (i.e., $\Pr[m^*(x) = a] = \Pr[M^*(x) = a | M^*(x) \neq \perp]$ for every $a \in \{0,1\}^*$). Then the following random variables are identically distributed:*

[1] Identically distributed meaning the cumulative distribution functions of $\langle P, V^* \rangle(x)$ and $M(x)$ are equal.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

- $\langle P, V^* \rangle(x)$ *(i.e., the output of the interactive machine V^* after interacting with P on common input x)*
- $m^*(x)$ *(i.e., the output of machine M^* on input x, conditioned on not being $\perp$)*

We could modify the first condition by requiring the machine M^* output $\perp$ with probability at most a polynomial-time computable function $b(\cdot)$. Then, the statistical difference between the interactive pair (P, V^*) and the simulator M^* becomes negligible. In this case, there is an extremely small error between whatever a verifier can efficiently compute (after interacting with the prover) and the simulator (for the same computation).

The notion of *"computational indistinguishability"* will be our basis for the next definition of zero-knowledge.

2.3.2 Computational Zero-knowledge

In computational complexity, we consider objects to be computationally indistinguishable if the differences between the objects cannot be observed by any efficient procedure. The objects themselves are considered as infinite sequences of strings. Hence, if for two sequences $\{x_n\}_{n \in N}$ and $\{y_n\}_{n \in N}$ there is no efficient algorithm D that can accept infinitely many x_n's while rejecting their y counterparts, then are said to be computationally indistinguishable.

This view of indistinguishability is not strict to sequences of strings but can extend naturally to the probabilistic settings. We can talk about computationally indistinguishable distributions when there is no efficient algorithm D that can tell apart the probability to accept strings chosen from these distributions. These strings are infinite sequences of distributions called *probability ensembles*, rather than fixed distributions.

Definition 2.3.4 (Probability Ensemble)**.** [Gol01a] *Let I be a countable index set. An **ensemble indexed** by I is a sequence of random variables indexed by I. Namely, any $X = \{X_i\}_{i \in I}$, where each X_i is a random variable, is an ensemble indexed by I.*

The ensembles can be indexed also by strings from a language L.

Definition 2.3.5 (Computational Indistinguishability)**.** [Gol01a] *Let $X \stackrel{def}{=} \{X_n\}_{n \in \mathbb{N}}$ and $Y \stackrel{def}{=} \{Y_n\}_{n \in \mathbb{N}}$ be two probability ensembles. We say $\{X_n\}_{n \in \mathbb{N}}$ and $\{Y_n\}_{n \in \mathbb{N}}$ are **computationally indistinguishable** (or **indistinguishable in polynomial time**) if for every probabilistic polynomial-time algorithm D, every positive polynomial $p(\cdot)$, and all sufficient large n's,*

$$|\Pr_{t \leftarrow X_n}[D(t) = 1] - \Pr_{t \leftarrow Y_n}[D(t) = 1]| < \frac{1}{p(n)}$$

$\Pr_{t \leftarrow X_n}[D(t) = 1]$ (similar notation as $\Pr[\mathrm{D}(\mathrm{X_t}) = 1]$) denotes the probability the algorithm D outputs 1 on input X_t.

The discussion on indistinguishability motivates us to a new definition of zero-knowledge. In practise if very difficult to construct a "perfect" simulation for the output of V^*. Actually, it suffices to generate a probability distribution that is computationally indistinguishable from the output of V^* after it interacts with P.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Definition 2.3.6 (Computational Zero-knowledge)**.** [Gol01b] *Let P be an interactive Turing machine and L a language. We say that P is **computational zero-knowledge** (or just **zero-knowledge**) if, for every probabilistic polynomial-time interactive verifier V^*, there exists a probabilistic polynomial-time algorithm M^* such that the following two ensembles are computationally indistinguishable:*

- $\{\langle P, V^* \rangle(x)\}_{x \in L}$ *(i.e., the output of the interactive machine V^* after interacting with P on common input x)*
- $\{M^*(x)\}_{x \in L}$ *(i.e., the output of machine M^* on input x)*

Think of the algorithm D as a procedure of distinguishing a distribution based on a sample and D's verdict of accepting the sample drawn from this distribution as 1. When the relation of computational indistinguishability is satisfied, it simply means that D accepts the sample coming from the first distribution as much as coming from the second distribution. In other words, D cannot efficiently determine the origin distribution of the sample.

As in perfect zero-knowledge, we can adopt an alternative definition considering the conditional output distribution of the simulator as in Definition 2.3.3 which does not increase the power of computational zero-knowledge.

There is an alternative option of simulation which appeals more naturally in our working of zero-knowledge and takes into account the verifier's view, that is, the complete sequence of its local configurations during the interaction with the prover. This is equal to the content of the random tape and the messages he received during the protocol's execution since its inner configuration depends on these data.

Definition 2.3.7 (Alternative formulation of Zero-knowledge)**.** [Gol01b] *Let (P, V) be an interactive proof system for some language L. We denote by $\text{view}_{V^*}^{P}(x)$ a random variable describing the content of the random tape of V^* and the messages V^* receives from P during a joint computation on common input x. We say that (P, V) is **zero-knowledge** if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* such that the ensembles $\{\text{view}_{V^*}^{P}(x)\}_{x \in L}$ and $\{M^*(x)\}_{x \in L}$ are computationally indistinguishable.*

Although the alternative definition is equivalent with definition 2.3.6, the former is more convenient to work with. Note that we can also obtain a perfect zero-knowledge definition by considering the view of the verifier instead of his output.

2.3.3 Statistical Zero-knowledge

A relaxed version of perfect zero-knowledge is the following:

Definition 2.3.8 (Almost-Perfect (Statistical) Zero-knowledge)**.** [Gol01b] *Let P be an interactive Turing machine and L a language. We say that P is **almost-perfect zero-knowledge** (or **statistical zero-knowledge**) if, for every probabilistic polynomial-time interactive verifier V^*, there exists a probabilistic polynomial-time algorithm M^* such that the following two ensembles are statistically close as functions of $|x|$:*

- $\{\langle P, V^* \rangle(x)\}_{x \in L}$ *(i.e., the output of the interactive machine V^* after interacting with P on common input x)*
- $\{M^*(x)\}_{x \in L}$ *(i.e., the output of machine M^* on input x)*

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Two statistical ensembles $X \stackrel{def}{=} \{X_n\}_{n\in\mathbb{N}}$ and $Y \stackrel{def}{=} \{Y_n\}_{n\in\mathbb{N}}$ are statistically close if their statistical difference is negligible[2] where the statistical difference between two distributions X and Y over a finite domain D is defined as the function:

$$\Delta(X,Y) = \frac{1}{2}\sum_{a\in D} |\Pr[X=a] - \Pr[Y=a]|$$

That is, the statistical difference between $\langle P, V^*\rangle(x)$ and $M^*(x)$ is negligible in terms of $|x|$.

There is a weak notion of zero-knowledge, called honest-verifier zero-knowledge, which as the name suggests requires simulatability of only the prescribed (or honest) verifier's view and not any possible verifier. This notion of zero-knowledge is interesting for the following special reason: public coin protocols that are zero-knowledge with respect to the honest verifier imply similar protocols that are zero-knowledge in general [GSV98].

Definition 2.3.9 (Zero-knowledge with respect to an honest verifier)**.** [Gol01b] *Let (P,V), L and $\text{view}_V^P(x)$ be as in definition 2.3.7. We say that (P,V) is **honest-verifier zero-knowledge** if there exists a probabilistic polynomial time algorithm M such that the ensembles $\{\text{view}_V^P(x)\}_{x\in L}$ and $\{M(x)\}_{x\in L}$ are computationally indistinguishable.*

Quadratic Residue protocol satisfies Statistical Zero-knowledge

We note here that QR in general is consider to be hard to compute. In [MA78] is proved that this problem is NP-complete for solutions within a limit parameter. In [Rab79] Rabin showed that finding square roots modulo n is equivalent to factoring n.

Let V^* be a verifier. The simulator S will do the following:

1. **Input**: v, n such that $v \in QR(\mathbb{Z}_n^*)$. Note that the simulator does not get s such that $v = s^2 \pmod{n}$.
2. Choose $\beta' \leftarrow \{0,1\}$ (guess the challenge of the verifier).
3. Choose $\gamma \leftarrow \mathbb{Z}_n^*$.
4. Compute $\alpha = \frac{\gamma^2}{v^{\beta'}}$
5. Invoke V^* on the message α to obtain a bit β.
6. If $\beta \neq \beta'$ HALT.
7. Output γ.

We will prove that the described simulator S with probability $1/2$ simulates the QR protocol.

We call *transcript* a set of messages exchanged during the conversation between parties and denote it with $tr = (\alpha, \beta, \gamma)$ where α, β, γ are the messages exchanged. Note that creating a simulator, the order of how messages are exchanged is not important.

Lemma 2.3.1. The transcripts (α, β, γ) as outputs from the conversations A_1 and A_2 defined as below have the same probability distributions.

[2] a function $\mu : \mathbb{N} \to \mathbb{R}$ is called negligible if for every positive polynomial $p(\cdot)$ there exists an N such that for all $n > N$, $\mu(n) < \frac{1}{p(n)}$

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

A_1	A_2
$r \leftarrow \mathbb{Z}_n^*$	$\beta \leftarrow \{0,1\}$
$\alpha \leftarrow r^2$	$\gamma \leftarrow \mathbb{Z}_n^*$
$\beta \leftarrow \{0,1\}$	$\alpha = \frac{\gamma^2}{v^\beta}$
$\gamma \leftarrow rs^\beta$	**return** (α,β,γ)
return (α,β,γ)	

Proof. We will prove the equality of the following probabilities:

$$\Pr[r \leftarrow \mathbb{Z}_n^*, \alpha \leftarrow r^2, \beta \leftarrow \{0,1\}, \gamma \leftarrow rs^\beta : \alpha, \beta, \gamma] \tag{2.1}$$

$$\Pr[\beta \leftarrow \{0,1\}, \gamma \leftarrow \mathbb{Z}_n^*, \alpha = \frac{\gamma^2}{v^\beta}, : \alpha, \beta, \gamma] \tag{2.2}$$

Since β ranges over $\{0,1\}$ we can use conditional probabilities. Thus the left side of (2.1) becomes:

$$\begin{aligned}
\Pr[r \leftarrow \mathbb{Z}_n^*, \alpha \leftarrow r^2, \beta \leftarrow \{0,1\}, \gamma \leftarrow rs^\beta : \alpha, \beta, \gamma] = \\
\Pr[\beta \leftarrow \{0,1\} : \beta = 0] \cdot \Pr[r \leftarrow \mathbb{Z}_n^*, \alpha \leftarrow r^2, \gamma \leftarrow r : \alpha, \gamma] + \\
\Pr[\beta \leftarrow \{0,1\} : \beta = 1] \cdot \Pr[r \leftarrow \mathbb{Z}_n^*, \alpha \leftarrow r^2, \gamma \leftarrow rs : \alpha, \gamma] = \\
\frac{1}{2} \cdot \Pr[r \leftarrow \mathbb{Z}_n^* : r = \gamma] \cdot \Pr[\alpha \leftarrow \mathbb{Z}_n^* : \alpha = \gamma^2] + \\
\frac{1}{2} \cdot \Pr[r \leftarrow \mathbb{Z}_n^* : r = \gamma s^{-1}] \cdot \Pr[\alpha \leftarrow \mathbb{Z}_n^* : \alpha = \gamma^2 s^{-2}]
\end{aligned}$$

Since s is invertible and r, γ are uniquely determined by α, the last sum of probabilities equals:

$$\frac{1}{2}(\frac{1}{\phi(n)} + \frac{1}{\phi(n)}) = \frac{1}{\phi(n)}$$

whenever $\alpha v^\beta = \gamma^2$ and 0 otherwise ($\phi(n)$ is the number of elements in $\mathbb{Z}_n^*$).
The left side of (2.2) becomes:

$$\begin{aligned}
\Pr[\beta \leftarrow \{0,1\}, \gamma \leftarrow \mathbb{Z}_n^*, \alpha = \frac{\gamma^2}{v^\beta}, : \alpha, \beta, \gamma] = \\
\Pr[\beta \leftarrow \{0,1\} : \beta = 0] \cdot \Pr[\gamma \leftarrow \mathbb{Z}_n^*, \alpha = \gamma^2 : \alpha, \gamma] + \\
\Pr[\beta \leftarrow \{0,1\} : \beta = 1] \cdot \Pr[\gamma \leftarrow \mathbb{Z}_n^*, \alpha = \frac{\gamma^2}{v} : \alpha, \gamma] = \\
\frac{1}{2} \cdot \Pr[\gamma \leftarrow \mathbb{Z}_n^*, \alpha = \gamma^2 : \alpha, \gamma] + \frac{1}{2} \cdot \Pr[\gamma \leftarrow \mathbb{Z}_n^*, \alpha = \frac{\gamma^2}{v} : \alpha, \gamma]
\end{aligned}$$

Since v is invertible and γ is uniquely determined by α, the last sum of probabilities equals:

$$\frac{1}{2}(\frac{1}{\phi(n)} + \frac{1}{\phi(n)}) = \frac{1}{\phi(n)}$$

whenever $\alpha v^\beta = \gamma^2$ and 0 otherwise.

Thus probabilities (2.1) and (2.2) are equal which shows that transcripts (α, β, γ) as outputs from A_1 and A_2 have the same probability distributions. □

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Lemma 2.3.2. The QR protocol satisfies statistical zero-knowledge.

Proof. The communication between prover and verifier looks like below:

$$\begin{array}{l} \underline{V^P(\phi)} \\ r \leftarrow \mathbb{Z}_n^* \\ \alpha \leftarrow r^2 \\ \beta \leftarrow \{0,1\} \\ \gamma \leftarrow rs^{\beta} \\ \textbf{return } V(\phi) \end{array}$$

We wonder how much different is the simulator's transcript from V^P.

$$\begin{array}{l} \underline{S^V} \\ \beta' \leftarrow \{0,1\} \\ \gamma \leftarrow \mathbb{Z}_n^* \\ \alpha \leftarrow \dfrac{\gamma^2}{v^{\beta'}} \\ \beta \leftarrow V(\alpha) \\ if \beta \neq \beta' then HALT \\ \textbf{return } V(\gamma) \end{array}$$

Using lemma 2.3.1 we can replace part of S^V which is equal to A_2 with that of A_1 getting:

$$\begin{array}{l} \underline{R^V} \\ r \leftarrow \mathbb{Z}_n^* \\ \alpha \leftarrow r^2 \\ \beta' \leftarrow \{0,1\} \\ \gamma \leftarrow rs^{\beta'} \\ \beta \leftarrow V(\alpha) \\ if \beta \neq \beta' then HALT \\ \textbf{return } V(\gamma) \end{array}$$

In R^V we notice that β depends only on α, so we can move the assign $\beta \leftarrow V(\alpha)$ up after the occurrence of α transforming R^V like this:

$$\begin{array}{l} \underline{R^V} \\ r \leftarrow \mathbb{Z}_n^* \\ \alpha \leftarrow r^2 \\ \beta \leftarrow V(\alpha) \\ \beta' \leftarrow \{0,1\} \\ \gamma \leftarrow rs^{\beta'} \\ if \beta \neq \beta' then HALT \\ \textbf{return } V(\gamma) \end{array}$$

Let us analyze the behaviour of R^V. First note that $\Pr[R^V = HALT] = \frac{1}{2}$, since it equals the probability β' will not much β. This means that if we run the procedure for k steps, we will halt with very high $(1 - 2^{-k})$ probability.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

If we denote the output of V^P as ψ we have:
$\Pr[R^V = \psi | R^V \neq HALT] = \Pr[V(\gamma) = \psi | \beta = \beta'] = \Pr[V^P = \psi]$.
This shows statistical probability distribution equality of S^V and R^V which means that the described simulator S with probability $1/2$ simulates the QR protocol. $\square$

The simulator algorithm has small probability of running for a long time. To make into an algorithm that at the worst case makes at most $|x|$ invocations of V^*, we can just stop and output an arbitrary value after more than $|x|$ iterations. We'll introduce at most $2^{|x|}$ statistical distance this way.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Chapter 3

Σ-Protocols

In this Chapter we will present an important class of zero-knowledge protocols called Sigma Protocols. Sigma Protocols (or Σ-Protocols) are interesting in their own sake. Lot of cryptographic schemes are actually Sigma protocols e.g., knowledge of discrete logarithm, equality of discrete logarithm, knowledge of message and randomness in a Pedersen commitment. Another virtue is that we can build upon Sigma protocols new such protocols and so on. Additionally, they give us the chance to present an important notion of interactive proofs, this of "Proof of Knowledge".

3.1 The Discrete Logarithm problem in Cryptography

In its most standard form, the *discrete logarithm problem* (DLP) in a finite Group G can be stated as follows[1]:

Given $\alpha \in G$ and $\beta \in \langle \alpha \rangle$, where $\langle \alpha \rangle$ is the subgroup generated by α, find the least positive integer x such that $\beta = \alpha^x$.

We call x the *discrete logarithm* of β with respect to the base α and denote it as $\log_\alpha \beta$. A slightly stronger version of the problem formulates as follows:

Given $\alpha, \beta \in G$, compute $\log_\alpha \beta$ if $\beta \in \langle \alpha \rangle$ and otherwise report that $\beta \notin \langle \alpha \rangle$.

This could be orders more difficult problem and that is because if we are lucky, we could find an algorithm that outputs efficiently $\log_\alpha \beta$ when the case is that β lies in $\langle \alpha \rangle$, otherwise, such an algorithm could be hard to find or even it could be hard tell if β does not lie in $\langle \alpha \rangle$. On the other hand, with a deterministic algorithm, we can unequivocally determine whether β lies in $\langle \alpha \rangle$ or not.

There is a generalisation called the *extended discrete problem*:

Given $\alpha, \beta \in G$, determine the least positive integer y such that $\beta^y \in \langle \alpha \rangle$, and then output the pair (x, y) where $x = \log_\alpha \beta^y$.

This yields positive integers x and y satisfying $\beta^y = \alpha^x$, where we minimize y first and x second. Note that there is always a solution: in the worst case $x = |\alpha|$ and $y = |\beta|$, the orders of α and β.

Finally, we can consider a vector form of the discrete logarithm problem:

Given $\alpha_1, \ldots, \alpha_r \in G$ and $n_1, \ldots, n_r \in \mathbb{Z}$ such that every $\beta \in G$ can be written uniquely as $\beta = \alpha_1^{e_1} \cdots \alpha_r^{e_r}$ with $e_i \in [1, n_i]$, compute the exponent vector $(e_1, \ldots, e_r)$ associated to a given β.

[1] https://math.mit.edu/classes/18.783/2021/LectureNotes9.pdf

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Observe that in all the statements of the DLP we use multiplicative terminology and notation. This stems from the fact that most of the early work on discrete logarithms focused on multiplicative groups of a finite field. In additive notation we have - for the initial statement of DLP - $x\alpha = \beta$. Look below in example 2 for an additive case.

Examples:

1. Suppose $G = \mathbb{Z}_{101}^{*}$, then $\log_3 37 = 24$ since $3^{24} \equiv \bmod 101$.
2. Suppose $G = \mathbb{Z}_{101}^{+}$, then $\log_3 37 = 46$ since $46 \cdot 3 \equiv \bmod 101$.

In the above examples we have respectively group orders of $100 = 2^2 \cdot 5^5$ and 101, which is prime. In both cases it is easy to compute the discrete logarithm but not due to the order of the groups but for different reasons. In the multiplicative case, although 100 is composite, is smooth (i.e. product of small primes), in this case it is easy to compute discrete logarithms. While 101 is prime which in terms of groups structure represents the hardest case, it turns out to be very easy to compute discrete logarithms in the additive group of a finite field. We observe that $x \cdot 3 = 37 \Rightarrow x = 37 \cdot 3^{-1}$, thus, all we need to do is to compute the multiplicative inverse of 3 mod 101, which is 34, and multiply by 37.

In the additive case, even if the field size is very large, we can "cheat" in a sense and use the extended Euclidean algorithm to compute multiplicative inverses in fast times. The cheating has to do with the fact that in a Euclidian domain we are allowed to use multiplication beside the standard group addition. Thus, the hardness of DLP does not really depends on the group's order, but on the group's representation. $\mathbb{Z}_{101}^{+}$ as every $\mathbb{Z}/p\mathbb{Z}$ group of prime order, is a Euclidian domain and every prime order p group is isomorphic to it. Computing the discrete logarithm amounts to computing this isomorphism.

In the multiplicative case DLP is believed to be much harder. There are sub-exponential time algorithms, whereas in the generic setting only exponential time algorithms exist. An extended survey on discrete logarithms and their computation can be found here [JOP14].

3.2 DLP for ZKPs

This hardness hypothesis of the discrete logarithm problem is the foundation for the security of a large variety of public key systems and protocols. The most famous of them being the Diffie-Hellman key exchange [DH76] which have been published more than four decades ago. Since then, there have been substantial algorithmic advances in the computation of discrete logarithms. However, the DLP in general is still considered to be hard both for multiplicative groups of finite fields and the additive group of a general elliptic curve.

The discrete logarithm problem is not used only for the Diffie-Hellman key exchange, but since the invention of the RSA cryptosystems it is used for encryption and signatures. No wonder the DLP made it in to the zero-knowledge proof protocols. Schnorr in [Sch90] gave an identification protocol based on an alternative ZKP protocol called zero-knowledge proof of knowledge of a discrete logarithm.

Let p be a prime and q a prime divisor of $p-1$. Let g be an element of order q in $\mathbb{Z}_p^*$. From the fundamental theorem of cyclic groups there is only one subgroup

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

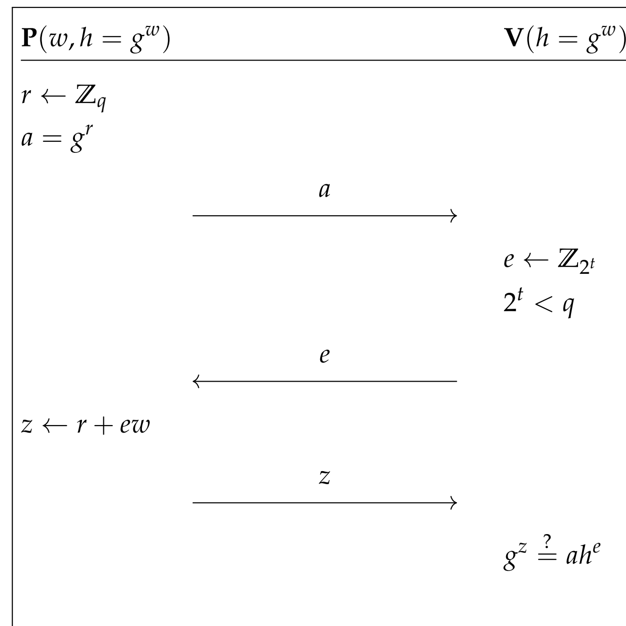
of order q in $\mathbb{Z}_p^*$ and this is $\langle g \rangle$. Suppose a prover chooses $w \in \mathbb{Z}_q$ and publishes $h = g^w \bmod p$ - automatically this means that $h \in \langle g \rangle$. A verifier who gets p, q, g, h can check that p, q are primes and that g, h have order q. Since $h \in \langle g \rangle$ the verifier can tell that there exists w such that $h = g^w$ but he is not sure that the prover knows such a w.

