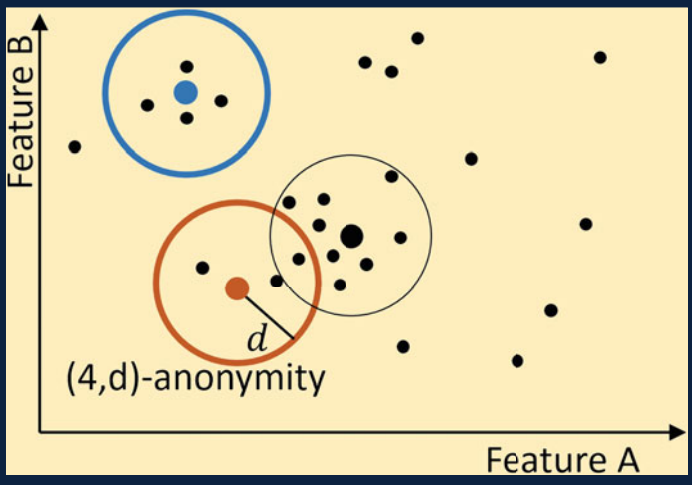
Tutorial

LNCS 9808

Alessandro Aldini
Javier Lopez
Fabio Martinelli (Eds.)

Foundations of Security Analysis and Design VIII

FOSAD 2014/2015/2016 Tutorial Lectures



Springer

Lecture Notes in Computer Science 9808

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

More information about this series at http://www.springer.com/series/7410

Alessandro Aldini · Javier Lopez
Fabio Martinelli (Eds.)

Foundations of Security Analysis and Design VIII

FOSAD 2014/2015/2016 Tutorial Lectures

Springer

Editors
Alessandro Aldini
University of Urbino
Urbino
Italy

Fabio Martinelli
National Research Council C.N.R.
Pisa
Italy

Javier Lopez
University of Malaga
Malaga
Spain

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-319-43004-1 ISBN 978-3-319-43005-8 (eBook)
DOI 10.1007/978-3-319-43005-8

Library of Congress Control Number: 2016945140

LNCS Sublibrary: SL4 – Security and Cryptology

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG Switzerland

Preface

The International Summer School on Foundations of Security Analysis and Design (FOSAD) has promoted the publication of books in the LNCS series that collect a selection of tutorials presented at FOSAD. We are very proud to present the eighth volume in this series, which includes contributions from three editions of FOSAD from 2014 to 2016. The history of FOSAD goes back to 2000, when it was established as a high education cradle for young researchers in the field of security for computer systems and networks. The overall number of participants since the first edition is now more than 750, and many of them have become well-known and appreciated researchers and FOSAD lecturers. Analogously, thanks to the quality and high standard of the lectures, the FOSAD book series represents a clear and comprehensive reference for graduate students and young researchers from academia and industry.

The first two contributions accompany presentations given at FOSAD 2014. The former is presented by Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, and Jens Groth from University College London. In the setting of proof systems for cryptographic protocols verification, the authors provide an overview of techniques behind the construction of zero-knowledge proofs. The latter is a work by Steven Van Acker and Andrei Sabelfeld from Chalmers University of Technology, who discuss the security of Web applications executing JavaScript code and the sandboxing systems used to restrict and control JavaScript functionalities. A contribution from FOSAD 2015 is authored by Michael Backes, Pascal Berrang, and Praveen Manoharan from Saarland University. They developed a user-centric privacy framework for quantitatively assessing the exposure of personal information in open environments. The proposed methodology is instantiated in the setting of identity disclosure and validated in a large-scale real-world case study. The last contribution, selected from FOSAD 2016, is by Ankur Taly and Asim Shankar, researchers at Google Inc. They define a fully decentralized authorization model for large and open distributed systems. Such a model is deployed as part of an open-source application framework called Vanadium.

We are grateful to the organizations and institutions that have supported FOSAD in the last few years, among which we would like to mention the IFIP Working Groups 1.7 on Theoretical Foundations of Security Analysis and Design and 11.14 on Secure Engineering. We also thank the EU FP7 project Confidential and Compliant Clouds (CoCoCloud), the EU H2020 project European Network for Cyber Security (NeCS), and the EPSRC CryptoForma network. We finally wish to thank the staff of the University Residential Centre of Bertinoro for the organizational and administrative support.