Schnorr suggested the following interaction as an efficient way to convince a verifier:

1. P chooses r at random in $\mathbb{Z}_q$ and sends $a = g^r \bmod p$ to V.

2. V chooses a challenge e at random in $\mathbb{Z}_{2^t}$, where t is fixed such that $2^t < q$, and sends it to P.

3. P sends $z = r + ew \bmod q$ to V who accepts if and only if $g^z = ah^e \mod p$.



Note the three movement form. Later we will prove that Schnorr's protocol is a zero-knowledge proof protocol. Additionally to this, the protocol belongs to a special kind of protocols called Σ-protocols which as we will see have their own interest. Σ-protocols primitive where first introduced as an abstract notion by Cramer [CRA96]. The following definitions capture the essential properties of these protocols.

3.2.1 Definitions

Let $\mathcal{R}$ be a binary relation in $\{0,1\}^* \times \{0,1\}^*$ such that if $(x, w) \in \mathcal{R}$, then the length of w is at most $p(|x|)$, for some polynomial $p(\cdot)$ (we say that $\mathcal{R}$ is polynomially bounded). We can think of x as an instance of some computational problem and w as a witness; a solution to that instance. For the DLP this relation is $\mathcal{R} = \{(h, w) \in \mathbb{Z}_q \times \mathbb{Z}_q : g^w = h\}$. So, $\mathcal{R}$ contains all the discrete logarithm problems in $\mathbb{Z}_q$ and their solutions.

Let $\mathcal{R}(x) = \{w : (x, w) \in \mathcal{R}\}$ and $L_{\mathcal{R}} = \{x : \exists w \text{ s.t. } (x, w) \in \mathcal{R}\}$ (the domain of the binary relation). We say that $\mathcal{R}$ is an $\mathcal{NP}$-relation if it is polynomially bounded and, in addition, there exists a polynomial-time algorithm for deciding membership in $\mathcal{R}$, indeed, it follows that $L_{\mathcal{R}} \in \mathcal{NP}$.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

The form of the Schnorr protocol can be abstractly stated as follows: P, V are probabilistic polynomial time machines, (x, w) belongs to a relation $\mathcal{R}$ where x is common input to P and V, w is private input to P and the interaction between them is described as:

1. P sends a message a.
2. V sends a random t-bit string e.
3. P generates and sends a response z, and V decides to accept or reject based on the data he has seen, namely x, a, e, z.

We define $V(x, a, e, z) = \texttt{true}$ if and only if the verifier accepts, otherwise he rejects.

Definition 3.2.1. *A protocol $\mathcal{P}$ is said to be a Σ-protocol for relation $\mathcal{R}$ if:*

- *$\mathcal{P}$ is of the above 3-move form. The Greek letter Σ visualizes the flow of the protocol.*
- ***Perfect Completeness**: If P, V follow the protocol on input x and private input w to P where $(x, w) \in \mathcal{R}$, the verifier always accepts.*
- ***Special Soundness**: From any x and any pair of accepting conversations on input $x, (a, e, z,), (a, e', z')$ where $e \neq e'$, one can efficiently compute w such that $(x, w) \in \mathcal{R}$.*
- *$\mathcal{P}$ is **Honest-Verifier Zero-Knowledge (HVZK)***

Some results from the definition are the following:

Lemma 3.2.1. *The properties of Σ-protocols are invariant under parallel composition, for instance repeating a Σ-protocol for $\mathcal{R}$ twice in parallel produces a new Σ-protocol for $\mathcal{R}$ with challenge length $2t$.*

Look at Section 4.1 for parallel composition.

Lemma 3.2.2. *If a Σ-protocol for $\mathcal{R}$ exists, then for any t, there exists a Σ-protocol for $\mathcal{R}$ with challenge length t.*

3.2.2 Proofs of knowledge

We will introduce now a notion of proof systems known as *Proofs of Knowledge* (PoK). Although there are many approaches on defining PoKs, most of them often mislead new researchers. In this thesis we will start with a naive approach and interpretation along with formal definition(s).

In the definition of interactive proofs 2.2.1 the prover P is computationally unbounded. Suppose now P wants to to prove a statement $x \in L$ for a language L in $\mathcal{NP}$. Since P is unbounded he can just do this, to establish easily by computation that x is in L and send the appropriate messages to the verifier. This is merely an assertion for existence of a witness (since the fact that "$x \in L$" provides the existence of a witness w for this x). Nonetheless, is often important a prover to demonstrate not only that $x \in L$ but also that "knows" a witness w for this x, which is a much stronger requirement. We need a notion which makes the distinction between a proof of language membership and a proof of knowledge.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Proof of knowledge is a two party protocol in which whenever one party (the verifier) is *"convinced"* then the other party (the prover) indeed *"knows"* something (i.e., the prover asserts knowledge of some object and not merely its existence) [BG93]. For the formalization of PoKs we are going to pay attention to the word *"knows"*. Particularly, what is meant by saying that a machine knows something?

The above question is an indication of the difficulty to a straightforward definition of this new notion. As a behaviour, knowing something means there is the ability to *"write"* down this information. In the context of interactive machines, how a machine write this information? Should it appear in the memory of the machine at some point (since the interactive aspect of a machine is its input-output behaviour)? If so, the machine does not necessarily spit out this information or in other words: if we can pull the information out of the machine, it must somehow have been there. Pulling out this information means that the machine can be *easily modified* to compute this information. More precisely, there exists an efficient algorithm, called the *knowledge extractor*, that using the machine as an oracle, extracts this knowledge.

Suppose a prover and verifier interacting on common input x. With more clarity and less sophistication we require that the probability the verifier accepts the common input is inversely proportional to the difficulty of extracting a witness w of x when using the prover as a black box. Namely, the extractor given access to a function specifying prover's behaviour, its running time is inversely related (by a factor polynomial in $|x|$) to the probability that prover convinces the verifier on accepting x.

Definition 3.2.2 (Message-Specification Function)**.** *Denote by $P_{x,y,r}(\overline{m})$ the message sent by machine P on common input x, auxiliary input y and random input r after receiving messages $\overline{m}$. The function $P_{x,y,r}$ is called the* ***message-specification function*** *of machine P with common input x, auxiliary input y and random input r.*

As we note in the definition of $P_{x,y,r}$ we included, except from the prover's program, its auxiliary and random input. This is because auxiliary as much as random input could enable the prover to "know" and prove more.

Comments on notation $\overline{m}$: As we have said, the interaction between prover and verifier consists of a sequence of moves in each of which one party sends a message to the other. In this interaction the prover or the verifier may move first, let us for simplicity assume the prover moves first and the verifier last. We denote by a_i (resp. b_i) the message sent be the prover (resp. verifier) in the i-th move. We consider with $\overline{m}$ any prefix of conversation, a unique description of messages been sent from both parties so far, i.e., $\overline{m} = a_1 b_1 \dots a_{i-1} b_{i-1}$. Then each message is specified by $P_{x,y,r}$ as follows: $a_i = P_{x,y,r}(a_1 b_1 \dots a_{i-1} b_{i-1})$

The extractor, K, is an algorithm favored with additional properties, these are: its ability to get responses the prover generates in arbitrary messages he gets from K and the ability to rewind the prover to a prior state. We say that K has *rewind black box oracle access* to the prover or that K is an oracle machine and denote it also as $K^{P_{x,y,r}}$.

Definition 3.2.3 (System for Proofs of Knowledge)**.** *Let $\mathcal{R}$ be a binary relation and $k : \mathbb{N} \to [0,1]$. We say that an interactive function V is a* ***knowledge verifier for the relation*** *$\mathcal{R}$ with* ***knowledge error*** *k if the following two conditions hold:*

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

- ***Non-triviality***: *There exists an interactive machine P such that for every* $(x, y) \in \mathcal{R}$ *all possible interactions of V and P on common input x and auxiliary input y are accepting.*
- ***Validity*** *(with error k): There exists a polynomial* $q(\cdot)$ *and a probabilistic oracle machine K such that for every interactive function P,* $x \in L_{\mathcal{R}}$ *and every* $y, r \in \{0,1\}^*$*, machine K satisfies the following condition:*

  Denote by $p(x, y, r)$ *the probability that the interactive machine V accepts, on input x, when interacting with the prover specified by* $P_{x,y,r}$*. If* $p(x, y, r) > k(|x|)$*, then, on input x and with access to oracle* $P_{x,y,r}$*, machine K outputs a solution* $w \in \mathcal{R}(x)$ *within an accepted number of steps bounded by*

  $$\frac{q(|x|)}{p(x, y, r) - k(|x|)} \tag{3.1}$$

  The oracle machine K is called a universal knowledge extractor.

We can think of the error $k(\cdot)$ as the probability that one can convince the verifier without knowing a correct witness w. Being better than that requires some ability to actually compute the witness. Since $p(\cdot)$ is the probability the prover convinces the verifier to accept on input x, $p(|x|) - k(|x|)$ is the probability the verifier accepts and the prover really knows the witness. When the difference of these probabilities get larger, we expect to be able to perform the extraction with fewer queries.

The above definition might be somewhat an inconvenient view of validity seen as expected running time. We will present and prove an alternative definition seen as success probability. Instead of requiring the knowledge extractor to output a solution within an expected time inversely proportional to the probability that prover convinces verifier on accepting x, that is, proportional to $p(x, y, r) - k(|x|)$, the alternative definition requires the extractor to run in expected polynomial time and output a solution with probability at least $p(x, y, r) - k(|x|)$.

Definition 3.2.4 (Validity with error k, Alternative Formulation)**.** *Let* V*,* $P_{x,y,r}$ *(with* $x \in L_{\mathcal{R}}$*), and* $p(x, y, r)$ *be as in Definition 3.2.3. We say that V satisfies the* ***alternative validity condition with error*** *k if there exists a probabilistic oracle machine K and a positive polynomial q such that on input x and with access to oracle* $P_{x,y,r}$*, machine K runs in expected polynomial time and outputs a solution* $w \in \mathcal{R}(x)$ *with probability at least*

$$\frac{p(x, y, r) - k(|x|)}{q(|x|)} \tag{3.2}$$

Proposition. Let $\mathcal{R}$ be an $\mathcal{NP}$-relation, and let V be an interactive machine. Referring to this relation $\mathcal{R}$, machine V satisfies (with error k) the validity condition of Definition 3.2.3 if and only if V satisfies (with error k) the alternative validity condition of Definition 3.2.4.

Proof Sketch. Suppose that V satisfies the alternative formulation (with error k), and let K be an adequate extractor and q an adequate polynomial. Using the hypothesis that $\mathcal{R}$ is an $\mathcal{NP}$-relation, it follows that when invoking K we can determine whether or not K has succeeded. Thus, we can iteratively invoke K until it succeeds. If K succeeds with probability $s(x, y, r) \geq (p(x, y, r) - k(|x|))/q(|x|)$, then the expected number of invocations is $1/s(x, y, r)$, which is as required.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Suppose that V satisfies (with error k) the validity requirement of Definition 3.2.3, and let K be an adequate extractor and q an adequate polynomial (such that K runs in expected time $q(|x|)/(p(x,y,r) - k(|x|))$). Let p be a polynomial bounding the length of solutions of $\mathcal{R}$ (i.e., $(x,w) \in \mathcal{R}$ implies $|w| \leq p(|x|)$), the existence of p follows from the hypothesis that $\mathcal{R}$ is an $\mathcal{NP}$-relation.

Then we proceed with up to $p(|x|)$ iterations as follows to extract w: in the ith iteration we run $K^{P_{x,y,r}}(x)$ until the time limit of $2^{i+1} \cdot q(|x|)$.

$$\underbrace{1^{st}\text{iteration}}_{\text{duration } 4\cdot q(|x|)}, \underbrace{2^{nd}\text{iteration}}_{\text{duration } 8\cdot q(|x|)}, \ldots, \underbrace{p(|x|)^{th}\text{iteration}}_{\text{duration } 2^{p(|x|)+1}\cdot q(|x|)}$$

If the ith iteration succeeds with solution w, we halt and output the solution. If it does not succeed, with probability $\frac{1}{2}$ we move to the next iteration, else (with probability $\frac{1}{2}$) we halt and output a special symbol for failure (e.g., $\perp$). So far K is expected to run in time:

$$\underbrace{1 \cdot 4 \cdot q(|x|)}_{1^{\text{st}}\text{run}} + \underbrace{2^{-1} \cdot 8 \cdot q(|x|)}_{2^{\text{nd}}\text{run}} + \ldots + \underbrace{2^{-(p(|x|)-1)} \cdot 2^{p(|x|)+1} \cdot q(|x|)}_{p(|x|)^{\text{th}}\text{run}}$$

In case these iterations fail to provide a solution we continue by exhausting all possible solutions. Since our solution is bounded by $p(|x|)$, there are $2^{p(|x|)}$ many possible solutions and thus $2^{p(|x|)} \cdot \text{poly}(|x|)$ time needed to try, at worst case, all of them. So, as before, with probability $\frac{1}{2}$ we move on to exhaustive search. Adding this search time to the above summation, the expected time is:

$$\sum_{i=1}^{p(|x|)} 2^{-(i-1)} \cdot (2^{i+1} \cdot q(|x|)) + 2^{-p(|x|)} \cdot (2^{p(|x|)} \cdot \text{poly}(|x|))$$
$$= 4 \cdot p(|x|) \cdot q(|x|) + \text{poly}(|x|)$$

To evaluate the success probability of the new extractor, note that the probability that $K^{P_{x,y,r}}(x)$ will run for more than twice its expected running time (i.e., twice $q(|x|)/(p(x,y,r) - k(|x|))$) is less than $\frac{1}{2}$. Also observe that in iteration $i' = -\log_2(p(x,y,r) - k(|x|))$ we emulate these many steps (i.e., $2^{i'+1} \cdot q(|x|) = 2q(|x|)/(p(x,y,r) - k(|x|))$ steps). Thus, the probability that we can extract a solution in one of the first i iterations is at least $\frac{1}{2} \cdot 2^{-(i'-1)} = p(x,y,r) - k(|x|)$, as required in the alternative formulation. □

Theorem 3.2.3. *Let $\mathcal{P}$ be a Σ-protocol for relation $\mathcal{R}$ with challenge length t. Then $\mathcal{P}$ is a proof of knowledge with knowledge error 2^{-t}. [HL10]*

Proof. Non-triviality is clear by definition.

For validity, let H be the matrix containing only 0s and 1s with a row for each possible set of random choices ρ by P^*, and one column for each possible challenge value e. An entry $H_{\rho,e}$ is 1 if V accepts with this random choice and challenge, and 0 otherwise. Using P^* as a black-box and choosing a random challenge, we can probe a random entry in H. By rewinding P^*, we can probe a random entry in the same row, i.e., where P^* uses the same internal random coins as before. Our goal is to find two 1's in the same row; using special soundness the resulting two conversations give us sufficient information to compute a witness w for x efficiently.

All we know is that $\epsilon := \epsilon(x)$ equals the fraction of 1-entries in H. Note that this gives no guarantees about the distribution of 1's in a given row. For instance, if we

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

stumbled across a row with a single 1, we will never finish if we keep looking in that same row.

We can however make the following observation about this distribution. Define a row to be heavy if the fraction of 1's along the row is at least $\epsilon/2$. By a simple counting argument, we see that more than half of the 1's are located in heave rows. Indeed, let H' be the sub-matrix of H consisting of all rows that are not heavy, and write h' for the total number of entries in H' and h for those in H. By assumption, the number of 1's in H is $h\epsilon$ and the number of 1's in H' is smaller than $h'\epsilon/2$. Then the number g of 1's in heavy rows satisfies

$$g > h\epsilon - h'\epsilon/2 \geq h\epsilon - h\epsilon/2 = h\epsilon/2.$$

Assume for the moment that

$$\epsilon \geq 2^{-t+2},$$

so that a heavy row contains at least two 1's. In this case, we will show that we can find two 1's in the same row in expected time $O(1/\epsilon)$. This will be more than efficient, since $1/\epsilon$ is less than required by the definition, namely $1/(\epsilon - 2^{-t})$.

Our approach will be to first repeatedly probe H at random, until we find a 1 entry, a "first hit". This happens after an expected number of $1/\epsilon tries$.

By the observation above, with probability greater than 1/2, the first hit lies in a heavy row. Now, if it does (but note that we cannot check if it does), and if we continue probing at random along this row, the probability of finding another 1 in one attempt is $\frac{\epsilon/2 \cdot 2^t - 1}{2^t}$ (2^t are all the entries in a row, times this with $\epsilon/2$ we get the least number of 1's in this heavy row, minus 1 for the first hit we get the least number of 1's remaining), and therefore the expected number T of tries to find the second hit satisfies

$$T = \frac{2^t}{\epsilon/2 \cdot 2^t - 1} \leq 4/\epsilon$$

tries. The inequality follows from the assumption on ϵ we made above. We would therefore be done in $O(1/\epsilon)$ tries, which is good enough, as argued above.

However, with some probability smaller than 1/2, the first hit is not in a heavy row. In that case we might spend too much time finding another 1 (if it exists at all!). To remedy this, we include an "emergency break", resulting in the following algorithm:

1. Probe random entries in H until the first 1 is found (the first hit).

2. Then start the following two processes in parallel, and stop when either one stops:

   Pr_1 Probe random entries in the row in which we found a 1 before, until another 1-entry is found (the second hit).

   Pr_2 Repeatedly flip a coin that comes out head with probability ϵ/d, for some *constant* d (we show how to choose d below), until you get heads. This can be done by probing a random entry in H and choosing a random number among $1, 2, \ldots, d$ - you output heads if the entry was a 1 and the number was 1.

Since d is constant, this algorithm certainly runs in expected time $O(1/\epsilon)$. But of course, what we really want is that Pr_1 finishes first since this will give us the result

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

we want. So we have to make sure that Pr_1 gets enough time to finish before Pr_2, if indeed the first hit is in a heavy row.

The probability that Pr_2 finishes after k attempts is $\epsilon/d \cdot (1-\epsilon/d)^{k-1}$. Using the (crude) estimate $(1-\epsilon/d)^{k-1} \leq 1$, we get that the probability of finishing after k or fewer attempts is at most $k\epsilon/d$. For $k = d/(2\epsilon)$, this bound is $1/2$, so we conclude that the probability that Pr_2 needs more than $d/(2\epsilon)$ trials to finish is at least $1/2$. Now choose d "large", say $d = 16$. This will mean that with probability at least $1/2$, Pr_2 finishes after more than $8/\epsilon$ tries.

As before, if indeed the first hit is in a heavy row, then with probability at least $1/2$, Pr_1 is done after fewer than $2T \leq 8/\epsilon$ tries[2].

Therefore, with probability greater than $1/2 \cdot 1/2 = 1/4$, Pr_1 finishes before Pr_2 in this case.

Overall, this procedure finds two 1's along the same row if we hit a heavy row and the right process finishes first, which happens with probability greater than $1/2 \cdot 1/4 = 1/8$, and it runs in expected time $O(1/\epsilon)$.

The required knowledge extractor now repeats the above algorithm until we have success. Since the expected number of repetitions is constant (at most 8), we obtain an algorithm that achieves its goal in expected time $O(1/\epsilon)$, as desired.

So what if $2^{-t} < \epsilon < 2^{-t+2}$? We treat this case by a separate algorithm, using the fact that when ϵ is so small, we are in fact allowed time enough to probe an entire row. The algorithm we describe then simply runs in in parallel with the above algorithm.

Define δ by $\epsilon = (1+\delta)2^{-t}$; so that $0 < \delta < 3$. Let R be the number of rows in H. Then we have at least $(1+\delta)R$ 1's among the $R2^t$ entries. At most R of these can be alone in a row, thus at least δR of them must be in rows with at least two 1's. Such a row is called semi-heavy. The algorithm now does the following:

1. Probe random entries until a 1 is found.
2. Search the entire row for another 1 entry. If no such entry was found, go to step 1.

To analyze this, note that the fraction of ones in semi-heavy rows is $\delta R/(1+\delta)R = \delta/(1+\delta)$ among all ones and $\delta R/R2^t = \delta/2^t$ among all entries. The expected number of probes to find a 1 is $1/\epsilon = 2^t/(1+\delta)$. The expected number of probes to find a 1 in a semi-heavy row is $2^t/\delta$. So we except to find a one in a semi-heavy row after finding $(1+\delta)/\delta$ 1's. For each 1 we find, we try the entire row, so we spend $O(2^t(1+\delta)/\delta)$ probes on this. In addition, we spend $O(2^t/\delta)$ probes on finding 1's in step 1, so altogether we spend

$$2^t\left(\frac{1}{\delta} + \frac{1+\delta}{\delta}\right) = 2^t\frac{2+\delta}{\delta}$$

which is certainly $O(2^t/\delta)$. But this is no more than the time we are allowed:

$$\frac{1}{\epsilon - k} \geq \frac{1}{\epsilon - 2^{-t}} = \frac{1}{(1+\delta)2^{-t} - 2^{-t}} = 2^t/\delta$$

□

[2] This follows from Markov's inequality: a non-negative random variable is less than twice its expectation with probability at least $1/2$

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

3.3 *AND* and *OR* Compositions

We now formulate two Sigma protocols for proving knowledge of multiple independent witnesses, the *AND* composition, and proving knowledge for one out of a set of witnesses, the *OR* composition.

3.3.1 Conjunction Composition $\mathcal{P}_{AND}$

We define a new relation $\mathcal{R}^{AND}$ (or $\mathcal{R}^{\wedge}$) given two relations $\mathcal{R}_0$ and $\mathcal{R}_1$, and their respected Sigma protocols $\mathcal{P}_0$ and $\mathcal{P}_1$ as follows:

$$\mathcal{R}^{AND} = \{((x_0, x_1), (w_0, w_1)) : (x_0, w_0) \in \mathcal{R}_0 \wedge (x_1, w_1) \in \mathcal{R}_1\}$$

The steps of the protocol $\mathcal{P}_{AND}$ are:

1. Prover's first step:
    a. The prover sets $\mathbf{w} := (w_0, w_1)$ and $\mathbf{x} := (x_0, x_1)$.
    b. He computes a_0 and a_1 respectively from their protocols $\mathcal{P}_0$ and $\mathcal{P}_1$.
    c. Sends to the verifier $\mathbf{a} = (a_0, a_1)$.
2. Verifier sends random t-bit string s to the prover.
3. Prover's next step:
    a. Computes z_0 and z_1 respectively from protocols $\mathcal{P}_0$ and $\mathcal{P}_1$.
    b. Sets $\mathbf{z} := (z_0, z_1)$ and sends it to the verifier.
4. The verifier outputs $V_{AND}(\mathbf{x}, \mathbf{a}, s, \mathbf{z}) := V_0(x_0, a_0, s, z_0) \wedge V_1(x_1, a_1, s, z_1)$.

3.3.2 Disjunction Composition $\mathcal{P}_{OR}$

This construction allows us to prove that given two inputs x_0 and x_1, from two different in general relations $\mathcal{R}_0$ and $\mathcal{R}_1$ we know a witness for one of them without revealing which one is.

The *OR* construction is somewhat more demanding than the $\mathcal{P}_{AND}$ protocol. The provers is asked to prove one instance for each relation (or two if we use the same relation twice). We can prove one instance, for the witness we know, but for other instance we do not have such witness. We can use nevertheless the simulator `Sim` for this relation.

We define a new relation $\mathcal{R}^{OR}$ (or $\mathcal{R}^{\vee}$) given two relations $\mathcal{R}_0$ and $\mathcal{R}_1$, and their respected Sigma protocols $\mathcal{P}_0$ and $\mathcal{P}_1$ as follows:

$$\mathcal{R}^{OR} = \{((x_0, x_1), (w_0, w_1)) : (x_0, w_0) \in \mathcal{R}_0 \vee (x_1, w_1) \in \mathcal{R}_1\}$$

In the following protocol $\mathcal{P}_{OR}$ we consider j to be such that w_j is known to the prover, whereas without loss of generality w_{1-j} is assumed to be unknown to the prover.

The steps of the protocol $\mathcal{P}_{OR}$ are:

1. Prover's first step:
    a. The prover sets $\mathbf{w} := (w_0, w_1)$, $\mathbf{x} := (x_0, x_1)$, and $w_{1-j} = \perp$, since it is unknown.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

b. He computes a_j from protocol $\mathcal{P}_j$.
c. He computes a simulated transcript for the unknown witness by choosing a random c_{1-j} and sets $\texttt{Sim}_{1-j}(x_{1-j}, c_{1-j}) = (a_{1-j}, c_{1-j}, z_{1-j})$.
d. Sends to the verifier $\mathbf{a} = (a_0, a_1)$.

2. Verifier sends random t-bit string c to the prover.

3. Prover's next step:

   a. He sets $c_j = c - c_{1-j}$ and computes z_j according to $\mathcal{P}_j$ using c_j (as challenge), x_j, a_j and w as inputs.
   b. Sends to the verifier c_0, z_0, c_1, z_1.

4. The verifier:

   a. Checks $c = c_j + c_{1-j}$.
   b. Outputs $V_{OR}(\mathbf{x}, \mathbf{a}, s, \mathbf{z}) := V_0(x_0, a_0, c_0, z_0) \wedge V_1(x_1, a_1, c_1, z_1)$.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Chapter 4

Additional Topics on Zero-knowledge Proofs

4.1 Composition of Zero-knowledge Proof Systems

Zero-knowledge Proofs are widely applicable as subroutines in cryptographic protocols and one may require the repetition of the proof subroutine to strengthen the protocol. So, it is natural from a theoretical point of view to question whether the zero-knowledge property is preserved under composition of zero-knowledge proofs.

There are three types of compositions we may consider, namely *sequential*, *parallel* and *concurrent*. In the sequential composition we invoke polynomially many times our protocol, where each invocation follows the termination of the previous one. In parallel composition polynomial (in time) instances of he protocol are invoked at the same pace. That is, all the executions are totally synchronized so that the ith message in all instances is sent exactly or approximately at the same time.

Concurrent composition is a generalisation of the foregoing cases. Here, polynomial instances of protocols are invoked *at arbitrary times and procced at arbitrary pace*. This is an asynchronous rather than synchronous model of communication.

Remarks: In the composition of proof systems we assume that the honest users follow the specified protocol description, that is, the actions of honest users in each execution are independent of the messages they received in other executions. We cannot assume the same for an adversary for his actions in an execution may depend on messages received in other executions. In other words, we require the bare minimum from the honest side in a protocol composition. The reason for this is that in practice, coordination of honest users is typically difficult, keeping track of executions is likely not achievable e.g., between employees in a big cooperation, but an adversary can get in the extra trouble to coordinate its movements.

4.1.1 Sequential Composition

The simplified version of Zero-knowledge proofs (without auxiliary inputs) is not closed under sequential composition [GK96]. Nonetheless, the augmented model (see Appendix A) is closed under sequential composition. Specifically we have the following theorem:

Theorem 4.1.1. *Let P be a prover that is zero-knowledge with respect to auxiliary input on some language L. Suppose that the last message sent by P, on input x, bears a special end-of-proof symbol. Let $Q(\cdot)$ be a polynomial, and let P_Q be an interactive machine that on common input x, proceeds in $Q(|x|)$ phases, each of them consisting of running P on common input*

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

x. Then P_Q is zero-knowledge (with respect to auxiliary input) on L. Furthermore, if P is perfect zero-knowledge (with respect to auxiliary input), then so is P_Q.

4.1.2 Parallel Composition

Unfortunately, parallel composition does not enjoy the same level of freedom as in the sequential composition theorem.

A Counter-Example for Closeness Under Parallel Repetition

We present a protocol that is zero-knowledge but lucks this property under parallel repetition[Gol02b].

Assume P a prover who posses a secret random function $f : \{0,1\}^{2n} \to \{0,1\}^n$. Let V be a verifier and both participating in the following protocol:

The verifier is supposed to send P a binary value $u \in \{0,1\}$. According to u they execute the following cases:

- For $u = 0$ the prover P selects uniformly $\alpha \in \{0,1\}^n$ and sends it to V which is supposed to reply with a pair of n-bit long strings (β, γ). Upon receiving the strings P checks whether or not $f(\alpha\beta) = \gamma$. If the equality is satisfied, the prover sends verifier a secret information.
- For $u = 1$ the verifier is supposed to uniformly select $\alpha \in \{0,1\}^n$ and sends it to P, which selects uniformly $\beta \in \{0,1\}^n$ ad replies with the pair $(\beta, f(\alpha\beta)$.

The protocol is in zero-knowledge. Observe that in case $u = 0$, it is infeasible for the verifier to guess a passing pair (β, γ) with respect to the message α received. Thus, the verifier does not obtain anything from the interaction. In case $u = 1$, the verifier receives a pair that is indistinguishable from any other pair of n-bit strings which are selected uniformly by the prover, since β is selected uniformly and $f(\alpha\beta)$ is random.

The verifier conducts two concurrent executions of the protocol with the prover. In the first session the verifier sends $u = 0$ and $u = 1$ in the other. When V receives prover's message α in the first session, sends α as its own message in the second session, which in return obtains a pair $(\beta, f(\alpha\beta)$ from prover's execution in this second session. Then, V sends the pair $(\beta, f(\alpha\beta)$ as answer to the first session of the prover this time, which satisfy the equality $f(\alpha\beta) = \gamma$ and eventually the verifier gets the secret information.

u=0	**u=1**
$P \overset{\alpha}{\to} V$	dummy step
dummy step	$P \overset{\alpha}{\leftarrow} V$
dummy step	$P \overset{(\beta, f(\alpha\beta)}{\to} V$
$P \overset{(\beta,\gamma)}{\leftarrow} V$	
$P \overset{\text{secret}}{\to} V(f(\alpha\beta) = \gamma)$	

FIGURE 4.1: Information leakage from parallel composition.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

In figure 4.1 we can see the time scheduling the verifier must achieve to gain the secret information. If it is needed, the verifier might send dummy messages in both sessions to make the timing perfect.

The non-closure behaviour of parallel composition is conceptually annoying but in practise we should not be bothered, since in general, cryptographic schemes exhibit more complex issues under parallel than sequential compositions and so ZKPs are not an exception to this behaviour. So, what can we state for the parallel composition about zero-knowledge? Goldreich and Krawczyk in [Gol02a] proved that under intractability assumptions there exist constant number round zero-knowledge protocols for $\mathcal{NP}$ that are closed under parallel composition.

If we let ourselves to be less docile on privacy criteria we can preserve parallel composition.

4.1.3 Witness Indistinguishability (WI)

Witness indistinguishability is a weaker notion than zero-knowledge that suffices for a parallel application. Loosely speaking, WI means that the view of any verifier is "computationally independent" of the witness that is used by the honest prover as auxiliary private input.

Here we will contain the definition of WI in auxiliary inputs as $\mathcal{NP}$-witnesses to the common input. As in the spirit of computational zero-knowledge 2.3.6, "computationally independent" will translate in the definition as computationally indistinguishable, that is, for every two choices of witnesses, the resulting views are computationally indistinguishable. Thus, the following definition:

Definition 4.1.1 (Witness Indistinguishability). *Let (P, V), $L \in \mathcal{NP}$ and V^* be as in the augmented definition [add reference], and let R_L be a fixed witness relation for the language L. We say that (P, V) is witness-indistinguishable for R_L if for every probabilistic polynomial-time interactive machine V^* and every two sequences $W^1 = \{w_x^1\}_{x \in L}$ and $W^2 = \{w_x^2\}_{x \in L}$, such that $w_x^1, w_x^2 \in R_L(x)$, the following two ensembles are computationally indistinguishable:*

- $\{\langle P(w_x^1), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$
- $\{\langle P(w_x^2), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$

Note that witness indistinguishability does not imply zero-knowledge.

WI proofs offer the same robustness in parallel composition as the sequential version. Namely we have the following theorem:

Theorem 4.1.2 (Parallel Composition for WI). *Let $L \in \mathcal{NP}$ and R_L be as in the WI definition and suppose that P is probabilistic polynomial-time and (P, V) is witness indistinguishable for R_L. Let $Q(\cdot)$ be a polynomial, and let P_Q denote a program that on common input $x_1, \ldots, x_{Q(n)} \in \{0,1\}^n$ and auxiliary input $w_1, \ldots, w_{Q(n)} \in \{0,1\}^*$ invokes P in parallel $Q(n)$ times, so that in the ith copy P is invoked on common input x_i and auxiliary input w_i. Then P_Q is witness indistinguishable for $R_L^Q \stackrel{def}{=} \{(\overline{x}, \overline{w}) : \forall i (x_i, w_i) \in R_L\}$, where $\overline{x} = (x_1, \ldots, x_m)$ and $\overline{x} = (w_1, \ldots, w_m)$, so that $m = Q(n)$ and $|x_i| = n$ for each i.*

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

4.1.4 Concurrent Composition

Cryptographic protocols, according to their environment of application, might extend to multi-party computations. In this environments many instances of the protocol are invoked at different times and proceeded in different pace. This allows one or more adversaries to engage in many proofs with a prover in a coordinated way to gain information for an attack to other protocols. They can interleave messages arbitrarily by running protocols in different times.

There are two models of concurrent composition, the *purely asynchronous model* and the *asynchronous model with timing*. The pure asynchronous model requires no assumptions about the underlying communication channels. At a first glance it seems quite hard to construct a zero-knowledge protocol in this model and one can wonder if such protocols even exist. This question has resolved in [RK99] under standard intractability assumptions but regarding the round complexity of concurrent compositions is still an open problem.

The timing model could be stated simply as a *"time control"* model. In this model the actual time of events is meaningful and we try to control it by introducing time-driven operations. Each party holds a local clock and its rate is bounded by a global bound. All parties know a bound on message delivery time. The time-driven operations are `time-out` of in-coming messages and `delay` of out-going messages. Within the timing model we get the benefit of having constructions of concurrent compositions [DNS98].

4.2 Commitment Schemes

Commitment schemes is an essential tool for zero-knowledge protocols mainly for two purposes. One, they allow a verifier to specify his choices ahead of time. An example protocol of this usage follows below. This has an additional advantage of composing proofs in parallel without revealing additional information to the prover[GK96]. Secondly, they allow a prover to *"cut and stich"* a part of a proof that matches a choice of the verifier than sending the whole proof, which is a much more tedious labour computationally. An example of this usage is presented in Chapter 5.

4.2.1 The 3-Coloring Problem

We introduce commitment schemes through a concrete example of a zero-knowledge proof system concerning the 3-coloring graph problem.

A graph G is 3-colorable if its vertices can be coloured with only three colours, such that no two vertices of the same colour are connected by an edge.

Formally, a graph $G = (V, E)$[1] is 3-colorable if there exists a mapping $\phi : V \to \{1, 2, 3\}$ such that $\phi(u) \neq \phi(v)$ for every $u, v \in E$. The language of all 3-coloring graphs is denoted with G3C.

What makes the notion of 3-coloring interesting is that, for some graphs, it can be quite hard to find a solution of coloring or even to determine if a solution even exists. The 3-coloring problem (decide if a graph is 3-colorable) is known to be in the class $\mathcal{NP}$-complete, thus it is natural to wonder if someone can prove in zero-knowledge that he posses a graph that is 3-colorable.

[1] For a definition of a Graph look at Chapter 6

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

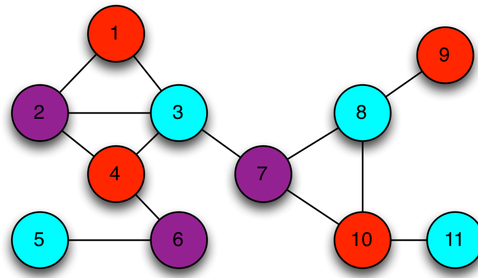


FIGURE 4.2: An example of a 3-colored graph.

We start with the following protocol:

1. *Let G be a 3-coloring graph, denoted with ϕ the 3-coloring map (this is the auxiliary input for the prover).*

2. *The Prover randomly selects a permutation π over $\{1, 2, 3\}$ and sets $\psi(u) = \pi(\phi(u)) = \pi(\phi_u))$ for each $u \in V$.*

3. *The Verifier chooses a random (i, j) from E and sends this choice to the prover.*

4. *The Prover then reveals to the Verifier the permuted colors of vertices i and j, namely $\pi(\phi_i)$ and $\pi(\phi_j)$.*

If the colors of the revealed vertices are different the verifier is "convinced" (if it is needed the challenge (i, j) is repeated, picking each time a new random edge). Obviously there is a catch. The Prover, knowing the protocol, might be lying and each time he is asked to response with the colored vertices he just sends two different colored vertices.

Fortunately, there is a clever way to go around this problem. What we need is a mechanism that forces the prover to ***commit*** to his choice and no longer changed it at his will, but only he is obligated to reveal it whenever he is asked to. This is a concept known in cryptography as a *commitment scheme*.

We can view the commitment as a box that inside it is placed a secret and then it is locked. The box is given to the verifier and whenever he wishes he can get the key from the prover, open the box and see the secret.

Utilizing a commitment scheme we can coerce the Prover, after the permutation of the vertices, to commit the color of all vertices and send them to the Verifier. Then, the Verifier chooses his random edge, asks for the keys of the respected vertices of that edge, unlocks the respected boxes containing the vertices, and their colors are revealed. The "locking" allows the verifier to accept the result since the prover cannot tricked him in sending a false coloring of this edge. Additionally, the prover is safe sending the locked vertices since the verifier is unable to open them without a key and thus gaining nothing in respect to the full graph.

There are two basic properties for this "box-locking" scheme which are essential to any commitment scheme. When a locked box (with a message in it) is given to the verifier, the prover cannot anymore change its content. Hence, when the verifier gets the key and unlocks the box, the revealed message is what the prover committed to originally. This property is called *commitment binding*. Moreover, the prover is

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

reassured that the content of the box stays locked as long as the key is not given to the verifier. This latter property is called *commitment hiding*.

4.2.2 Defining Commitment Schemes

Formal definitions of commitment schemes vary strongly in notation and in flavour. For this thesis we present the subsequent definition (an alternative definition considers bit commitments schemes). The flavour relates to the security of the binding and hiding properties.

A commitment scheme is a triple $(\mathcal{G}, \texttt{commit}, \texttt{verify})$ of algorithms which satisfy the following:

- $\mathcal{G}$ is a probabilistic polynomial time algorithm called the generator. It takes as input 1^k (unary representation) and outputs a string pk, called the *public key* (or *public parameter*) of the commitment scheme that must be authentically transferred to all participants.
- The scheme defines for every pubic key pk a function $\texttt{commit}_{pk}(m) = (c, d)$. It takes in a plaintext and outputs a corresponding *commitment* (or *digest*) c and a *decommitment* (or *opening*) string d.
- The verification algorithm $\texttt{verify}(c, d, m, pk)$ takes a commitment c, an opening d, a message m and a key pk and outputs 1 or 0 respectively if m is the original committed message or not[2].

The security properties of commitment schemes can be unconditional, that is, holding against computationally unbounded adversaries, or computational, that is, polynomial time adversaries. Note that no commitment scheme is possible to achieve unconditional (resp. computational) security for both binding and hiding. We distinguish two flavours of security:

1.
   - **Computational Binding:** Means that unless you have "very large" computing resources, then the chances of an adversary being able to change the committed message are extremely small. A more precise formulation is the following: Let $\mathcal{A}$ be a probabilistic polynomial time adversary who takes as input the public key pk (on input 1^k). For outputs c, c' and d, d' of increasing length k, the probability that $\texttt{verify}(c, d, m, pk) = \texttt{verify}(c', d', m, pk)$ is a negligible function in k.
   - **Unconditional Hiding:** Means that an infinite computing power adversary cannot reveal a commitment c with no additional information. That is, the probability ensembles $\{\texttt{commit}_{\mathcal{G}(1^k)}(m)\}_{k\in\mathbb{N}}, \{\texttt{commit}_{\mathcal{G}(1^k)}(m')\}_{k\in\mathbb{N}}$ are equal or statistically close.
2.
   - **Unconditional Binding:** Means that an infinitely powerful adversary cannot change a message after committing to it. That is, m is uniquely determined from $\texttt{commit}_{pk}(m)$.
   - **Computational Hiding:** Means that an adversary who is in probabilistic polynomial time will have a very hard time guessing what is inside a commitment. Formally, over the uniform choice of $pk \leftarrow \mathcal{G}(1^k)$, for two messages $m_0 \neq m_1$ and $\texttt{commit}_{pk}(m_0) = (c_0, d_0), \texttt{commit}_{pk}(m_1) = (c_1, d_1)$ the distributions of c_0 and c_1 are indistinguishable.

[2] An alternative notation\definition is $\texttt{verify}_{pk}(c, d) = m$, thus $\texttt{verify}_{pk}(\texttt{commit}_{pk}(m)) = m$.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

We will encounter an example of commitment scheme called Pedersen's Commitment in the next chapter.

Note that if we bake in the 3-coloring protocol a commitment scheme, we will end up with a zero-knowledge proof.

4.3 Non-Interactive proofs

Interactive protocols compromising from just one move are called *non-interactive zero knowledge proofs* also known as NIZKs. Obviously, the movement consists of the prover sending at once all the proof and the verifier just following the verification process.

We will present an example of non-interactivity and then we will proceed with some examples built on previous ones.

4.3.1 Walkthrough of an Illustrated NIZK (Sudoku Protocol)

Sudoku is a puzzle consisting of a 9×9 grid which is divided into 3×3 sub grids. The puzzle is given with some of its cells already filled with numbers between 1 to 9. The goal is to fill all the empty cells with numbers so that each row, each column and each sub grid contains all the numbers from 1 to 9, as in the figure below.

We will present a protocol that allows a prover to convince a verifier that there is a solution to the Sudoku puzzle and the prover knows it without revealing any information about the solution.

		1			5	6	7	
	2				4	8		
6	7							
3				5				
				4			1	8
					8	2		9
					2	4		
	9	2			7		8	3
	6		1					2

8	3	1	9	2	5	6	7	4
9	2	5	6	7	4	8	3	1
6	7	4	8	3	1	9	2	5
3	1	8	2	5	9	7	4	6
2	5	9	7	4	6	3	1	8
7	4	6	3	1	8	2	5	9
1	8	3	5	9	2	4	6	7
5	9	2	4	6	7	1	8	3
4	6	7	1	8	3	5	9	2

FIGURE 4.3: An instance of a standard Sudoku problem and its solution.

This protocol utilizes physical cards, the face side of each of which has one number between 1 and 9, and all the back sides are identical with a question mark (?).

The protocol proceeds as follows:

1. The prover places three face-up cards to each cell with an existing value and three face-down cards on each cell according to the Sudoku solution.

2. Starting with each row, the verifier picks one card randomly from each cell to make a packet of 9 cards corresponding to the row. The same procedure is applied for each column and each sub grid. In total, the verifier makes 27 packets and passes them to the prover.

3. The prover randomly shuffles the cards in each packet and returns them to the verifier.

4. The verifier flips the cards over and verifies that each packet contains all the numbers from 1 to 9.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

The above protocol satisfies the three properties of zero knowledge protocols. This protocol can be converting into a non-interactive one. We can imagine a *"tamper-proof"* machine that picks all the 27 packets of the prover, shuffles them and gives them back to the verifier for checking. This way the prover can generate the proof at once and the verifier can respond at a later time. Such non-interactive proof let any party use the machine to verify prover's claim. Prover does not have to be present to be challenged.

Notes: The Sudoku protocol belongs to a wider set of ZKPs also know as *physical zero-knowledge proof protocols* which are restricted to everyday items for their execution. The above protocol can be extended to a generalized version with a grid of $n \times n$ cells and sub grids of $k \times k$ cells with $n = k^2$ [Sas+20]. Previous attempts to the Sudoku puzzle but with an extractability error can be found here [Gra+09] (unlike in Sasaki, Miyahara, Mizuki and Sone above, they present three different protocols with zero extractability error).

4.3.2 From Examples to Practice

The question is if we can move from an example to practical computational practises. Once again we will choose a constructive way. Consider the zero-knowledge definition. This implies the existence of a simulator, namely $M(x)$, that can simulate a message indistinguishable from the real conversation for every $x \in L$, and for every $x \notin L$, by soundness the proof of $M(x)$ is rejected by the verifier. Therefore $V(M(x)) = 1 \iff x \in L$, that is, we get an efficient algorithm to decide L in a single message from the prover to the verifier. But this is a trivial case which nevertheless gives us an indication on how to construct a non-interactive proof.