June 2016

Alessandro Aldini
Javier Lopez
Fabio Martinelli

Contents

Efficient Zero-Knowledge Proof Systems

Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, and Jens Groth(✉)

University College London, London, UK
j.groth@ucl.ac.uk

Abstract. A proof system can be used by a prover to demonstrate to one or more verifiers that a statement is true. Proof systems can be interactive where the prover and verifier exchange many messages, or non-interactive where the prover sends a single convincing proof to the verifier. Proof systems are widely used in cryptographic protocols to verify that a party is following a protocol correctly and is not cheating.

A particular type of proof systems are zero-knowledge proof systems, where the prover convinces the verifier that the statement is true but does not leak any other information. Zero-knowledge proofs are useful when the prover has private data that should not be leaked but needs to demonstrate a certain fact about this data. The prover may for instance want to show it is following a protocol correctly but not want to reveal its own input.

In these lecture notes we give an overview of some central techniques behind the construction of efficient zero-knowledge proofs.

1 Introduction

Imagine a company is trying to assess a candidate for a highly specialized position. A simple solution would be for them to present her with a task of their choice and rate her performance. The candidate declines, as the assessment might have her doing useful work without compensation. She proposes the choice of the task is left to her, to ensure the company does not unfairly profit from this process. The company is not convinced; the task may be too easy, or selected to hide the candidate's weaknesses.

For the assessment to go forward we need a special set of tasks: on the one hand they must be hard enough such that only qualified candidates are able to accomplish them, on the other hand they should not give away anything else since candidates do not want to function as unpaid workers. In job interviews this often takes the form of logic puzzles.

In cryptographic protocols, we do not have jobs and candidates; but we often have situations where we want to demonstrate some property holds or a statement is true without giving away any other information. Here zero-knowledge proofs are appropriate tools.

Zero-knowledge proof systems, introduced by Goldwasser et al. [GMR85], take place between two parties called prover and verifier. The prover wants to convince the verifier a certain statement is true, but without the verifier gaining

A. Aldini et al. (Eds.): FOSAD VIII, LNCS 9808, pp. 1–31, 2016.
DOI: 10.1007/978-3-319-43005-8_1

any other knowledge during the exchange (e.g. *why* the statement is true). Thus, there are three core requirements in zero-knowledge proofs:

Completeness. For true statements, a prover can convince the verifier.
Soundness. For false statements, a prover cannot convince the verifier (even if the prover cheats and deviates from the protocol).
Zero-Knowledge. The verifier will not learn anything from the interaction apart from the fact that the statement is true.

The statements we will be concerned with here are of the form $u \in L_R$, where L_R is an NP-language defined by a polynomial time decidable binary relation R. For $(u, w) \in R$, we say u is the statement and w is a witness for $u \in L_R$. The prover knows the witness w, and wants to convince the verifier that $u \in L_R$ without revealing anything else. In particular, the prover does not want to reveal the witness w.

1.1 Motivation

Here we will describe a few applications of zero-knowledge protocols. We do not aim to be exhaustive, but rather to provide some context in terms of applications.

e-Voting. Let's consider a simple voting setting: individual voters cast their votes and the electoral authorities produce the tally. To keep their votes private, the voters encrypt their votes. There are encryption schemes with a homomorphic property that allows the addition of the votes in encrypted form. If ballots consist of encrypted 0 s and 1 s (signifying "no" and "yes"), then the authorities can use the homomorphic property to produce an encrypted sum of all the votes. The authorities can then decrypt the ciphertext with the sum of the votes to get the election result, the number of "yes" votes, without the need to decrypt any of the individual ballots.

Unfortunately, this solution is *too* simple. What is to stop a voter from cheating by encrypting a 2 instead of the prescribed 0 or 1? In effect, the voter would be voting twice. We want to prevent voters from deviating from the voting protocol, but at the same time we want their votes to remain private. So, we want to verify that their ballots are valid, i.e., encrypt 0 or 1, but at the same time the voters do not want to reveal which vote they are casting, i.e., whether the plaintext is a 0 or a 1. This can be accomplished by using zero-knowledge proofs between the voters and the electoral authorities. Each voter acts as a prover that demonstrates her encrypted vote is valid. Completeness means that the ballots of honest voters are accepted. Soundness ensures that invalid ballots are rejected. And zero-knowledge keeps the votes secret.

Mix-Nets. Mix-nets [Cha88] are a tool for anonymous messaging given a broadcast channel. Instead of directly addressing messages to their recipients, a sender might opt to use a Mix server as an intermediary. The sender encrypts the message and recipient with the server's public key and addresses the message to the

server. Once the server has collected a number of messages, it decrypts all of them and broadcasts the plaintexts in a randomized order. We can expand on this construction by using multiple servers in sequence and threshold decryption: each server removes part of the encryption and randomly reorders the list of ciphertexts. The advantage of this construction is that no single server can determine which input corresponds to which output.