So we will run the protocol with a simulator, more accurately, with an impartial version of the prover. At first, it seems like a dead end to use a simulator since both the simulator and the prover stands on equal grounds. But we want the simulator to have "verifier capabilities", that is to choose random challenges and in this case *random* points out to the random oracle model (ROM). If we have a hash function that is really a random oracle, we can use it to issue the challenges. The prover cannot guess its output and that is where security comes from.

In the random oracle model both the prover and the verifier have access to a hash function modeled as a random oracle. The simulator for the non-interactive proof can program the random oracle, so if an adversary tries to distinguish between a real and a simulated transcript, can make calls to the oracle but the simulator chooses the responses as well.

In this direction of a non-interactive ZKP the Fiat-Shamir heuristic [FS87] comes in play. Their technique converts an interactive proof of knowledge to a non-interactive proof in the random oracle model, even more it can induce a digital signature based on it.

Below we give the programs for the non-interactive prover, the non-interactive verifier and the simulator respectively for the non-interactive Schnorr identification protocol (with H we denote the hash function):

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

$\mathbf{P}(g, w, h = g^w)$:
$r \leftarrow \mathbb{Z}_q$
$a \leftarrow g^r$
$e \leftarrow H(g, h, a)$
$z \leftarrow r + ew$
output $\pi = (a, e, z)$

$\mathbf{V}(g, h = g^w, \pi = (a, e, z))$:
$c \stackrel{?}{=} H(g, h, a)$
$g^w \stackrel{?}{=} a \cdot h^e$

$\mathbf{S}(g, h)$:
$z \leftarrow \mathbb{Z}_q$
$e \leftarrow \mathbb{Z}_q$
$a \leftarrow \frac{g^w}{h^e}$
Program $H(g, h, a)$ to be c
output (a, e, z)

Let us see the behaviour of the protocol definitions in the non-interactive case:

The *Completeness* is the same as in the interactive Schnorr protocol. The *Proof of knowledge* is similar to the original extractor, but now when rewinding the prover, the extractor changes the value of the random oracle to obtain two different transcripts with the same commitment. *Zero-knowledge* proceeds the same only that the challenge is not a message from the verifier, but rather a value of the hash function. The simulator programs the random oracle to have the value e it has chosen at the point (g, h, a).

Comments: On one hand, the Fiat-Shamir heuristic is proved to be secure against chosen attack messages [PS96], and this in case random oracles exist. On the contrary, if random oracles do not exist, the Fiat-Shamir heuristic has been proven insecure [GK03]. This is one of the many reasons some researchers have a strong distrust against the ROM.

Random oracles are used as an ideal version of cryptographic hash functions in schemes where strong randomness assumptions are needed. Such schemes are proved secure since they require by an attacker an unintended behaviour from the oracle (which does not happen since it is ideal) or to solve some mathematical problem believed to be hard in order to break it.

There are two main problems raised here. The first is assuming that we have access to ideal hash functions, we can emulate a random unpredictable choice of the verifier. Since the hash function is deterministic, we cannot change the value without changing the previous interactions. So, if a prover is able to cheat with a certain probability, then he can just try different commitments if he does not like the result of the hash function.

The other one turns out to be that random oracles are very difficult to build. The best candidates we have are hash functions and the most common of them are SHA-256 and SHA-512. They suffer from the *"length extension attack"*. Based on $H(\text{string}_1)$ and the length of string_1, this attack allows to compute a valid hash for $\text{string}_1|\text{string}_2$, where $|$ means the concatenation of string_1 and string_2, where string_2 is a string produced by us. This proves that SHA-256 is not an ideal random oracle. Yet, it preserves the most important property for our work, its resistance to collisions or preimages. Also, it is proven that there are secure schemes in the random oracle model for which any implementation by cryptographic hash functions results in insecure schemes [CGH00].

In general, proofs in the ROM are fine, but are never complete enough to cover a practical implementation. An attacker could make use of some interesting property of the hash function to break the protocol, which justify the mistrust of some researchers in using them. So we rely our hopes of security in that parts of the hash functions that do not actual impact security.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

If the random oracle model seems inadequate, the alternative is the *Common Reference String* (CRS) model. The CRS model is a public string that was generated through a trusted setup in which all parties have access to the same string taken from some distribution. The common reference string may have an arbitrary distribution, it is a generalization of the *Common Random String model*, in which the distribution of bit strings is uniform. The main difference between ROM and CRS is that the later does not rely on heuristic belief system that the real protocol uses a standard hash function is secure.

In [BFM88] it is shown the first result of a public key cryptosystem using a common reference string shared between the prover and the verifier sufficiently to achieve zero-knowledge without requiring interaction. In [Blu+91] the $\mathcal{NP}$-language of satisfiability is proved to posses non-interactive zero-knowledge proofs using short random strings shared beforehand between prover and verifier.

The CRS model has a main disadvantage. In order to have a string we need to generate it under a trusted setup. This setup is often a *multi-party computation* (MPC) protocol, a method for parties to jointly compute a function over their inputs while keeping those inputs private . As long as we believe that not all the participants colluded, then we can trust the system from that point on. Zcash, a cryptocurrency which provide enhanced privacy for its users, is a well known example for using an MPC protocol [Zca19].

4.4 $\mathcal{NP}$ Problems and $\mathcal{ZKP}$ Results

Let us consider again the 3-coloring graph ZKP protocol. The frequent occurrence of this protocol, for anyone studying ZKPs, is due to its importance as a link between the $\mathcal{NP}$ class and Zero-knowledge proofs. Specifically, the $\mathcal{NP}$-complete problem of 3-coloring can be used to present a zero-knowledge proof system for every language in $\mathcal{NP}$, this is probably the most celebrated result of Zero-knowledge Proofs theory proved by Goldreich, Micali and Wigderson in [GMW91].

Let us recall first some conventions and facts from computational complexity theory to build the necessary background before presenting this famous result.

Definition 4.4.1 ($\mathcal{NP}$ Class)**.** *A language L is in $\mathcal{NP}$ if there exists a binary relation $R_L \subseteq \{0,1\}^* \times \{0,1\}^*$ and a polynomial $p(\cdot)$ such that R_L can be recognized in (deterministic) polynomial time, and $x \in L$ if and only if there exist a y such that $|y| \leq p(|x|)$ and $(x, y) \in R_L$. Such y is called a* ***witness*** *of x in L.*

The $\mathcal{NP}$ class can simply stated as the collection of all problems for which is difficult to find a solution but once given a solution we can efficiently test its validity.

Definition 4.4.2 (Polynomial Reduction)**.** *We say that the language (problem) $\mathcal{A}$ is* ***polynomial time reducible*** *to language (problem) $\mathcal{B}$ if there exists an algorithm for deciding $\mathcal{A}$ in a time that would be polynomial if we could decide arbitrary instances of language $\mathcal{B}$ at unit cost.*

In polynomial time reduction we transform (reduce) one problem to a different problem that we know how to solve to show an algorithm that solves the former one or to show that we cannot find algorithms for some problems.

Definition 4.4.3 ($\mathcal{NP}$-Complete Problems)**.** *A language (problem) is $\mathcal{NP}$**-complete** if it is in $\mathcal{NP}$ and every language in $\mathcal{NP}$ is polynomially reducible to it.*

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

An $\mathcal{NP}$-complete problem $\mathcal{A}$ means that, since it is included in $\mathcal{NP}$ class, it can be used to simulate every other problem in it.

With the above definitions we can sketch now the original result of [GMW91]. Let x be a string for which the prover wants to prove that is in L, where L is in $\mathcal{NP}$. So, x will be given as a common input to both prover and verifier. Since the prover knows that $x \in L$ he uses a corresponding witness w as auxiliary input. We know that L is polynomial reducible (via Karp's reduction [Kar72]) to G3C and x to a graph G in G3C. Additionally, this reduction has the property that reduces the witness w in polynomial time to a witness w' for 3-colorability of G. The verifier upon receiving x reduces it to the same G (via the same reduction). Then, both the parties invoke a zero-knowledge proof with the common input G while the prover enters the proof with auxiliary input w'.

The above result is succeeded with the help of commitment schemes which is a form of secure encryption function. Commitments schemes are implemented by using any one-way function, but the existence of such functions is still an open conjecture. Thus, the (GMW) theorem assumes the existence of one-way function, that is, it is satisfied if and only if there exist one-way functions.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Chapter 5

Zero-knowledge Databases

In this chapter we introduce Zero-knowledge Databases, an application of Zero-knowledge methods in Databases. That is, for a database D, the verifier queries an x and only learns the corresponding database value $D(x)$ in a provable zero-knowledge way.

5.1 Key-value Databases

Key value databases is a special and simpler type of databases where the data are stored in the form "key-value", see figure below, and are optimized to read and write data of this form, which justifies also the name key value stores. A single key, or several keys are used to fetch the data that are associated with them. The values of these data can be of scalar data types like numbers and strings or more complex objects such as lists, JSON or even a key-value pair. Operations and queries in a key value database are based on keys to insert (or update or delete) values or to retrieve values.

Key value stores are present in most programming paradigms as dictionaries (or arrays or maps), but they are stored in a persistent way. They are managed by Database Management Systems (DBMS) and in comparison with other types of databases they allow horizontal scaling in greater magnitudes. They use compact, efficient index structures to be able to quickly and reliably locate a value by its key, making them ideal for systems that need to be able to find and retrieve data in constant time.

Zero-knowledge Databases are applied to key-value databases through the notion of Elementary Databases (EDBs). More precisely, a key value database can induce a Zero-knowledge Elementary Database (ZK-EDBs) cryptographic scheme that allows a prover to commit to an EDB D so as to be able to prove statements such as "x belongs in D and the value of x is y ($D(x) = y$)".

The construction we will present here is due to Micali, Rabin and Kilian in [MRK03]. and is based on the discrete logarithm assumption (look Chapter 3).

5.2 Building Blocks

We will present natural numbers in their binary expansion whenever are given as an input to an algorithm.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

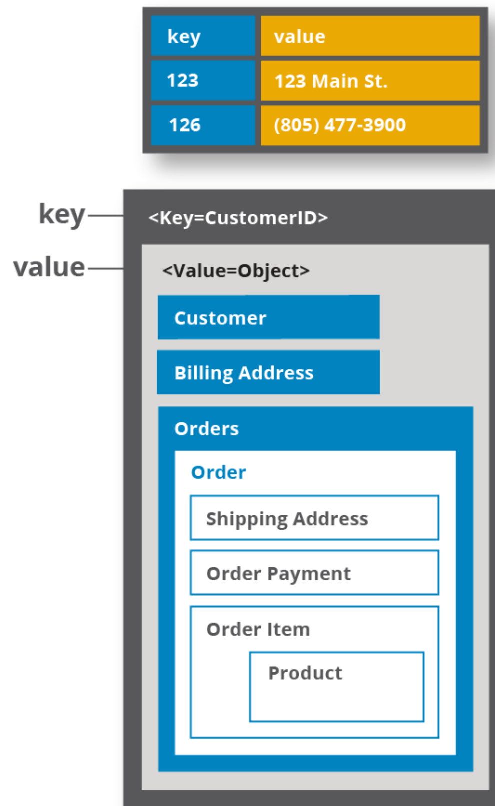


FIGURE 5.1: An example of a key-value pair.

5.2.1 Elementary Databases

By an elementary database we mean a partial[1] function D mapping a (sub)set of *keys* into *values*. Obviously, an EDB has only an elementary functionality, the operation of querying D with a key x and obtain in response the only value $D(x)$ associated with x, either the symbol $\perp$ if no value is associated with x.

More precisely, an EDB D is a function $D : \{0,1\}^* \to \{0,1\}^*$, that is $D \subseteq \{0,1\}^* \times \{0,1\}^*$. D being a function means that if $(x, v_1) \in D$ and $(x, v_2) \in D$, then $v_1 = v_2$, not two pairs have equal first entries but different second entries.

We denote with $[D]$ the *support* of D, that is, the set of finite binary strings x for which, there exists an $v \in \{0,1\}^*$ such that $(x, v) \in D$. Some keys will not assign to any value, for these xs not in the support of D we write $D(x) = \perp$[2].

5.2.2 Zero-knowledge EDBs (ZK-EDBs)

The construction of zero-knowledge elementary databases starts with the owner of the database D who *"pins down"* the database in the form of a commitment but leaks nothing, not even its size, and publishes it. Then, the owner acts as a prover. On querying x, the prover provides proof of $D(x) = y$ or that x lies outside D, while still not revealing any further information about the database.

ZK-EDBs enjoy *completeness*, *soundness* and *zero-knowledge*. Here soundness requires the infeasibility, in polynomial-time, of proving two distinct values v_1, v_2 for

[1] A function which is not defined for some inputs of the right type, that is, for some of a domain.
[2] Actually, an EDB is a function $D : \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ and $[D] = D^{-1}(\{0,1\}^*)$.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

any given x or the infeasibility $D(x) = y$ and $D(x) = \perp$. Zero-knowledge is shown by a simulator that produces in polynomial-time a transcript (of a sequence of proofs for the value of D on querying strings x) distributed identically to that of the real prover knowing only $D(x) = y$ or $D(x) = \perp$.

5.2.3 Pedersen's Commitment Scheme

We form a public quadruple (p, q, g, h), where p and q are prime, $q|p-1$ and g, h are generators for G, the cyclic subgroup of $\mathbb{Z}_p^*$ of order q. All operations are performed modulo p.

1. The committer (or sender) decides (or is given) a secret message m taken in some public message space with at least two elements.
2. Decides a random $r \in \mathbb{Z}_q^*$.
3. Produces a commitment Ped_Commit$((p, q, g, h), m) = (c, r)$, where $c = g^m h^r$ and a verification algorithm Ped_Verify.
4. Makes c public.
5. At a later time reveals m and r.
6. The verifier (or receiver) checks if $c = g^m h^r$, Ped_Verify$((p, q, g, h), c, m, r) =$ accept; else reject.

If someone is able to find in polynomial-time the same commitment to two different messages, implies the ability to compute efficiently the discrete logarithm of h in base g.

Note: The Pedersen commitment scheme can induce collision intractable hash functions H from $\{0, 1\}^*$ to $\{0, 1\}^k$, for some k.

5.2.4 Binary Trees

We denote by $\mathcal{T}_k$ to be the complete binary tree with 2^k leaves. The level of a vertex is defined as the number of edges along the unique path between it and the root node. We label each of the 2^i nodes of $\mathcal{T}_k$ of level i with an i-bit string such that the vertex label u has children labeled $u0$ and $u1$, the root node is labeled e.

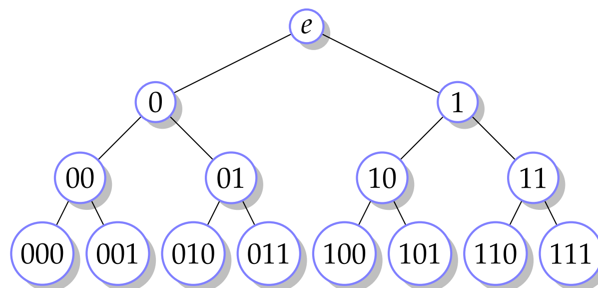


FIGURE 5.2: An example of a $\mathcal{T}_3$ complete labeled binary tree.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

5.3 Construction

We start with a public random reference string σ which is polynomially long in k, the security parameter controlling the probability of error or successful cheating. From σ we can *"extract"* a quadruple (p, q, g, h), thus inducing Pedersen's commitment scheme and from this to induce a collision-free hash function $H : \{0,1\}^* \to \{0,1\}^k$. The mathematical nature of this extraction (commitment scheme plus hash function) can be omitted, what matters the most is that the extraction step is achieved in polynomial-time.

Then, we consider the complete binary tree $\mathcal{T}_k$. In this tree we will place the database D as a tree T as follows:

1. We form the set $H[D]$, this is the set of hashes of all keys x in our database which have a value $y \neq \perp$, that is, $H(x) \in H[D]$ with $D(x) = y$. We put the elements of $H[D]$ to the respective nodes of the tree.

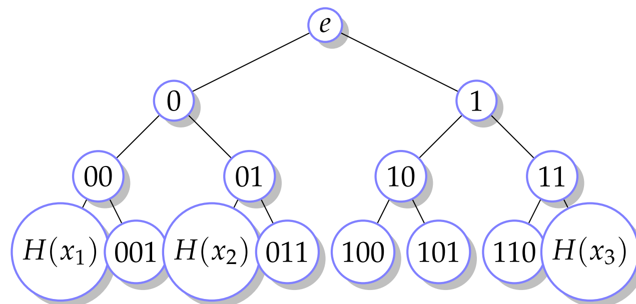


FIGURE 5.3: Nodes of $H[D]$.

2. We construct the subtree $T' = \text{Tree}(H[D])$. This is the union of all the paths from the root to the leaves in $H[D]$.

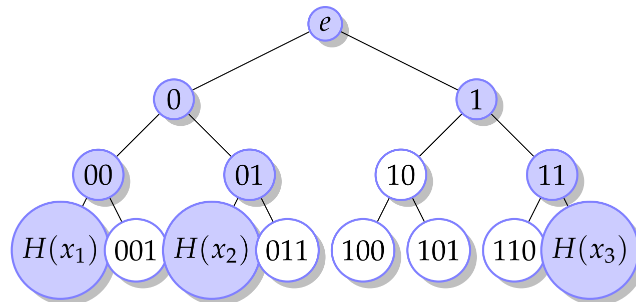


FIGURE 5.4: The light shaded vertices compromise the subtree T'.

3. We obtain T by adding to T' all the nodes of $\mathcal{T}_k$ whose parents are in T'.

In the tree $\mathcal{T}_k$ we call a node *full* if it is the ancestor of at least one leaf in T'; else we call it *empty*.

Next, we will associate and store various quantities in T's nodes. We start with T's leaves. We associate a value m_x as follows: if $x \in H[D]$, that is, $D(x) = y$ for some $y \in \{0,1\}^*$, we associate the leaf $H(x)$ with $m_x = H(y) = H(D(x))$; else, $m_x = 0$. We procced to the nodes of T.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

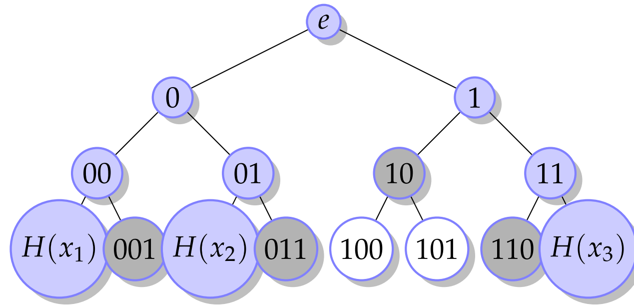


FIGURE 5.5: The light and darkly shaded vertices together compromise the subtree T.

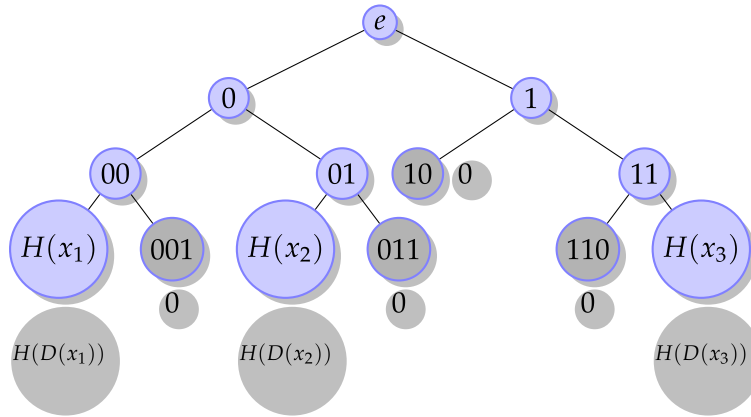


FIGURE 5.6: Subtree T with associated values m_x.

We procced to the nodes u of T. We associate a random exponent e_u of G_q^* ($\mathbb{Z}_q^*$) and store in u the value h_u as follows: if $u \in T'$, then $h_u = h^{e_u}$; else, $h_u = g^{e_u}$.

Note that the leaves of T are populated with m_x values as well as h_u values.

5.3.1 Merkle Tree (Committing steps)

We will store in every leaf u of T a Pedersen commitment c_u as follows: We first associate a random element r_u in $\mathbb{Z}_q^*$ and then store in u the value $c_u = g^{m_u} h_u^{r_u} \pmod{p}$.

We continue by populating the internal nodes respectively with m_u values and commitments c_u in a recursive way starting with the leaves and moving upwards.

1. If an internal node u whose left child u_0 and right child u_1 (note that every internal node has a left and right child since we added in T' all nodes whose parent is in it) are already committed, we store in u the value $m_u = H(c_{u0}, h_{u0}, c_{u1}, h_{u1})$.

2. Then, we continue to the commitment of u at the same fashion as we did with the leaves, that is, $c_u = g^{m_u} h_u^{r_u}$.

We end up with the commitment c_D to the database D which consist of the commitment c_e and the generator h_e stored in the root.

Remarks: When the quadruple (p, q, g, h) is formed, the discrete logarithm of h in base g modulo p, $\log_g h$, is not known to anyone, including the one committing Pedersen's scheme. Thus, the committer does not know $\log_g h_u$ for any full node u but

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

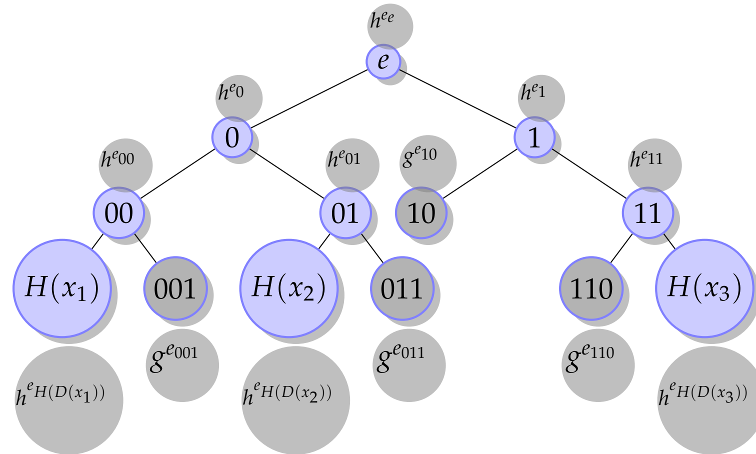


FIGURE 5.7: Subtree T with associated values h_u.

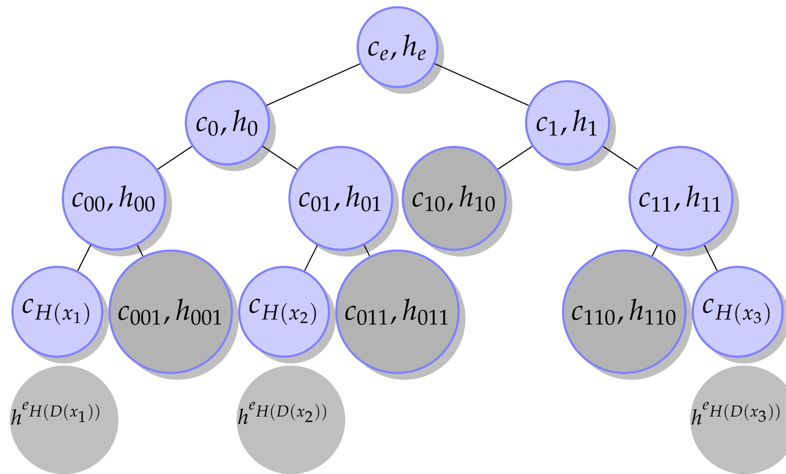


FIGURE 5.8: Merkle Tree of T.

does know it for empty nodes u ($h_u = g^{e_u}$). For the same reason full leaves have *genuine* commitments c_u, that is, the committer cannot decommit c_u to any string other than m_u. On the contrary, empty leaves have *fake* commitments, the committer can decommit c_u to any string he wishes ($\log_g h_u = \log_g g^{e_u}$).

We named the resulted tree 5.8 a Merkle Tree because it satisfies a Merkle-tree like property, that is, every internal node is labeled with the hash of its child nodes' hashes. This allows an efficient and secure verification that a leaf is part of the tree by computing the sequence of values stored in the siblings of the nodes along the path from the root to the leaf.

5.4 Proving Database Values

There are different proof steps for $D(x) = y$ and $D(x) = \perp$. We start with the case $D(x) = y$ and we later move on to the later case.

5.4.1 Proof Construction of $D(x) = y$

Let $P_{H(x)}$ be the path from the leaf $H(x)$ to the root. The prover constructs a proof π_x which for every node u along the path $P_{H(x)}$ consists of:

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

- The values m_u, e_u, h_u, r_u and c_u except for $u = e$,
- The values c_v and h_v for u's sibling v.

The sequence of associated and stored values in the siblings of the nodes along the path $P_{H(x)}$ is called the authentication path of $H(x)$.

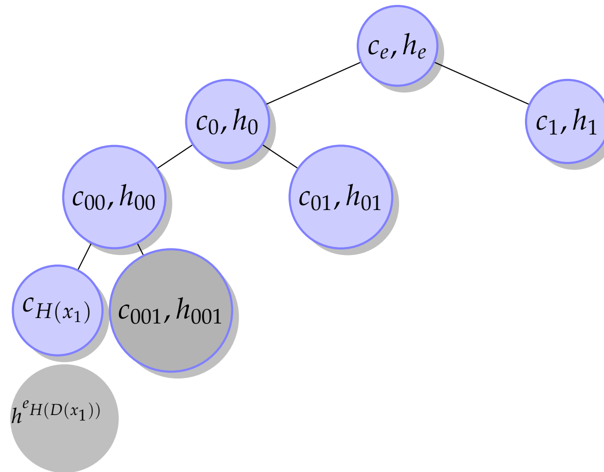


FIGURE 5.9: Authentication path of $H(x)$.

5.4.2 Proof Verification of $D(x) = y$

The verifier:

- Checks $m_{H(x)} = H(y)$ for the leaf of $P_{H(x)}$,
- Checks recursively, for every other node of $P_{H(x)}$, that $m_u = H(c_{u0}, h_{u0}, c_{u1}, h_{u1})$,
- Check for every u in $P_{H(x)}$ that $h_u = h^{e_u}$ and $c_u = g^{m_u} h_u^{r_u} \pmod{p}$,
- Verifies that c_D consists of c_e, h_e.

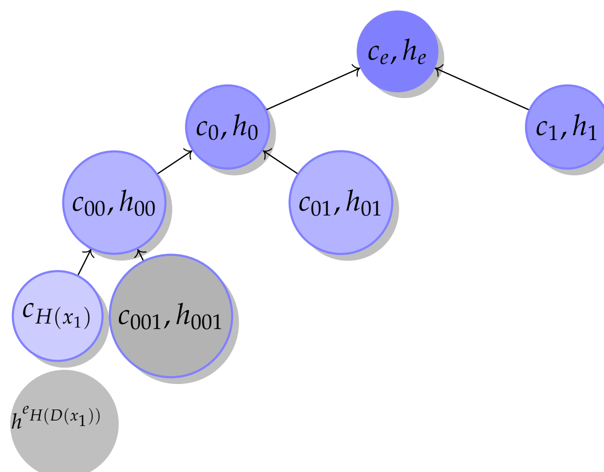


FIGURE 5.10: The recursive verification of proof π_x.

Remember that H is collision resistance, meaning that a malicious prover cannot find two distinct strings with the same hash. Additionally, he cannot decommit any of the values c_u since Pedersen's commitment is computationally binding. Thus, the generated proof π_x is convincing.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

5.4.3 The case $D(x) = \perp$

We first compute $H(x)$ and then, in the tree $\mathcal{T}_k$, we move from the root down towards the leaf $H(x)$ until we find the last node u that also belongs to the subtree T. In our minimal example, see figure 5.5, this node might be $u = 10$ in case of $H(x) = 100$ (or 101) or u is the same leaf $H(x)$ in the case $u = 110$ (in any case, since u is the last node belonging in T we have that $m_u = 0$).

There is a catch here, if we follow the same procedure of generating a proof for x that $D(x) = \perp$ would result decommitting c_u to 0, which in return would reveal additional knowledge: under u we fall outside D's support, that is, we would give away the size (or some of it) of our database. In figure 5.11 imagine that node 6 is the one with $D(6) = \perp$, node 4 is our u and $H(6)$ falls on node 20 associated with 0. Then, a verifier will learn that "under" and "left" from u there are at most 5 hashes $H(x)$ with valid values y giving away the size of $[D]$. We have to find a way and pass smoothly u into our tree T.

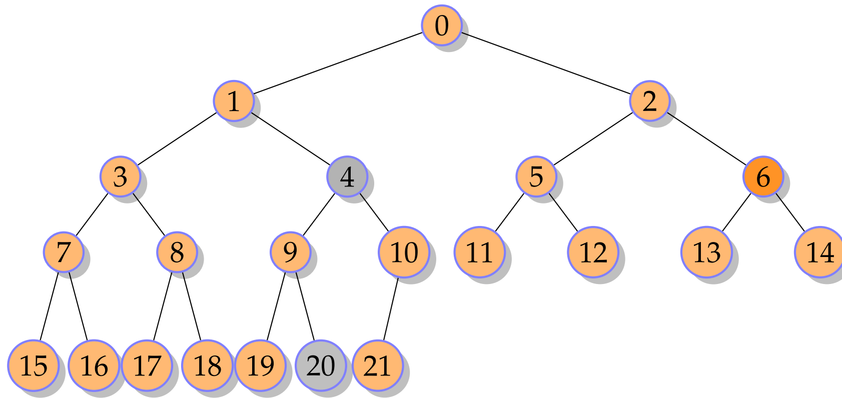


FIGURE 5.11: Anti-pradigm of u betraying D's support.

We form a subtree T_u rooted at u which consists of the subpath from u to $H(x)$ along with the siblings of the nodes in this subpath (except the root u). Our goal is to weld T_u into the old T.

First, we will act on the tree T_u at the same fashion as with T on committing. In each new node v of T_u we choose random e_v in $\mathbb{Z}_q^*$ and we set $h_v = g^{e_v}$ modulo p. For each leaf v in T_u we choose random r_v in $\mathbb{Z}_q^*$ and we set $m_v = 0$ and $c_v = g^{m_v} h_v^{r_v} = g^0 h_v^{r_v}$ modulo p. As before, we move recursively from bottom up to the root of T_u filing the internal nodes with the rest of the values m_v and c_v.

Observe that for the root u, we know the discrete logarithm of h_u in base g. So, we can decommit c_u to the new value m_u by choosing a new r_u such that $c_u = g^{m_u} h_u^{r_u} \pmod{p}$, thus, "welding" T_u into T.

The proof π'_x of $D(x) = \perp$ is the same as in the case $D(x) = y$ with the difference of producing only x. The verification moves in the same fashion but this time we check $m_{H(x)} = 0$.

5.4.4 Dealing with Collisions

We have almost done with the proof cases, but we should beware for collision pitfalls. Let $(x_1, y_1), (x_2, y_2) \in D$. If $H(x_1) \neq H(x_2)$, then we can have two distinct proofs π_{x_1} and π_{x_2} for $D(x_1) = y_1$ and $D(x_2) = y_2$ respectively. But what would happened if the two hashes fall under the same leaf, that is $H(x_1) = H(x_2) = x$?

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

One solution is to degrade our zero-knowledge and provide only one proof, either $m_x = H(y_1)$ or $m_x = H(y_2)$. If we desire to keep the zero- knowledge perfect we can choose longer keys (avoiding collisions) to address the leaves. Thus, store $H(y_1)$ in leaf $P(x_1)$, where $P(\cdot)$ adds a prefix to x. This way, we win in security but losing in efficiency.

We are fortunate enough that H and commit are correlated. The commitment scheme is trapdoor, that is, with an additional information, it allows us to overcome the binding property and to open a commitment ambiguously. The correlation here is that if an H-collision is found, we can decommit arbitrarily strings generated by commit. Furthermore, the discrete logarithm assumption provides very efficient constructs of such *"hash-commitment"* pairs.

So, assume that we have the above collision $H(x_1) = H(x_2) = x$. Without loss of generality lets process first (x_1, y_1), compute $m_x = H(y_1)$ and produce the proof π_x for $D(x_1) = y_1$. For $D(x_2) = y_2$ we go on with the same proof π_x except we follow different way for the values m_x and r_x. We use the trapdoor to compute r'_x for c_x to decommit to $H(y_2)$ and we set $m_x = H(y_2)$ and $r_x = r'_x$.

5.5 On Soundness and Zero-knowledge

We argued that for soundness is infeasible for a dishonest prover to provide for two distinct values $u \neq u'$ proofs $\pi_x, \pi_{x'}$ with the same key x or to find both a proof of $D(x) = y$ and $D(x) = \perp$.

Each proof π_x consists of values m_u, h_u, r_u, c_u and e_u and sibling values c_v and h_v. Let u be the honest value and u' the cheating value. Note that whatever cheating strategy the prover uses he must discipline to the root commitment c_D to achieve his success. Thus, in path $P_{H(x)}$ along the way from the cheating leaf to the root, the values might be different from the "honest" ones but there is a node v (this can even be the leaf node) for which (and the ones above it) $c_v = c_{v'}$ and $h_v = h_{v'}$ and for v's child $v0$ (resp. $u1$) in the path $P_{H(x)}$ we have $(c_{v0}, h_{v0}) \neq (c'_{v0}, h'_{v0})$. Then, one case is $m_v = m_{v'}$ which means the prover has found a collision for H because $m_v = H(c_{v0}, h_{v0}, c_{v1}, h_{v1})$ and $m_{v'} = H(c'_{v0}, h'_{v0}, c'_{v1}, h'_{v1})$. The other possible case is $m_v \neq m_{v'}$. This means that the prover has included in the proof the value of $\log_g h_v$. This, per se, may not be hard to find but the value $\log_h h_v$ is also included in the proof, and given the former value is easy to compute $\log_g h$, which violates the discrete logarithm assumption (since h is not chosen by the prover). Eventually, the cheating prover has to compute something infeasible thus the soundness is satisfied.

Note that we can construct syntactically identical proofs for queries on $D(x) = y$ or queries on $D(x) = \perp$ inversely, that is, from the root commitment down to the leaves. We start with a value PK' as our database commitment and we use trapdoor commitment to generate nodes from the top to the leaves. For $x \in D$ we can create real commitments to the verifier and for $x \notin D$ we can tease commitments to decommit in any values we want.

5.6 The Necessity for Zero-knowledge Databases

Security in ZK-EDBs refers to belonging, that is, provides privacy of "membership" information. Zero knowledge databases can be used as a service to end users in such a way that no one except the user himself can access the data, not even the

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

hosting provider or the database administrator[3]. For example, a user of a database ($x \in D$) is the only one able to see his record $D[x]$. Moreover, it is impossible to prove portions of a record $D[x]$, meaning that we can separate information in the record, like financial and medical information.

A zero-knowledge database decouples the provider from the data stored in the server. The provider neither can release the data nor can be held responsible for the content is stored.

We can additionally increase security by encrypted the data in client's side before sent to the database as a ciphertext. Hence, the data as a ciphertext is free of indexing and calculations.

Disadvantages of ZK databases: When the client has to encrypt all the data it means that all the processing has to be done client- side. As the client server acts only as storage, useful techniques such as de-duplication and compression are not applicable to encrypted data. Even with encryption some meta-data could easily leaked, such as file names, file types and access permissions.

5.7 Recent Advances in ZK-EDBs

Pedersen's commitments can be viewed as an instance of a more general commitment construction named *Mercurial Commitments*. This result was proved be Chase et al. in [Cha+05]. Mercurial Commitments are two-tiered: they allow a partial opening and a true opening. A sender can create a commitment that can be "teased" to any value of his choice. As an example think the proof case of $D(x) = \perp$. There, we weld the new subtree T_u in T by "teasing" the c_u in the value m_u with a suitable r_u. On the contrary, a true opening is binding in the traditional sense, the sender cannot come up with two different opening values for the same commitment.

Good news for efficiency as a "Mercurial" database requires $O(k \cdot |D|)$ commitments. A proof has length $O(k^2)$ and the verifier needs only $O(k)$ mercurial decommitments to accept proof's validity.

Mercurial Commitments spurred further development in ZK-EDBs. One is the Updatable Zero-knowledge Databases [Lis05].

So far a ZK-EDB is "static", once committed one cannot arbitrarily update it. An updatable ZK database allows the update of proofs while maintaining the secrecy of the respected update. For example, a database of people under investigation for criminal activities would be a critical part of a system.

A simple solution would be to commit to a new version of the updatable database from scratch. This would be a tiresome procedure in running time and the holder of the database would have to reissue new proofs for the rest of the keys at an additional expense.

Mercurial Commitments offer an efficient way of updating for both the database owner and the user. More precisely, if an owner wants to update n pairs simultaneously he will have to update $O(n(k - \log n)k$ nodes in time $O(nk^2)$.

In 2019 Libert et al. in [Lib+19] proved richer commitments using Mercurial commitments. ZK databases enable range queries over keys and values. Specifically, we allow queries of the form "Give me all database records $(x, y) \in D$ whose keys x lie within the range $[a_x, b_x]$" or range queries over values "Send me all records

[3] See this Github.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

$(x, y) \in D$ with values y in the interval $[a_y, b_y]$" or analogously of $D(x) = \perp$ we can prove statements like "No key x is assigned the value y" or "y occurs in D with keys in $D^{-1}(y)$". We can, in general, query records in rectangles $[a_x, b_x] \times [a_y, b_y]$.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Chapter 6

Zero-knowledge Graph Databases

Motivated from elementary databases we will research in this chapter the zero-knowledge aspect of graph databases. We start with a presentation of the Graph database notion and the necessity of a zero-knowledge construction. Then, we give basic preliminaries and definitions, and move on in a general construction applicable to any graph database.

6.1 Graph Databases

It turns out that elementary databases handle elements poorly. There are collections of data in which the data itself is of lower priority than the relationships between these data. The data in this case are represented with nodes and edges forming a graph structure where the edges portray the relationship between nodes.

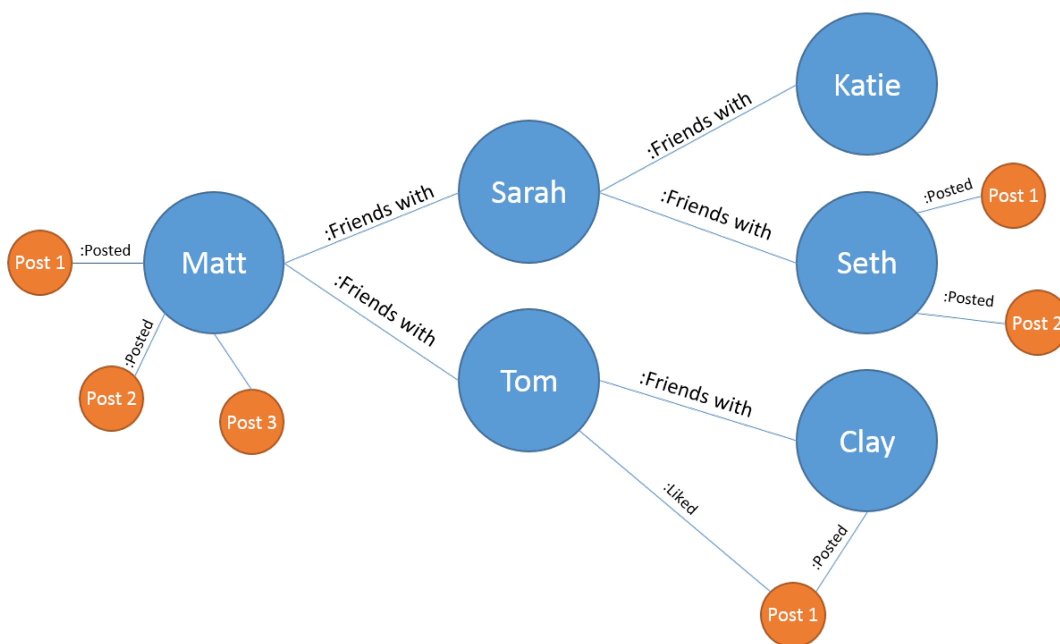


FIGURE 6.1: An example of a Graph Database.

As an example we can think of a social network graph. We can visualize people as nodes and the edges between them as the "friendship" attribute. The nodes can even be businesses, records or documents. The edges can be directed or undirected. Undirected edges are simpler and have a unique meaning while in directed graphs edges have different meaning based on their direction. The "friendship" relationship is an example of undirected edges. As a directed graph we can think of a set of nodes where each node is an algorithm or a program. An edge from an algorithm to a

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

program is the relationship of the algorithm included to the program and an inverse edge is a feedback relation of this program back to the algorithm.

6.1.1 Comparison with Relational Databases

Relational databases are composed strictly in tables to offer flexible row-by-row access. This means that the data are recorded in a predefined model, thus operating much faster on large number of records. This schema though comes with a cost, relational databases are not inherently equipped with a concept of relationship. One example to form complex relations between tables is join operations with several characteristics spanning over (or across) multiple tables. Another is using a foreign key to model a relationship which means spending an additional table as a relationship table. Since the database lucks the concept of relation, this table is stored as user data and thus favored explicitly to the user's motivation and authority.

In a nutshell, relational databases treat data as sets - set oriented databases. Meaning that when expressing relations, the query latency increases rapidly together with the memory.

On the contrary, graph databases are hardwired with the notion of relationship between data. We can create a relation for data on the spot without the bulk of foreign keys or complex logics. Relationships are stored consistently in the database bypassing the need of additional tables, thus data and relations are obtained in one operation. Additionally, the absent of special tables with keys makes the transition from one entity to the next and so on more straightforward, in constant time.

We can say that graph databases are more path oriented. The query latency corresponds directly to the graph size we query.

6.1.2 The Necessity for Zero-knowledge Graph Databases

Relations exist in all areas where a network is present, from simple chat applications, social networks and remote collaborations to businesses and even the banking sector. In social networks relations are first class citizens which address new challenges in security in contrast with the security issues concerning only data.

One such challenge is to hide deep hierarchies. In a remote collaboration for software development, decisions of sensitive parts of the development are left on administrators and thus a malicious adversary is possible to target them in order to steal part of the code before its final release. In this situation we wish to hide this higher level relations where most nodes end up. Moreover, distinct groups of developers specialise in certain parts of the code. These groups form inner relationships in the whole network, yet another sensitive part that we wish to keep hidden.

In social networks one might be connected with a user not directly but through a chain of relationships no matter how distant the relations might be. Most users want to have the option of hiding not only the direct connections but distant connections as well.

6.2 Towards Zero-knowledge Graphs

In elementary databases we were interested in securing individual data with the form of a commitment. In graph databases we are interested mainly in securing individual relations between data. First, let us define the queries in a graph database.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

6.2.1 Querying Graph Databases

A query on a graph database can be of the form "does x relates to y" or "for which y (maybe more than one y), x relates to" or "what is the shortest path between x and y". Except the first query, the other ones might be more complicated to tackle since they deal with more than one relations. So, we will constrain ourselves for the moment to queries of the form "x relates to y".

6.2.2 Directed and Undirected Graphs

We will examine separately directed and undirected graph databases. We will not cover mixed graphs (part of them directed and part of them undirected). We will start with undirected graphs and move on to directed ones.

In all approaches we try to give a general construction of securing a graph database and then we will move on to special cases. Furthermore, as in elementary databases we cared for the most trivial database (key-value form), in graph databases we will care for only one relation between nodes (either this is directed or undirected relation).

6.2.3 Preliminaries

Undirected Graph
An undirected graph G is defined as an ordered pair $G = (V, E)$, where V is the set of vertices and E is a set of unordered pairs of vertices, $E \subseteq \{\{x, y\} : x, y \in V \text{ and } x \neq y\}$.

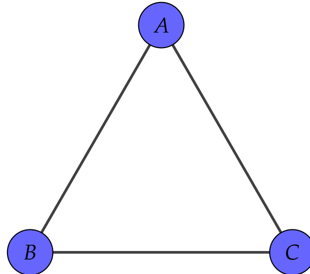


FIGURE 6.2: An uniderected graph example.

Directed Graph
The directed graph is defined with the same terms as undirected ones with the distinction of its edges being oriented, that is, $E \subseteq \{(x, y) : x, y \in V^2 \text{ and } x \neq y\}$.

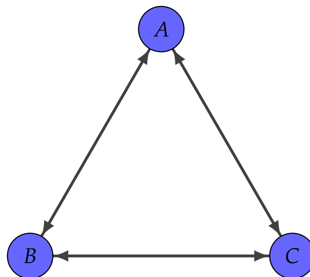


FIGURE 6.3: A directed graph example.

We call the elements of V vertices or nodes. The pairs of E are called ***edges*** or ***links*** in case of undirected graph and ***directed edges*** or ***directed links*** in the case of

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

directed graph. When two vertices x, y are connected with an edge (resp. directed edge), they are called ***adjacent***.