However, this procedure gives dishonest servers too much freedom. In particular, they might opt to drop messages and replace them with their own. Worse, even if the replacement is noticed, it will be hard to attribute it to a single server. Again zero-knowledge proofs come to the rescue. A solution is to ask each server to prove it acted honestly; that is to demonstrate that there exists a reordering such that the server's outputs is a partial decryption (consistent with the server's private key) of the server's inputs. Obviously, this proof should not reveal the concrete reordering or the server's private key, which is why it should be zero-knowledge.

Playing Nicely. In general, we can use zero-knowledge protocols to ensure that parties are following the prescribed protocol for any particular operation. This is a powerful tool since it prevents active attacks where somebody tries to cheat by deviating from the protocol.

1.2 Example: A Zero-Knowledge Proof for Graph Isomorphism

In this section we will use a simple zero-knowledge proof for graph isomorphism [GMW91] as an example to illustrate a common protocol flow. We can think of an undirected graph as a set of vertices V and a set of edges E between the vertices. Two graphs $G_0 = (V, E_0)$ and $G_1 = (V, E_1)$ are isomorphic if there is a permutation of the vertices and edges mapping one graph to the other, see Fig. 1 for an example. More precisely, we say a permutation $f : V \to V$ is an isomorphism from G_0 to G_1 if for all pairs of vertices $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$. Graph isomorphisms are transitive, if we have two graph isomorphisms $f : G_0 \to G_1$ and $g : G_1 \to G_2$ then $g \circ f : G_0 \to G_2$ is a graph isomorphism between G_0 and G_2.

Given graphs G_0, G_1 it is easy to check whether a permutation f of the vertices is a graph isomorphism. On the other hand, there is currently no known polynomial time algorithm that given two graphs G_0 and G_1 can determine whether they are isomorphic or not. We will in the following consider the situation where the statement consists of a claim that two graphs are isomorphic to each other. The prover knows an isomorphism between the graphs, but wants to convince the verifier that they are isomorphic without revealing the isomorphism. More precisely, we consider the language of pairs of isomorphic graphs $L_R = \{(G_0, G_1)\}$ defined by the relation $R = \{((G_0, G_1), f) : G_1 = f(G_0)\}$.

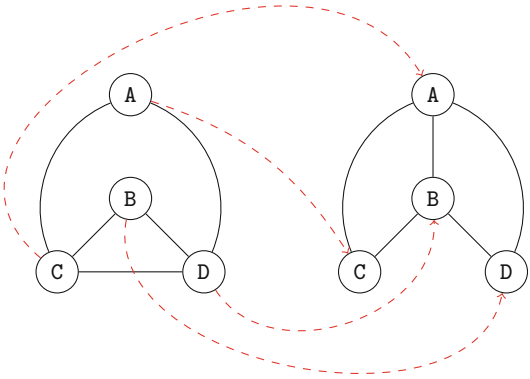


Fig. 1. Two isomorphic graphs: Reordering ABCD to CDAB maps the first graph to the second.

Statement: A pair of graphs G_0, G_1 on the same set of vertices V.
Prover's witness: An isomorphism f between G_0 and G_1.
Protocol:

- The prover picks a random permutation $h : V \to V$ and computes $H = h(G_1)$. She stores h and sends H to the verifier.
- The verifier picks a random challenge $b \leftarrow \{0, 1\}$.
- If $b = 0$ the prover sends $g = h \circ f$ to the verifier.
  If $b = 1$ the prover sends $g = h$ to the verifier.
- The verifier accepts the proof if $g(G_b) = H$.

It is simple to see that our protocol is complete. If G_0 and G_1 are isomorphic to each other, then both of them are isomorphic to H. Furthermore, the prover who knows the isomorphism f can easily compute both the isomorphism between G_0 and H and the isomorphism between G_1 and H. So she can answer both of the possible challenges $b \in \{0, 1\}$ and has 100 % probability of convincing the verifier.

Our protocol is sound in the sense that there is 50 % chance of catching a cheating prover. If G_0 and G_1 are not isomorphic then H cannot be isomorphic to both. So, if the verifier picks b such that G_b is not isomorphic to H, then the prover cannot answer the challenge.

To increase our chance of catching a cheating prover, we can repeat the challenge and response protocol. We modify the protocol to perform n repetitions for the same G_0, G_1 but different H_i, b_i and g_i. In each interaction, we have 50 % chance of catching the cheating prover, so overall the risk of cheating is reduced to 2^{-n}.