Graph homomorphism

A homomorphism f between two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ is an edge-preserving mapping from G's vertices to H's vertices, that is, the mapping is a function from V_G to V_H such that if $\{u, v\} \in E_G$, then $\{f(u), f(v)\} \in E_H$, for all edges $\{u, v\}$ in G. If such f exists we say that the graphs are homomorphic and we write $f : G \to H$. If f is a bijection and the inverse map is also a graph homomorphism, then it is called graph isomorphism.

In other words, graph homomorphism is a structure-preserving map.

In the directed case the definition is analogous (if $(u, v) \in E_G$, then $(f(u), f(v)) \in E_H$).

Matching of a Graph

Let $G = (V, E)$ be an undirected graph. A *matching* M is a subset of G's edges such that every two edges have no common vertex.

If a path starts and ends with unmatched vertices, and alternates between matched and unmatched edges (between $E \setminus M$ and M), it is called an *augmented path*.

6.3 Construction - Undirected Case

Our goal is more or less in the same spirit of ZK-EDBs, we try to build a commitment such that we can answer queries of the form "x relates to y" or "x does not relate to y" using only the "edge" characteristic and leak nothing else, not even the size of the graph (its number of nodes and relations).

One solution to secure a graph would be to write the graph in an elementary database notation and transform the problem to an elementary database. This is impossible since databases do not behave as functions, there are many-to-many relations. In figure 6.2 we have that B relates to A and A relates to C, so we could write in database style that $D(B) = A$ and $D(A) = C$. However, we also have that $D(B) = C$ which crumbles the D function property and thus we cannot commit in an elementary database fashion.

A different solution would be to use the *adjacent matrix* of the graph. The adjacent matrix of a graph is a square matrix A where each ij-element is 1 when the vertices i, j are adjacent, otherwise is 0. The adjacent matrix of the above undirected graph is:

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Now we can turn this matrix to an elementary database notation as follows: We associate to each ij-element its value in the matrix, that is, $D(ij\text{-element}) = 1$ or 0 respectively for adjacent vertices i, j or non-adjacent vertices i, j. We are able to build a commitment for D but now one can query for relations among vertices that are actually "outside" of the graph. One could query if there exists a relation for e.g., the vertices $i = 4, j = 7$. The answer will be negative, but since it is known that D is actually shaped by a square matrix, one could find after more queries with high probability the size of the matrix, thus the number of nodes.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

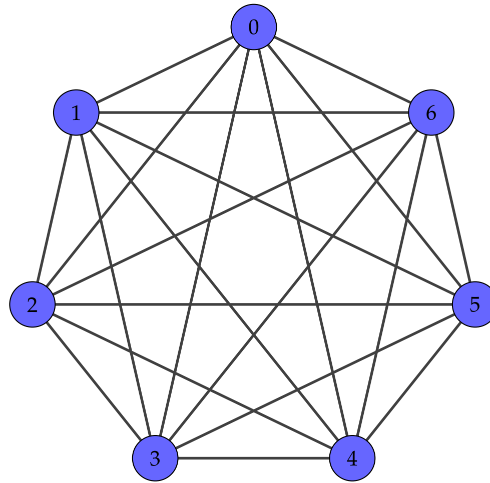


FIGURE 6.4: Graph K_7.

In general, we try to avert from elementary database constructions. Even if an EDB modification of a graph induces a zero-knowledge result this is merely a feasibility result. We aim our construction, as much possible, to be in graph terms rather in EDB terms to mirror the graph structure. In practice, such resemblance issues a suitable implementation and faster proofs in a database which includes explicitly the notion of relationship.

6.3.1 Complete Graphs

The first step in our construction is to obscure the graph structure of our database. That is, we want to place our graph in a "wider" graph structure in such way that the relationships between nodes are blurred in the same way someone places keys of an EDB to the leaves of a binary tree and hides the cardinality and the values of the database.

A candidate for a graph structure that can act as a substrate to carry our graph is a ***complete graph***. A graph where each distinct pair of vertices is connected with a unique edge is called a completed graph. A complete graph with n vertices is denoted as K_n. The next step after we place our database on the complete graph is to commit in an upward fashion the graph. Before we explain what we mean by "commit" in a graph database we will explore more what we mean when we say "upward".

We need to "contract" the graph in a smaller one of the same characteristics, that is, to contract it to a new complete graph with fewer edges. There are various ways we can form a new graph H given a graph G. These are by removing edges and vertices and by contracting edges.

Removing edges and vertices is self-explanatory. Edge contraction is an operation of merging two adjacent vertices into one.

Definition 6.3.1 (Edge Contraction)**.** *Let $G = (V, E)$ be an undirected graph and $e = \{u, v\}$ an edge we wish to contract. Let $G' = (V', E')$ be the graph such that $V' = (V \backslash \{u, v\}) \cup \{w\}$ and $E' = E \backslash \{e\}$.* ***Edge contraction*** *is a function f mapping V' to itself such that for every vertex z in V, its image vertex $z' = f(z)$ in V' is incident to an edge e' in E' if and only if, the corresponding edge, e in E is incident to z in G.*

The above definition is analogous for directed graphs. Note that after an edge contraction we might end up with a multigraph, a graph that has multiple edges

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

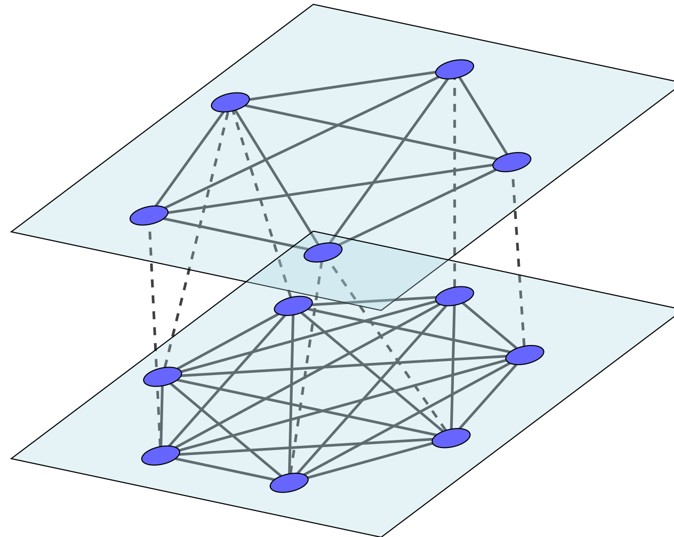
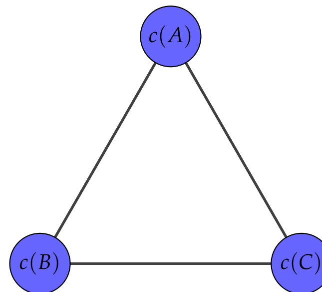


FIGURE 6.5: An edge contraction from K_7 to K_5.



that join the same two vertices. With an "edge remove" operation we scratch away one of the edges to satisfy our undirected graph definition.

6.3.2 Edge Commitment Schemes

We introduce here a form of *edge committing scheme*. Informally, by edge committing we mean a procedure of opaquing the real position of an edge in the graph as we do with data when hiding their real value until open in some time later. Committing data is easy since data are of scalar quantity, they can be represented by a single number (or by a single binary string), whereas edges are unordered pairs $\{u, v\}$ (resp. ordered pairs (u, v) for the directed case). These pairs are made up by two numbers (or binary strings), thus we could just commit the vertices of an edge but this creates a pathological situation.

Let $c(\cdot)$ be a commitment algorithm. If we try to commit the simple graph 6.2 we have the following: The edges $\{A, B\}, \{B, C\}, \{A, C\}$ will commit to $c(\{A, B\})$, $c(\{B, C\}), c(\{A, C\})$:=$\{c(A), c(B)\}, \{c(B), c(C)\}, \{c(A), c(C)\}$. This way of committing does not betray where each vertex A, B, C is coming from but keeps unaltered the structure of the graph, we still end up with a graph similar to the original.

To treat this situation we can just "break" the joints of the graph. Breaking the joints means to split each node into two new nodes, where the new nodes are adjacent to the vertices the former nodes were and form a new edge between them. Breaking blurs the edges position so we have to keep track of the original edges when we need to go backwards and find them on our graph. This construction of a new graph is troublesome and computationally expensive to track the original

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

edges. There is another way to commit this "backwards" movement and open the original position of the edges.

We can represent the "edgeness" of the graph by interchanging the roles of vertices and edges using the line graph[1]. This view of the graph G means that we could commit the nodes of its line graph instead of committing the edges as they are originally. Even better, we do not need to construct the line graph but only label the edges of G as we would labeled them as nodes in the line graph.

There are various ways we can label the edges; one is just to concatenate the adjacent nodes of each edge $\{u, v\}$ and get the label $u|v$. After we have labeled the edges we can use a commitment scheme to commit the label (e.g., the string $u|v$) and open this commitment at a later time as usual.

Definition 6.3.2 (Edge Commitment Scheme)**.** An edge commitment scheme includes the following:

An edge labeling function, $f : E_G \to \{0,1\}^*$ and a triple `Setup`, `Com`, `Ver` of a commitment scheme which acts on the labeled edges.

6.3.3 Committing up to an Edge

So far we have labeled the edges of our graph and committed them. We need now to place our graph, by means of hashing its vertices, on a complete graph K_n. This is the same as placing the keys of an EDB into the leafs of a binary tree. We do not hash each vertex as it is, since we will end up with the same graph as we discussed above when examining the edge commitments. Instead, we can indirectly "label" our nodes.

Let u, v, w be three nodes such that u, v and v, w are adjacent. Let H be a hash function. Consider the strings $H(u|v)$, $H(v|u)$, $H(v|w)$ and $H(w|u)$, and the edges $\{H(u|v), H(v|u)\}$ and $\{H(v|w), H(w|v)\}$. Apparently these edges are not adjacent as the edges $\{u, v\}$ and $\{v, w\}$, and thus we can place them without harm on the complete graph. Note that this hashing of vertices is not strict, one can use any way of hashing he finds meaningful.

All we have to do now is to commit upwards. In a binary tree this was easy, the tree itself was pointing to the manner of committing: sibling nodes are committed up to their parent node; while in a complete graph there is no clear way how to commit upwards.

More precisely, after we identify edges of K_n with the edges of G with hashed vertices, we wish to contract the edges of K_n to a new complete graph $K_{n'}$, with $n > n'$, and then store commitments of the previous contracted edges to new edges. There are many ways to partition a complete graph (here we mean partition of the edge set) and contract the edges accordingly to each partition (contract the edges in each subset of the partition).

This should not be of surprised since we could face the same problem with an EDB. One could place the keys of an EDB to a tree structure in general where the committing phase would not be as clear as in a binary tree (seen as a Merkle tree).

We give a definition of an upward construction that encapsulates all possible ways of an upward commitment.

Definition 6.3.3 (Committing Graph Contraction)**.** Let $G = (V, E)$ be a graph and P_G an edge partition of this graph:

[1] A line graph $L(G)$ of a graph G had as vertices the edges of G and two vertices in $L(G)$ are connected only when as edges in G they have a common node.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

- $\emptyset \notin P_G$
- $\bigcup_{A \in P_G} A = E$
- $\forall A, B \in P_G$ with $A \neq B$, then $A \cap B = \emptyset$

Let $\{f_1, f_2, \ldots, f_n\}_{n \leq |P_G|}$ be an indexed family of homomorphisms from $A_i \in P_G$ to a graph G', with $|E_G| < |E_{G'}|$, $|V_G| < |V_{G'}|$ and $|f_i(A_i)| = 1$, with $f_i(A_i) \cap f_j(A_j) = \emptyset$ for $i \neq j$ and $i, j < n$, and f_n is an injective homomorphism, that is, each f_i maps its respective subset of edges to a unique edge in G' except f_n that maps distinct vertices to distinct vertices. We say that G' is a ***committing contraction*** of G under the indexed family $\{f_i\}_{i \leq |P_G|}$.

Remarks: In practice, we partition the graph G and then we can apply vertex and edge operations in each subset of the partition until we end up to a unique edge. The way we contract the partition's subsets is various and is left on us to find a meaningful way of contracting a subset according to our computational needs. After forming the graph G' we can continue on the next contraction G'' and so on with each graph having its own indexed family $\{f_i'\}$.

6.4 Zero-Knowledge Graph Databases

We describe here formally the notion of a graph database system and then we define zero-knowledge graph databases (ZK-GD). Our protocol is non-interactive and relies on a public random string σ.

Let $G = (V, E)$ be an undirected graph, 1^k a unary string called the *security parameter* (the longer the k parameter the smaller the probability of successful cheating) and σ a *reference string* polynomially long in k. Let P_1, P_2, V probabilistic polynomial-time algorithms defined as follows:

- P_1 algorithm takes as input the triple $(G, 1^k, \sigma)$ and outputs the *public key PK* and the *secret key SK*.
- P_2 algorithm takes as input the above triple together with the public and secret key along with two vertices u, v and outputs a string $\pi_{u,v}$ as the proof.
- V algorithm takes as input the triple $(1^k, \sigma, PK)$ and the proof $\pi_{u,v}$ together with the corresponding vertices and outputs: 1 when accepts the proof, 0 when rejects the proof and $\perp$ when there is not a relation between u and v.

The triple (P_1, P_2, V) is called a ***Graph Database System***.

The above algorithms are executed with the order they are presented here. P_1 and P_2 may coincide and act as one algorithm P (at first committing the graph and subsequently creating the proofs).

Definition 6.4.1 (Zero-knowledge Graph Database)**.** Let (P_1, P_2, V) be a graph database system. We say that (P_1, P_2, V) is a *Zero-knowledge Graph Database* if there exists a positive constant c that satisfies the following:

1. *Perfect Completeness:* For every graph database $G = (V, E)$ and $\forall \{u, v\} \in E$,

$$Pr\left\{ \begin{array}{c} \sigma \leftarrow \{0,1\}^{k^c}; (PK, SK) \leftarrow P_1(G, 1^k, \sigma); \pi_{u,v} \leftarrow P_2(\{u,v\}, SK) : \\ V(1^k, \sigma, PK, \{u,v\}, \pi_{u,v}) = 1 \end{array} \right\} = 1$$

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

2. *Soundness:* $\forall \{u,v\} \in \{0,1\}^* \times \{0,1\}^*$ and for every efficient algorithm P' the following probability is negligible function:

$$Pr\left\{\begin{array}{c} \sigma \leftarrow \{0,1\}^{k^c}; (PK', \pi'_{u,v}, \pi'_{s,t}) \leftarrow P'(1^k, \sigma): \\ V(1^k, \sigma, PK', \{u,v\}, \pi'_{u,v}), V(1^k, \sigma, PK', \{s,t\}, \pi'_{s,t}) \neq 0 \wedge \\ V(1^k, \sigma, PK', \{u,v\}, \pi'_{u,v}) \neq V(1^k, \sigma, PK', \{s,t\}, \pi'_{s,t}) \end{array}\right\}$$

3. *Zero-knowledge:* There exists an efficient (probabilistic polynomial-time) algorithm SIM called the simulator such that for every - possibly dishonest - verifier Adv (the adversary), for every $k \in \mathbb{N}$ and for every graph database G the following views are equal, statistical closed or computational indistinguishable:

$$\begin{array}{ll} \text{view}(k) = & \text{view}'(k) = \\ \{\sigma \leftarrow \{0,1\}^{k^c}; (PK, SK) \leftarrow P_1(G, 1^k, \sigma); & \{(\sigma', PK', SK') \leftarrow SIM(1^k); \\ (\{u_1, v_1\}, s_1) \leftarrow Adv(1^k, \sigma, PK); & (\{u_1, v_1\}, s_1) \leftarrow Adv(1^k, \sigma, PK); \\ \pi_{u_1,v_1} \leftarrow P_2(\{u_1, v_1\}, SK); & \pi'_{u_1,v_1} \leftarrow SIM(\{u_1, v_1\}, SK'); \\ (\{u_2, v_2\}, s_2) \leftarrow Adv(1^k, \sigma, PK, s_1, \pi_{u_1,v_1}); & (\{u_2, v_2\}, s_2) \leftarrow Adv(1^k, \sigma, PK, s_1, \pi_{u_1,v_1}); \\ \pi_{u_2,v_2} \leftarrow P_2(\{u_2, v_2\}, SK); & \pi'_{u_2,v_2} \leftarrow SIM(\{u_2, v_2\}, SK'); \\ \vdots & \vdots \\ : PK, \{u_1, v_1\}, \pi_{u_1,v_1}, \{u_2, v_2\}, \pi_{u_2,v_2}, \ldots\} & : PK', \{u_1, v_1\}, \pi'_{u_1,v_1}, \{u_2, v_2\}, \pi'_{u_2,v_2}, \ldots\} \end{array}$$

The SIM algorithm is given oracle access to the graph database; it calls the database on the pair $\{u,v\} \in \{0,1\}^* \times \{0,1\}^*$ and receives 1 if there is a relation for this pair and 0 otherwise. Before SIM makes oracle calls it outputs σ', PK' and SK', these are the *"fake"* strings of σ, PK, and SK.

Note that for the zero-knowledge property both the commitment on the graph and a sequence of proofs are simulated. The sequence of proofs starts with the adversary sending the asked pair of vertices $\{u_1, v_1\}$, after seeing the proof π_{u_1,v_1} sends the next pair $\{u_1, v_1\}$ and so on. The first view is the computation of public and secret keys given a reference string σ by the prover, and the proofs $\pi_{u_1,v_1}, \pi_{u_2,v_2}, \ldots$ the prover produces respectively for the verifier's queries $\{u_1, v_1\}, \{u_2, v_2\}, \ldots$ The second view is the computation of fake public and secret keys with a reference string σ', and (after the SIM is told the "edgeness" of the pairs $\{u_1, v_1\}, \{u_2, v_2\}, \ldots$) the proofs $\pi'_{u_1,v_1}, \pi'_{u_2,v_2}, \ldots$ it produces.

The above definitions are analogous for directed graphs.

6.5 An example of ZK-GD

We will now give a trivial example, going through all steps of constructing a zero-knowledge graph database for the graph G in figure 6.2. To make notation more readable we change the vertices as a_1, a_2, a_3.

We start with a common reference string σ polynomially long in k.

6.5.1 Contraction and Committing Steps

Step 1. Edge Labeling and Committing

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

As in elementary databases, we calculate from σ a quadruple (p, q, g, h) and we induce a Pedersen commitment and a hash function $H : \{0,1\}^* \to \{0,1\}^k$.

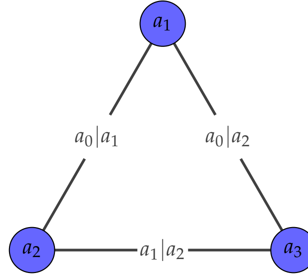


FIGURE 6.6: Edge-Labaled Graph.

We label the edges and committing them in the natural way: $\{a_i, a_j\} \mapsto a_i|a_j$ (see figure 6.6) and $a_i|a_j \mapsto c_{i,j} = g^{a_i|a_j} h^{r_{i,j}}$, where $r_{i,j}$ is a random number in $\mathbb{Z}_q$ (see figure 6.7).

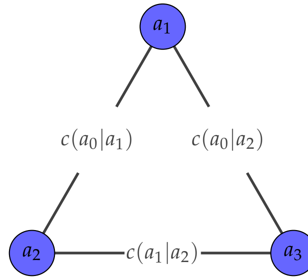


FIGURE 6.7: Committed Labels.

Step 2. Graph Placing

We will place the labeled graph G on a complete graph K_n, where n could be equal to k or substantially longer than k, at least polynomially long in k.

Using the induced hash function, we place our vertices in the following way: we take advantage of the natural order of positive numbers. Let a_i be a vertex, for every edge $\{a_i, a_j\}$, with $i > j$ adjacent to a_i, we hash the vertices as $a_i \mapsto H_{i,j} = H(a_i|a_j)$ and $a_j \mapsto H_{j,i} = H(a_j|a_i)$ on the complete graph. At this point and for the rest of this example, we will identify the graph G with the graph K_7 unless specified differently.

Step 3. Graph Contraction

We will construct a graph contraction G' of G. We start by forming an edge partition P_G.

1. We choose at random a hashed edge $H^{i,j} = \{H(a_i|a_j), H(a_j|a_i)\}$.
2. We find a matching $M^{i,j}$ for the edges $E_G \setminus \{H^{i,j}\}$ such that $H^{i,j} \subseteq \bigcup M^{i,j}$ (the vertices of $H^{i,j}$ are adjacent in edges of the matching).
3. We find an augmented path $p_{i,j}$ between $M^{i,j}$ and $E_G \setminus M^{i,j}$ which includes $H^{i,j}$ and $|p_{i,j}| = 3$.
4. We form the set $A_1 = p_{i,j}$.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

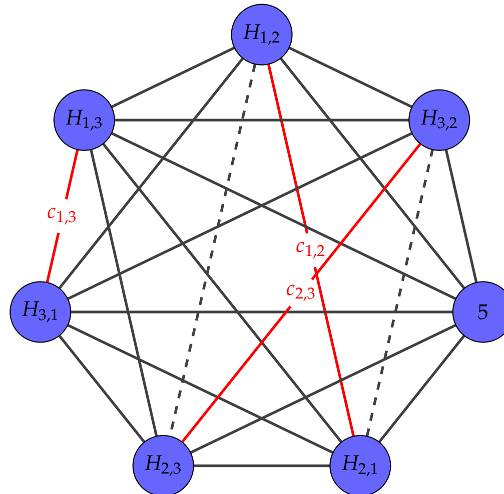


FIGURE 6.8: Committed edges of G placed on the complete graph K_7.

We continue recursively with the set $E_1 = E_G \setminus A_1$ forming the next set A_2 and so on until we exhaust our labeled edges and what is left together with the sets A_k form an edge partition of G.

In figure 6.8 with dashed lines we represent an augmented path for the edge $H_{1,2}$. Note that edge $H_{2,3}$ will have a different path even if the previous path can be also considered augmented for this edge.

Next, we build homeomorphisms f_i for our partition. For our next graph G' we use the complete graph K_5 (two vertices smaller than the previous one). This selection is only for our visual aid in our example.

For each set A_i, except $A_{|P_G|}$, we contract its edges to one edge and place it in K_5 at random e.g., with a new hash function $H : \{0,1\}^* \to \{0,1\}^5$ at the same fashion as we mapped the edges of G in K_7. Note that we translate this procedure, contraction - hashing, to a homomorphism f_i to match our definition of graph contracting.

Step 4. Upwards Committing

At this point our labeled edges together with their augmented path is a partition that looks like the one in figure 6.9 (the non-adjacent vertices might both contain hashes of vertices or no hashes at all). The dashed edges we call them *sibling edges*.

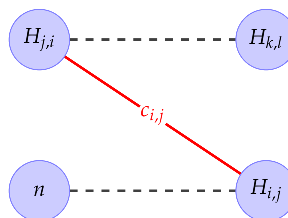


FIGURE 6.9: An augmented path together with a labeled edge as a partition.

In each partition we want to commit in a natural way the labeled edge up to the edge homomorphic to this partition. Since we have pairs of vertices (dashed edges) we will imitate the binary tree commitment.

Committing Steps

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

1. We associate in each vertex n a value m_n as follows: for hashed vertices we set $m_n = H(c_{i,j})$, otherwise we set $m_n = 0$.

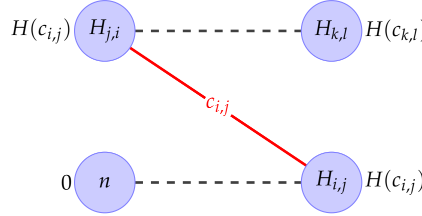


FIGURE 6.10: Associated values m_n.

2. We associate a random exponent e_n of $\mathbb{Z}_q^*$ in each vertex n and we store in it the value h_n as follows: if the vertex is hashed we store $h_n = h^{e_n}$; else, $h_n = g^{e_n}$.

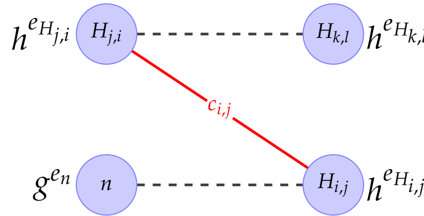


FIGURE 6.11: Associated values h_n.

3. We store commitments c_n in each vertex: we get random $r_n \in \mathbb{Z}_q^*$ and store $c_n = g^{m_n} h_n^{r_n}$ modulo p.

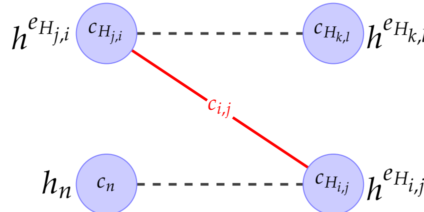


FIGURE 6.12: Stored values c_n.

4. We commit the edges of the upper graph G'. More precisely, we commit the edge homomorphic to the partition below it. The mapping can be constructed in terms of contracting the dashed edges to the upper vertices of the new edge. Then, we store in each new vertex the value $m_{n'_1} = H(c_n, h_n, c_{H_{i,j}}, h_{H_{i,j}})$ and $m_{n'_2} = H(c_{H_{j,i}}, h_{H_{j,i}}, c_{H_{k,l}}, h_{H_{k,l}})$ respectively, and commitments $c_{n'_1} = g^{m_{n'_1}} h_{n'_1}^{r_{n'_1}}$ and $c_{n'_2} = g^{m_{n'_2}} h_{n'_2}^{r_{n'_2}}$ respectively.

It is apparent that we create a finite sequence of graphs $\{G, G', G'', \ldots, G^a\}$, for some $a \in \mathbb{N}$, with each graph having its own partition and finite family of homomorphic functions. In our example, we form all the partitions and functions after G such that the only requirement is every partition be of cardinality 3. Hence, each new partition can carry as much as two committed edges. We will end up with a commitment $C_G = \{c_G, h_G\}$ of the graph database stored in a graph as a single vertex.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

FIGURE 6.13: Upper graph edge commitment.

When committing an edge and place it to its new graph, it is certain that at some point the partitions will contain a committed edge as a sibling edge or two committed edges as sibling edges. The prover has in memory the graphs and all values in vertices and edges that are associated with and stored, and thus he can keep track of the path the committed edge travels.

6.5.2 Proving Graph Relations

We construct the proof for the query *"x relates to y"*.

Let $P_{c_{x,y}}$ be the sequence of edges starting from the labeled edge $c_{x,y}$ in G and traveling up to the vertex commitment C_G.

The proof $\pi_{x,y}$ consists of:

- The values $c_{x,y}$ and $r_{x,y}$,
- The values m_{n_1}, m_{n_2}, e_{n_1}, e_{n_2}, h_{n_1}, h_{n_2}, r_{n_1}, r_{n_2} and c_{n_1}, c_{n_2} of all the committed edges in the sequence of $P_{c_{x,y}}$,
- The values h_{n_3}, h_{n_4} and c_{n_3}, c_{n_4} of the sibling edges in each respected partition of the edges in $P_{c_{x,y}}$.

The verification of the proof consists of:

- Checking that $m_{H_{x,y}} = m_{H_{y,x}} = H(c_{x,y})$,
- Checking recursively that $m_{n'_1} = H(c_{n_1}, h_{n_1}, c_{n_3}, h_{n_3})$ and $m_{n'_2} = H(c_{n_2}, h_{n_2}, c_{n_4}, h_{n_4})$ for every committed edge in the partitions (of the upper graphs edges),
- Checking for every committed edge in the partitions that $h_{n_1} = h^{e_{n_1}}$, $h_{n_2} = h^{e_{n_2}}$ and $c_{n_1} = g^{m_{n_1}} h_{n_1}^{r_{n_1}}$, $c_{n_2} = g^{m_{n_2}} h_{n_2}^{r_{n_2}}$,
- Verifying that $C_G = \{c_G, h_G\}$.

Proof construction for queries of the form *"x does not relate to y"*.

The proof here is analogous to the case of keys that are not in the support of an EDB (that is $D(x) = \perp$).

We consider a "fake" relation between x and y, $\{x, y\}$ and place this edge on the graph G but we set $m_{H_{x,y}} = m_{H_{y,x}} = 0$ and we start to commit upwards as usual. While the vertices of this edge travel upwards the graph sequence $\{G, G', G'', \ldots, G^a\}$ they will meet at some point vertices n_i, n_j with values $m_{n_i} = m_{n_j} = 0$. As in the EDB case we will weld the vertices by teasing their decommitment and provide a proof of "no relation" by checking that $m_{H_{x,y}} = m_{H_{y,x}} = 0$ instead of $m_{H_{x,y}} = m_{H_{y,x}} = H(c_{x,y})$.

Comments: There is the possibility a committed edge does not left with sibling edges. This is in case we exhaust all the edges of the graph taking place in partitions except one, the edge we want to commit. Lets count the edges in our construction

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

in this case: We have a number m of this partitions and 1 edge left committed by the graph under it. Every partition has 3 edges, in total we have $3m + 1$ edges occupying the new graph. Let k be the vertices of this complete graph, this means $\frac{k(k-1)}{2}$ edges. Hence, this committed edge is left "single" in case where the next equality holds:

$$\frac{k^2 - k}{2} = 3m + 1$$

If we increase the graph by one node we will get k additional edges.

In case where the committed edge is left only with one edge, missing an additional edge to create a partition, the same reasoning leads to the same result. In all cases, with an additional node we can have a suitable complete graph to create all the partitions we need. We could also decrease by a node and to a complete graph but we risk to decrease the soundness of the proof.

Queries where there is no relation between two vertices, if there is need for additional vertices, we can always increase our graph while keeping the other partitions and committed edges as they are.

6.6 The Case of Bipartite Graphs

We will consider as a special case of graphs the bipartite graphs and present a different construction of proofs less abstract from the foregoing example.

A *bipartite graph* $G = (V, E)$ is a graph whose vertices can be partitioned into two disjoint sets X and Y, written also as $G = (X \cup Y, E)$ (or $G = (X + Y, E)$), such that every edge connects a vertex in X with a vertex in Y.

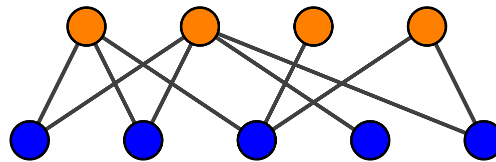


FIGURE 6.14: An example of a Bipartite Graph.

The reason for considering graphs of this type is that there are realistic cases where distinct groups of parties, apart from their inner relations, wish to coordinate their actions together, such as groups of programmers joining their apps development. We can think of a layer where each group has its own graph databases and a different layer at top of this where both groups cooperate as a bipartite graph database.

Bipartite graphs have a predefined form, their adjacent vertices spread among two disjoint sets. Hence, we could think of a specific construction, other than complete graphs, as a substrate to commit their edges. Naturally, a complete bipartite graph comes in mind as a candidate. In a *complete bipartite graph* every vertex of one disjoint set is adjacent to every vertex of the second disjoint set. We only have to consider how to place a bipartite graph and commit its edges.

6.6.1 Bipartite Committing

Let $G = (X \cup Y, E)$ be a bipartite graph, σ a common reference string polynomially long in k the security parameter.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

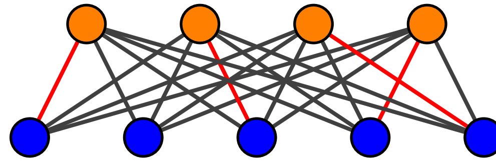


FIGURE 6.15: A Complete Bipartite Graph with hashed vertices in red.

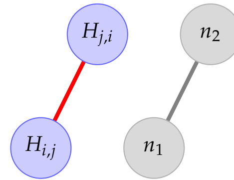


FIGURE 6.16: An example of a labeled-unlabeled edge pair.

We stat by labeling and committing its edges at the same fashion as in our first example. Then, we will place G in a complete bipartite graph $H = (H_V, H_E)$, with $H_V = H_X + H_Y$ of order k, where order is the number of vertices, i.e., $|H_V| = k$.

We could arbitrarily place the edges on H as we did in our example, but this could cause edges with vertices in the same disjoint set. To avoid this problem we choose a different process.

At first, for every disjoint set of G we hash its vertices and place them to a disjoint set i.e., every disjoint set is hashed to a disjoint set. Then, we complete these sets with vertices up to order k as follows: We place every vertex (in lexicographical order) in one disjoint set, starting with vertex 1, until we meet the first hashed vertex. When we encounter such hashed vertex we randomly move to the other set or we move on placing vertices in front of the hashed vertex. We continue at the same fashion until we reach order k. Next, we create the respected labeled edges over the hashed vertices and we fill with with empty edges the non-hashed vertices and hashed vertices until we have a complete bipartite graph H.

At this point we have a bipartite graph as the one in figure 6.15. In this form we can have a more meaningful and less complicated upward committing construction. More precisely, we can couple every labeled edge with hashed vertices, with an unlabeled edge with empty vertices, see figure 6.16 or with another labeled edge and hashed vertices, see figure 6.17. Then, commit in a bottom-up fashion pairs of hashed-non hashed vertices belonging on the same disjoint set to a new vertex (resp. two hashed vertices for two labeled edges) and create a new edge. We end up with a number of new edges for which we can repeat the same process, place them in a new complete graph H' with order $k' < k$ and commit upwards. This will result on a single edge where we commit both of its vertices into a single vertex with associated values $C_G = \{c_G, h_G\}$ as in our example. We can now create proofs $\pi_{x,y}$ of relations $\{x, y\}$ as usual.

We distinguish two cases for queries of non-relations. Let $\{x, y\}$ be the fake relation between two non-adjacent vertices x, y. The hashed vertices could exist in disjoint sets. In this case we can start constructing the proof as usual until we meet a committed edge and weld the proof on it. The second case is where the hashed vertices belong to the same disjoint set. Here, we choose two empty vertices adjacent to the hashed vertices of $\{x, y\}$ and commit according to them.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

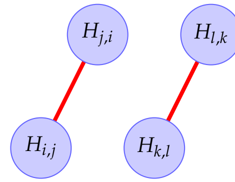


FIGURE 6.17: An example of two labeled edges.

6.7 On Soundness and Zero-Knowledge

A dishonest prover is an adversary who tries to prove the existence of a relation between vertices which does not exist in our graph or tries to prove the absence of a relation for two adjacent vertices. As in EDBs, the adversary must be able to find discrete logarithms or collisions for the hash function H. In any case both are hard to find (negligible probability) hence, the soundness.

Zero-knowledge is also analogous to the ZK-EDBs zero-knowledge.

6.8 Communication Complexity

We analyze the size of the proof our construction requires. So, for each query for relations we commit by performing at most $4d$ hashes and $12d$ modular exponentiations, where d is the number of graphs we use to commit upwards to a vertex. It should be noted here that $d \leq k$ since the construction of these graphs is arbitrary and is left on the way we construct the partitions and how many edges we contract in each partition. In our case we contract 3 adjacent edges, that is, 4 vertices. Hence, we have 4 hashes and 3 times 4 exponentiations (one for h^{r_1}, one for $(h^{r_1})^{r_2}$ and one for g^m for each vertex). A proof of non-relation requires usually less than $4d$ hashes and $12d$ exponentiations since these proofs are constructed by welding them to existing committed edges. Verification of proofs require the same operations.

6.9 Directed Case

There are several solutions to handle directed graphs. One is to use different labels for both directions. A label $a_i|a_j$ defined for one direction and the inverse $a_j|a_i$ for the second direction. This means that every edge has to carry two committed labels and thus twice as many proofs.

The other choice is a pair of hashed vertex and committed label. We can denote each edge $(a_i, a_j), (a_j, a_i)$ as one value which corresponds to a committed label, that is, a function $f : \{0,1\}^k \rightarrow \{0,1\}^*$, where $f(H(a_i|a_j)) = c(a_i|a_j)$ and $f(H(a_j|a_i)) = c(a_j|a_i)$. Hence, we can commit as we would commit a database to a binary tree[2].

[2]Here f has the role of D.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Chapter 7

Conclusions and Open Problems

This thesis presents the notion of zero-knowledge proofs and zero knowledge techniques in constructing proofs with an emphasis in elementary databases and a contribution in graph databases. We believe the first goal achieves to provide a solid theoretical foundation. To this aim, we made extensive use of the basic definition of interactive proofs and zero-knowledge without auxiliary inputs. The number theoretic tools provided in Appendix C encourage to test and experiment with simple protocols. Additionally, in zero-knowledge definition we selected the simpler variant which includes probabilistic polynomial-time distinguisher D rather than non-uniform polynomial-size circuits. In theory, zero-knowledge under the former distinguisher implies zero-knowledge under the latter distinguisher and as an introductory notion it is easier to digest than a non-uniform treatment.

Concerning elementary databases, there have been improvements with the adaptation of mercurial commitments in updatable EDBs and expressive queries. Most recent improvements include proving in zero knowledge that a value has been read or written in a specific position in the database [CDR21].

7.1 Open Questions

Our work in graph databases aims to provide zero knowledge of relations as well as the order of a graph which upon we query these relations. Still, there is room for more complex queries and efficient constructions.

Question 1 *Is it possible to construct a zero-knowledge graph database with less computational overhead?*

In our constructions, for general graphs and bipartite graphs, we have a computational cost (for proofs and verifications) of 4 hashes and 12 modular exponentiations with a factor of d the number of upward graphs taking place in building the commitment PK. We could use only one hash for the vertices of a relation as in the case of directed graphs (considering the undirected edge as one direction only) but then we would have a proof of length $k \geq d$. Another possibility would be a construction which utilizes three instead of four vertices to commit upwards to an edge. In this case we would have a d' between d and k ($d \leq d' \leq k$) but fewer hashing and exponent operations. So, it is natural to wonder is there is a construction with lower additional cost, or if there is a class of graphs that accepts less computational overhead in contrast with other classes of graphs.

Question 2 *Is it possible to keep invariant the commitment of a zero-knowledge graph under updates?*

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Here, the term update responds to creating a new edge between two vertices or removing an edge. We question if under these updates we can keep the commitment PK unchangeable as well as the proofs of the other relations invariant without having to issue new ones.

Question 3 *Is it possible to combine key-value pairs and relations into one zero-knowledge protocol scheme in an efficient way?*

The Pedersen Commitment (or Mercurial Commitments in general) are used in constructing EDBs and Graph Databases in zero-knowledge. We wonder if the same commitment scheme can associate data as well as their relations and create provable queries in zero knowledge such as *"what are the values of all y that x is related with?"* or *"give the values for all records in the rectangle $[a_i, b_i] \times [a_j, b_j]$ which are connected with a path of distance 2 or with degree 1"*.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Appendix A

Augmented Model

Interactive proofs are not stand alone concepts, they may be used as sub-protocols inside larger protocols in which case they do not presented "as is" but rather need additional private inputs associated with the local configurations of the machines before entering the sub-protocol. In the following augmented definitions of interactive proof systems we allow each of the parties to have a private input in addition to the common input.

Augmented definitions of 2.2.1, 2.2.3, 2.2.4 and notation 2.2.1:

Definition A.0.1 (Augmented Interactive Machine)**.** *The interactive machine is defined as before except that has an additional read-only tape called the* ***auxiliary-input*** *tape. The content of this tape is called* ***auxiliary input****.*

The complexity of such an interactive machine is still measured as a function of the (common) input length.

We denote by $\langle A(y), B(z)\rangle(x)$ the random variable representing the (local) output of B when interacting with machine A on common input x, when the random input to each machine is uniformly independently chosen, and A (resp., B) had auxiliary input y (resp., z).

Definition A.0.2 (Augmented Interactive Proof System)**.** *Let P and V be interactive Turing machines with P (called the Prover) computationally unbounded and V (called the Verifier) having probabilistic polynomial-time complexity. Let L be a language, we call the pair (P, V) an* ***interactive proof system for language*** *L if the following two properties are satisfied:*

- **Completeness**: *For every $x \in L$, there exists a string y such that for every $z \in \{0,1\}^*$,*

$$\Pr[\langle P(y), V(z)\rangle(x) = 1] \geq 1 - |x|^{-c} \ (c > 0)$$

- **Soundness**: *For every $x \notin L$, every interactive machine B, and every $y, z \in \{0,1\}^*$,*

$$\Pr[\langle B(y), V(z)\rangle(x) = 0] \leq 1 - |x|^{-c} \ (c > 0)$$

As with interactive proofs, zero-knowledge proofs can also be used as sub-protocols. Thus, it is natural to consider a situation where the verifier when interacting with the prover on common input x, may have some additional a priori information, encoded by a string z, that assist him in extracting knowledge from the prover. What we need is an augmented definition of zero-knowledge. In the spirit of our initial informal statement in page 17 of knowledge we require that whatever can be efficiently computed from x and z after interacting with the prover on any common input x, can be efficiently computed from x and z without any interaction with the prover.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Naturally, we have the following definition:

Definition A.0.3 (Zero-knowledge, Revisited). *Let (P, V) be an interactive proof for a language L as defined above. Denote by $P_L(x)$ the set of strings y satisfying the completeness condition with respect to $x \in L$. We say that (P, V) is **zero-knowledge with respect to auxiliary input** (or is **auxiliary-input zero-knowledge**) if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic (noninteractive) algorithm M^*, running in time polynomial in the length of its first input, such that the following two ensembles are computationally indistinguishable (when the distinguishing gap is considered as a function of $|x|$):*

- $\{\langle P(y_x), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$ for arbitrary $y_x \in P_L(x)$
- $\{M^*(x, z)\}_{x\in L, z\in\{0,1\}^*}$

Namely, for every probabilistic algorithm D with running time polynomial in the length of the first input, for every polynomial $p(\cdot)$, and for all sufficient long $x \in L$, all $y_x \in P_L(x)$ and $z \in \{0,1\}^*$, it holds that:

$$|\Pr[D(x, z, \langle P(y), V^*(z)\rangle(x)) = 1] - \Pr[D(x, z, M^*(x, z)) = 1]| < \frac{1}{p(|x|)}$$

It follows that this definition produces output that is indistinguishable from the real interactions also by non-uniform polynomial-size circuits. Namely, for every non-uniform polynomial-size circuit family $\{C_n\}_{n\in\mathbb{N}}$, every polynomial $p(\cdot)$, all sufficient large n's, all $x \in L \cap \{0,1\}^n$, all $y \in P_L(x)$ and $z \in \{0,1\}^*$:

$$|\Pr[C_n(x, z, \langle P(y), V^*(z)\rangle(x)) = 1] - \Pr[C_n(x, z, M^*(x, z)) = 1]| < \frac{1}{p(|x|)}$$

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Appendix B

Schnorr's Protocol: Proof of Sigma Protocol

Completeness: if $z = r + we$, then $g^z = g^{r+we} = g^r \cdot (g^w)^e = a \cdot h^e$.

Proof of Knowledge: Let P^* be a (possibly malicious) prover that convinces the honest verifier with probability $\delta = 1$. We construct the extractor as follows:

$K^{P^*}(h)$:	
1 :	Run the prover P^* to obtain an initial message a.
2 :	Send a random challenge $e_1 \leftarrow \mathbb{Z}_q$ to P^* and get a response z_1.
3 :	Rewind the prover P^* to its state after the first message.
4 :	Send it another random challenge $e_2 \leftarrow \mathbb{Z}_q$ and get a response z_2.
5 :	Compute and output $x = \frac{z_1 - z_2}{e_1 - e_2} \in \mathbb{Z}_q$.

Since P^* succeeds with probability 1, we know that:

$$g^{z_1} = a \cdot h^{e_1} \text{ and } g^{z_2} = a \cdot h^{e_2}$$

Therefore:

$$\frac{g^{z_1}}{h^{e_1}} = \frac{g^{z_2}}{h^{e_2}} \Rightarrow g^{z_1 - z_2} = h^{e_1 - e_2} \Rightarrow h = g^{\frac{z_1 - z_2}{e_1 - e_2}} \Rightarrow w = \frac{z_1 - z_2}{e_1 - e_2}.$$

Note that the extraction fails if $e_1 = e_2$, which happens with probability $1/q$. Therefore, the knowledge error here is $\epsilon = 1/q$.

Honest-verifier Zero-knowledge (HVZK): For every $g, h \in \mathbb{Z}_q m$, the output of the simulator needs to be indistinguishable from the distribution of the transcripts

$$\{\text{view}_V(P(w,h)) \leftrightarrow V(h)\} = \{g^r, e, r + we : r, e \leftarrow \mathbb{Z}_q\} = \{(a, e, z) : e, z \leftarrow \mathbb{Z}_q, g^z = a \cdot h^e\}$$

We construct a simulator that outputs the same distribution by running the protocol in "reverse":

$M(h)$:
$z \leftarrow \mathbb{Z}_q$
$e \leftarrow \mathbb{Z}_q$
$a \leftarrow \frac{g^z}{h^e}$
output (a, e, z)

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Since z is chosen at random, then the resulting a is random, and the output is distributed identically as the real transcript.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Appendix C

Computational Number Theory and Algorithms

We will present the minimum theory needed for the examples presented in this thesis, namely the Quadratic Residuosity and Discrete Logarithm, along with pseudocodes for practical construction. A full fledged implementation of the aforementioned examples requires an advanced treatment of number theoretic results that can be found in graduate level textbooks of number theory.

Before continue reading recall first the definitions introduced in Chapter 1 and 2.

C.1 Prime Numbers

Cryptography is full of prime numbers. One will often find himself to search for a large prime number or check if a number n is prime or not (primality testing). A method for testing is checking for factors, just divide n with all the numbers a smaller than $n/2$ and if we find a perfect division we conclude that n is not prime. This process is obviously cumbersome, especially for large numbers. A faster way of checking is provided by the following theorem.

Theorem C.1.1 (Root Primality Test)**.** *Every composite number has a proper factor less than or equal to its square root.*

Proof. Suppose n is composite. Then, we can write $n = ab$, where a and b are both between 1 and n. If both a and b satisfy $a > \sqrt{n}, b > \sqrt{n}$, then $ab > \sqrt{n}\sqrt{n} \Rightarrow ab > n$, which contradicts our assumption of $n = ab$. Hence, at least one of a, b is less than or equal to $\sqrt{n}$. That is, if n is composite, then n has a prime factor $p \leq n$. □

So, it suffices to check for factors less than or equal to the square root of n.

C.2 On Quadratic Residues

When making the simulator algorithm of the Quadratic Residue protocol we will need to find the multiplicative inverse x of a congruence class a modulo a number n with respect to this modulus, that is, to find x such that:

$$ax \equiv 1 \pmod{n}$$

But $ax \equiv 1 \pmod{n}$ implies there exists an integer $k \in \mathbb{Z}$ such that $ax - 1 = kn \Leftrightarrow ax - kn = 1$. So, we reduce the problem of finding a multiplicative inverse to the problem of finding integers x, y such that $ax + by = 1$.

We will need at first the help of the following lemma:

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Algorithm 1: Prime Root Test

```
    input  : A number n
    output: Flag 1 in case of prime. Flag 0 in case of composite.
1   flag ⟵ 1;
2   i ⟵ 2;
3   while i ≤ √n do
4   |   if (n = 0 mod i) then
5   |   |   flag⟵ 0;
6   |   |   break;
7   |   end
8   |   i ⟵ i + 1;
9   end
```

Lemma C.2.1 (Bezout's Identity). *Let a and b be integers with greatest common divisor d. Then there exist integers x and y such that $ax + by = d$. Moreover, the integers of the form $ax' + by'$ are exactly the multipliers of d.*

Since we have the relation $ax + by = 1$ this means that there is no other greatest common divisor for a and b than 1 and thus there is a solution x, y. To find such x, y we use the venerable Extended Euclidean Algorithm.

For a recursive algorithm we work backwards up the recursive calls. We start with a trivial solution. Assume, without loss of generality, that $a = d$ and $b = 0$. Then, $x = 1$ and $y = 0$. We have to figure the new coefficients x, y for the transition from (a, b) to $(b, \bmod a)$.

The coefficients x_1, y_1 in the transition $(b, \bmod a)$ will satisfy:

$$b \cdot x_1 + (a \bmod b) \cdot y_1 = g$$

and we want to find x, y such that:

$$a \cdot x + b \cdot y = g$$

We can represent $a \bmod b$ as $a - \lfloor \frac{a}{b} \rfloor \cdot b$, thus:

$$g = b \cdot x_1 + (a \bmod b) \cdot y_1 = b \cdot x_1 + (a - \lfloor \frac{a}{b} \rfloor \cdot b) \cdot y_1$$

After rearranging $g = a \cdot y_1 + b \cdot (x_1 - y_1 \cdot \lfloor \frac{a}{b} \rfloor)$, we get the values $x = y_1$ and $y = x_1 - y_1 \cdot \lfloor \frac{a}{b} \rfloor$[1].

C.3 On Discrete Logarithm

For an example implementation we fix a group G and a generator g of this group and prove for an element $a \in G$ that there exists a least positive integer k such that $g^k = a$.

Definition C.3.1 (Primitive Roots). *A number g is called a primitive root modulo n if every number a coprime to n is congruent to a power of g modulo n. That is, there exists the discrete logarithm k for every a coprime to n.*

[1]See here too.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Algorithm 2: Extended Euclidean Algorithm (Recursive)

input : Integers a and b
output: x and y such that $ax + by = \gcd(a, b) = d$

```
1  ee-algo (int a, int b, int x, int y)
2      if (b = 0) then
3          x ⟵ 1;
4          y ⟵ 0;
5      return x, y;
6      ee-algo (int b, int ⌊a/b⌋, int x_1, int x_2);
7      x ⟵ y_1;
8      y ⟵ x_1 − y_1⌊a/b⌋;
9      return x, y;
10 end
```

Let n be a positive integer. Then, all the integers between 0 and $n-1$ form a group with respect to multiplication modulo n and is denoted as $\mathbb{Z}_n^x$. This group is cyclic if and only if $n = 2, 4, p^k, 2p^k$ where p is an odd prime number. In the case of cyclic group, the generator g of this group is a primitive root modulo n.

To find the number of elements in $\mathbb{Z}_n^x$ we need to find the number of positive integers coprime to n. This is given by Euler's totient function (or Euler's phi function) $\phi(n)$.

Euler's Theorem. *If n and a are coprime positive integers, then:*

$$a^{\phi(n)} = 1 \pmod{n}$$

When there is an integer k for which $a^k = 1 \pmod{n}$ and k is the small integer with this property, then is called the **multiplicative order** of a.

Lets assume now that we have found an $a \in \mathbb{Z}_n^x$ such that $a^{\phi(n)} = 1 \pmod{n}$. This means that the powers of a run over all the elements of $\mathbb{Z}_n^x$ until it reaches 1. Hence, this a is a generator for the group, that is, a primitive root. So to find a primitive root, we just have to find an a for which the Euler's theorem holds and $\phi(n)$ is its order.

C.3.1 Computing $\phi(n)$

Euler's totient function has a number of properties making easy its computation.

1. If p is prime number, then:
$$\phi(p) = p - 1$$

2. If p is a prime number and $k \geq 1$, then:
$$\phi(p^k) = p^k - p^{k-1}$$

3. If a and b are relative prime, then:
$$\phi(ab) = \phi(a) \cdot \phi(b)$$

With these properties we can compute $\phi(n)$ through the factorization of n. Let $n = p_1^{a_1} \cdot p_2^{a_2} \ldots \cdot p_k^{a_k}$, where p_i are the prime factors of n. Then:

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

$$\begin{aligned}\phi(n) &= \phi(p_1^{a_1}) \cdot \phi(p_2^{a_2}) \ldots \cdot \phi(p_k^{a_k}) \\ &= (p_1^{a_1} - p_1^{a_1-1}) \cdot (p_2^{a_2} - p_2^{a_2-1}) \cdot \ldots \cdot (p_k^{a_k} - p_k^{a_k-1}) \\ &= p_1^{a_1} \cdot (1 - \frac{1}{p_1}) \cdot p_2^{a_2} \cdot (1 - \frac{1}{p_2}) \cdot \ldots \cdot p_k^{a_k} \cdot (1 - \frac{1}{p_k}) \\ &= n \cdot (1 - \frac{1}{p_1}) \cdot (1 - \frac{1}{p_2}) \cdot \ldots \cdot (1 - \frac{1}{p_k})\end{aligned}$$

From the last equality we can build an algorithm where n passes from parenthesis to parenthesis and each parenthesis "absorbs" each distinct prime factor from n.

Algorithm 3: Euler's Totient Function

```
   input : integer n
   output: φ(n)
 1 φ ⟵ n;
 2 for (i ← 2, i² ≤ n) do
 3 │ if (n = 0 mod i) then
 4 │ │ while (n = 0 mod i) do
 5 │ │ │ n ⟵ n/i;
 6 │ │ end
 7 │ │ φ ⟵ φ − φ/i;
 8 │ end
 9 │ i ⟵ i + 1;
10 end
11 if (n > 1) then
12 │ φ ⟵ φ − φ/n;
13 end
14 return φ;
```

Comments: Line 7 is the passing from the parenthesis $(1 - \frac{1}{p_i})$. The `while` loop exhausts the rest of $a_i - 1$ primes p_i so we can move to the next prime. Note that when n reaches the last parenthesis, the `for` loop will include the condition $p_k^2 \leq n$. At this point all the primes of n have been exhausted except the last one p_k for which n might included in a power less than 2, thus it will not be counted in the `for` loop and a necessary `if` statement is issued in line 10.

C.3.2 Computing Primitive Roots

In general there is not a known simple general formula to compute primitive roots modulo n. However, there is something we can do to remedy this situation. Let us compute $\phi(n)$ for the group $\mathbb{Z}_n^x$. Let $p_1, \ldots, p_k$ be the different prime factors of $\phi(n)$. Now, suppose the d is a divisor of $\phi(n)$. If we consider the factorization of $\phi(n)$, there is k such that $d \cdot k = \frac{\phi(n)}{p_i}$ for some prime factor p_i of $\phi(n)$. If for this d and an element g in the group we have $g^d \equiv 1 \pmod{n}$, then:

$$g^{\frac{\phi(n)}{p_i}} = g^{d \cdot k} = (g^d)^k = 1^k = 1 \pmod{n}$$

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Hence, if get twice the same result for an element g with two different powers $\frac{\phi(n)}{p_i}$, $\frac{\phi(n)}{p_k}$, where $i \neq j$, then, this g fails as a primitive root candidate. So, if a g has these k results different from 1, it is a primitive root.

Algorithm 4: Primitive Root Algorithm

```
   input  : integer n
   output: primitive root of n
1  flag ⟵ 1;
2  m ⟵ φ(n);
3  // for φ(n) we can subroutine the previous algorithm
4  for (a ← 2, a < n) do
5      flag ⟵ 1;
6      m ⟵ φ(n);
7      for (j ← 2, j^2 ≤ φ(n)) do
8          if (m = 0 (mod j)) then
9              if (a^{φ(n)/j} = 1 (mod n)) then
10                 flag ⟵ 0;
11                 break;
12             end
13             if (m = 0 (mod j)) then
14                 while (m = 0 (mod j)) do
15                     m ⟵ m/j;
16                 end
17             end
18         end
19         j ⟵ j + 1;
20     end
21     if (flag= 1) then
22         break;
23     end
24     a ⟵ a + 1;
25 end
26 if (m > 1) then
27     go to line 8;
28 end
29 return a;
```

Comments: The first `for` loop checks each element of $\mathbb{Z}_n^x$, from the smaller to the largest, if it is a primitive root. The second `for` loop checks a if all powers $\frac{\phi(n)}{p_i}$ are different from 1. The `break` statement breaks the `for` loop for which is nested in. As in Euler's phi algorithm `while` loop serves the purpose of moving to the next prime, and the line 27 moves in line 8 and tests if the last prime remained in the factors of $\phi(n)$.

Note that this algorithm finds the **smallest** primitive root.

With the above algorithms we made two programs in C for the Schnorr non-interactive sequential protocol. The first program, acting as the prover, creates the proof and the second program, as the verifier, tests its validity. We choose as n a prime number to guarantee the existence of a generator. For the non-interactive behaviour we choose

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

```
1  Public parameters:
2  Public prime p and generator g: 6959, 7
3  Private parameters:
4  x: 4512
5  Public h: 1126
6
7  Repetition k: 1 ---> Output (u,c,z):
8  68f_f3b_10c852f
9  Repetition k: 2 ---> Output (u,c,z):
10 3ec_c2e_d6b464
11 Repetition k: 3 ---> Output (u,c,z):
12 d2a_17ef_1a5dd8b
13 Repetition k: 4 ---> Output (u,c,z):
14 14b7_267_2a6ff2
15 Repetition k: 5 ---> Output (u,c,z):
16 ba5_1212_13e859c
17 Repetition k: 6 ---> Output (u,c,z):
18 795_335_388936
19 Repetition k: 7 ---> Output (u,c,z):
20 9c0_161e_185d916
21 Repetition k: 8 ---> Output (u,c,z):
22 72a_274_2b514a
23 Repetition k: 9 ---> Output (u,c,z):
24 e70_70f_7c7553
25 Repetition k: 10 ---> Output (u,c,z):
26 b10_741_7fdae4
```

FIGURE C.1: Non-interactive proof of discrete logarithm.

the Random Oracle Model according to the Fiat-Shamir heuristic and a sequence of 10 repetitions.

The triples the proof sends to the verifier were represented for ease in hexadecimal and separated by an underscore.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

Bibliography

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. "Non-Interactive Zero-Knowledge and Its Applications". In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC '88. New York, NY, USA: Association for Computing Machinery, Jan. 1988, pp. 103–112. ISBN: 978-0-89791-264-8. DOI: 10.1145/62212.62222.

[BG93] Mihir Bellare and Oded Goldreich. "On Defining Proofs of Knowledge". In: *Advances in Cryptology — CRYPTO' 92*. Ed. by Ernest F. Brickell. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1993, pp. 390–420. ISBN: 978-3-540-48071-6. DOI: 10.1007/3-540-48071-4_28.

[Blu+91] Manuel Blum et al. "Noninteractive Zero-Knowledge". In: *SIAM Journal on Computing* 20.6 (Dec. 1991), pp. 1084–1118. ISSN: 0097-5397. DOI: 10.1137/0220068.

[CDR21] Jan Camenisch, Maria Dubovitskaya, and Alfredo Rial. "Concise UC Zero-Knowledge Proofs for Oblivious Updatable Databases". In: *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. June 2021, pp. 1–16. DOI: 10.1109/CSF51468.2021.00008.

[CGH00] Ran Canetti, Oded Goldreich, and Shai Halevi. "The Random Oracle Methodology, Revisited". In: *arXiv:cs/0010019* (Oct. 2000). arXiv: cs/0010019.

[Cha+05] Melissa Chase et al. "Mercurial Commitments with Applications to Zero-Knowledge Sets". In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by Ronald Cramer. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 422–439. ISBN: 978-3-540-32055-5. DOI: 10.1007/11426639_25.

[CRA96] R. CRAMER. "Modular Design of Secure yet Practical Cryptographic Protocols". In: *Ph. D. Thesis, CWI and University of Amsterdam* (1996).

[DH76] W. Diffie and M. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6 (Nov. 1976), pp. 644–654. ISSN: 1557-9654. DOI: 10.1109/TIT.1976.1055638.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. "Concurrent Zero-Knowledge". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. STOC '98. New York, NY, USA: Association for Computing Machinery, Feb. 1998, pp. 409–418. ISBN: 978-0-89791-962-3. DOI: 10.1145/276698.276853.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. "Zero-Knowledge Proofs of Identity". en. In: *Journal of Cryptology* 1.2 (June 1988), pp. 77–94. ISSN: 1432-1378. DOI: 10.1007/BF02351717.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

[FS87] Amos Fiat and Adi Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology — CRYPTO' 86*. Ed. by Andrew M. Odlyzko. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1987, pp. 186–194. ISBN: 978-3-540-47721-1. DOI: 10.1007/3-540-47721-7_12.

[GK03] S. Goldwasser and Y.T. Kalai. "On the (In)Security of the Fiat-Shamir Paradigm". In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* Oct. 2003, pp. 102–113. DOI: 10.1109/SFCS.2003.1238185.

[GK96] Oded Goldreich and Hugo Krawczyk. "On the Composition of Zero-Knowledge Proof Systems". In: *SIAM Journal on Computing* 25.1 (Feb. 1996), pp. 169–192. ISSN: 0097-5397. DOI: 10.1137/S0097539791220688.

[GMR85] S Goldwasser, S Micali, and C Rackoff. "The Knowledge Complexity of Interactive Proof-Systems". In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC '85. New York, NY, USA: Association for Computing Machinery, Dec. 1985, pp. 291–304. ISBN: 978-0-89791-151-1. DOI: 10.1145/22145.22178.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. "Proofs That Yield Nothing but Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems". In: *Journal of the ACM* 38.3 (Apr. 1991), pp. 690–728. ISSN: 0004-5411. DOI: 10.1145/116825.116852.

[Gol01a] "Pseudorandom Generators". In: *Foundations of Cryptography: Volume 1: Basic Tools*. Ed. by Oded Goldreich. Vol. 1. Cambridge: Cambridge University Press, 2001, pp. 101–183. ISBN: 978-0-521-03536-1. DOI: 10.1017/CBO9780511546891.004.

[Gol01b] "Zero-Knowledge Proof Systems". In: *Foundations of Cryptography: Volume 1: Basic Tools*. Ed. by Oded Goldreich. Vol. 1. Cambridge: Cambridge University Press, 2001, pp. 184–330. ISBN: 978-0-521-03536-1. DOI: 10.1017/CBO9780511546891.005.

[Gol02a] Oded Goldreich. "Concurrent Zero-Knowledge with Timing, Revisited". In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. STOC '02. New York, NY, USA: Association for Computing Machinery, Feb. 2002, pp. 332–340. ISBN: 978-1-58113-495-7. DOI: 10.1145/509907.509959.

[Gol02b] Oded Goldreich. *Zero-Knowledge Twenty Years after Its Invention*. Tech. rep. 186. 2002.

[Gra+09] Ronen Gradwohl et al. "Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles". In: *Theory of Computing Systems* 44.2 (Feb. 2009), pp. 245–268. ISSN: 1433-0490. DOI: 10.1007/s00224-008-9119-9.

[GSV98] Oded Goldreich, Amit Sahai, and Salil Vadhan. "Honest-Verifier Statistical Zero-Knowledge Equals General Statistical Zero-Knowledge". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. STOC '98. New York, NY, USA: Association for Computing Machinery, Feb. 1998, pp. 399–408. ISBN: 978-0-89791-962-3. DOI: 10.1145/276698.276852.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

[HL10] Carmit Hazay and Yehuda Lindell. "Sigma Protocols and Efficient Zero-Knowledge1". In: *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Ed. by Carmit Hazay and Yehuda Lindell. Information Security and Cryptography. Berlin, Heidelberg: Springer, 2010, pp. 147–175. ISBN: 978-3-642-14303-8. DOI: 10.1007/978-3-642-14303-8_6.

[JC06] Wei Jiang and Chris Clifton. "A Secure Distributed Framework for Achieving K-Anonymity". In: *The VLDB Journal — The International Journal on Very Large Data Bases* 15.4 (Nov. 2006), pp. 316–333. ISSN: 1066-8888. DOI: 10.1007/s00778-006-0008-z.

[JOP14] Antoine Joux, Andrew Odlyzko, and Cécile Pierrot. "The Past, Evolving Present, and Future of the Discrete Logarithm". In: Nov. 2014, pp. 5–36. ISBN: 978-3-319-10682-3. DOI: 10.1007/978-3-319-10683-0_2.

[JS02] A. Juels and M. Sudan. "A Fuzzy Vault Scheme". In: *Proceedings IEEE International Symposium on Information Theory*, June 2002, pp. 408–. DOI: 10.1109/ISIT.2002.1023680.

[Kar72] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations: Proceedings of a Symposium on the Complexity of Computer Computations, Held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and Sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. The IBM Research Symposia Series. Boston, MA: Springer US, 1972, pp. 85–103. ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9.

[Lib+19] Benoît Libert et al. "Zero-Knowledge Elementary Databases with More Expressive Queries". In: *Public-Key Cryptography – PKC 2019*. Ed. by Dongdai Lin and Kazue Sako. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 255–285. ISBN: 978-3-030-17253-4. DOI: 10.1007/978-3-030-17253-4_9.

[Lis05] Moses Liskov. "Updatable Zero-Knowledge Databases". In: *Advances in Cryptology - ASIACRYPT 2005*. Ed. by Bimal Roy. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 174–198. ISBN: 978-3-540-32267-2. DOI: 10.1007/11593447_10.

[MA78] Kenneth L. Manders and Leonard Adleman. "NP-Complete Decision Problems for Binary Quadratics". In: *Journal of Computer and System Sciences* 16.2 (Apr. 1978), pp. 168–184. ISSN: 0022-0000. DOI: 10.1016/0022-0000(78)90044-2.

[McL90] J. McLean. "Security Models and Information Flow". In: *Proceedings. 1990 IEEE Computer Society Symposium on Research in Security and Privacy*. May 1990, pp. 180–187. DOI: 10.1109/RISP.1990.63849.

[MRK03] S. Micali, M. Rabin, and J. Kilian. "Zero-Knowledge Sets". In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* Oct. 2003, pp. 80–91. DOI: 10.1109/SFCS.2003.1238183.

[PS96] David Pointcheval and Jacques Stern. "Security Proofs for Signature Schemes". In: *Advances in Cryptology — EUROCRYPT '96*. Ed. by Ueli Maurer. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1996, pp. 387–398. ISBN: 978-3-540-68339-1. DOI: 10.1007/3-540-68339-9_33.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14

[Qui+90] Jean-Jacques Quisquater et al. "How to Explain Zero-Knowledge Protocols to Your Children". en. In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by Gilles Brassard. Lecture Notes in Computer Science. New York, NY: Springer, 1990, pp. 628–631. ISBN: 978-0-387-34805-6. DOI: 10.1007/0-387-34805-0_60.

[Rab79] M. O. Rabin. *DIGITALIZED SIGNATURES AND PUBLIC-KEY FUNCTIONS AS INTRACTABLE AS FACTORIZATION*. Technical Report. USA: Massachusetts Institute of Technology, 1979.

[RK99] Ransom Richardson and Joe Kilian. "On the Concurrent Composition of Zero-Knowledge Proofs". In: *Advances in Cryptology — EUROCRYPT '99*. Ed. by Jacques Stern. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1999, pp. 415–431. ISBN: 978-3-540-48910-8. DOI: 10.1007/3-540-48910-X_29.

[Sas+20] Tatsuya Sasaki et al. "Efficient Card-Based Zero-Knowledge Proof for Sudoku". In: *Theoretical Computer Science* 839 (Nov. 2020), pp. 135–142. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2020.05.036.

[Sch90] C. P. Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by Gilles Brassard. Lecture Notes in Computer Science. New York, NY: Springer, 1990, pp. 239–252. ISBN: 978-0-387-34805-6. DOI: 10.1007/0-387-34805-0_22.

[Sha79] Adi Shamir. "How to Share a Secret". In: *Communications of the ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176.

[Zca19] Zcash. *Parameter Generation*. Aug. 2019. URL: https://z.cash/technology/paramgen/.

Institutional Repository - Library & Information Centre - University of Thessaly
13/02/2023 10:25:48 EET - 137.108.70.14