In the soundness discussion above, we considered a cheating prover using non-isomorphic G_0, G_1. But what about the case where G_0 and G_1 are isomorphic but the prover might or might not know f? Soundness provides no guarantees: it ensures that a witness w exists, but not that the prover knows it. The graph isomorphism protocol gives a stronger guarantee, which we will

refer to as extractability. Suppose the prover after having sent H can answer both challenges $b = 0$ and $b = 1$, then it could actually compute an isomorphism $f = g_1^{-1} \circ g_0$ between G_0 and G_1. We will later define a zero-knowledge protocol to be *extractable* if it is possible to extract a witness from a successful prover, for instance by rewinding it and running it again on a new challenge.

Finally, there is the zero-knowledge property. This can be somewhat puzzling: what does it mean for a run of the protocol not to give the verifier any new information? We will define zero-knowledge through simulation. If the verifier could simulate the protocol transcript himself without interacting with the prover, then he cannot have learned anything new from seeing the transcript.

The graph isomorphism proof can be simulated by first picking a random permutation g, then guessing at random $b \leftarrow \{0, 1\}$, and finally setting $H = g(G_b)$. With 50 % probability the guess b matches what the verifier would send after seeing H, and in that case we have a simulated proof transcript (H, b, g). If our guess is wrong, we just rewind the verifier to the start and try again with a new random g and b until we guess the challenge correctly.

Let us argue that if G_0, G_1 are isomorphic then the transcripts produced by successful simulations are identical to those produced by real executions of the protocol. In both cases we can think of g as a uniformly random renumbering of the vertices of G_0 or G_1, which means that H is uniformly random. We also note that the distribution of b given H is unchanged. Therefore, we see that the probability distributions of real and simulated transcripts are identical. The important thing to notice about the simulation is that we do not use the witness at all to simulate. Therefore, the simulated transcript cannot leak any information about the witness. Since the real proofs have the same probability distribution as the simulated proofs this means they do not leak any information either.

1.3 Security and Performance Parameters

Zero-knowledge proofs come in many flavours depending on the application. The particular choice depends on the desired security properties and performance parameters. We will now discuss some of these options.

Security Properties. Completeness, soundness and zero-knowledge often come in one of three flavours: perfect, statistical and computational. Perfect completeness means that an honest prover will always convince an honest verifier on a true statement, perfect soundness means that it is impossible to prove a false statement, and perfect zero-knowledge means that transcripts can be perfectly simulated and leak no information whatsoever.

In the graph isomorphism example we have perfect completeness and perfect zero-knowledge, but not perfect soundness since a cheating prover has 50 % chance of convincing the verifier on a false statement. Even if we repeat the protocol n times, there is still 2^{-n} chance of cheating and we do not get perfect soundness. However, we get statistical soundness in the sense that there is negligible small probability of cheating the verifier.

Perfect soundness can be relaxed to statistical soundness, where we require a prover has negligible probability of cheating the verifier. We can relax it further to computational soundness, where we admit the possibility of cheating, but are content if it is computationally infeasible to find a way to cheat. We have computational soundness, when it is unlikely that a probabilistic polynomial time prover can cheat.

Perfect zero-knowledge can be relaxed to statistical zero-knowledge, where the simulated transcript just needs to have a probability distribution that is close to that of a real proof. It can be further relaxed to computational zero-knowledge, where a computationally bounded verifier cannot tell whether it is seeing a transcript of an interaction with a real prover or a simulation of its view of such an interaction.

Interaction. The graph isomorphism proof we described needs three messages to be exchanged between the two parties, starting with the prover. In general, we measure the interaction of a zero-knowledge proof in the number of messages or *moves* the parties makes. We will refer to two moves as a *round* consisting of one move from each side.

When expanding the graph isomorphism protocol to n repetitions, we can easily see that the number of moves becomes $2n + 1$ since we can combine the last message of iteration i with the first message of iteration $i + 1$ since both are sent from the prover. Another option would be to perform the multiple iterations in parallel to reduce interaction. However, parallel composition does not always yield the desired result. Parallel composition of zero-knowledge proofs does not necessarily result in a zero-knowledge proof [GK96], or soundness may be less than what we might expect [BIN97].

In general, we aim to restrict the number of rounds used by protocols, as it requires that participants are available and need to remember previous messages for an extended period. One particular class of zero-knowledge proofs are those consisting of a single move from the prover to the verifier. We call these proofs *non-interactive* and will return to them in Sect. 3.

Communication. We consider the communication cost of the protocol to be the total bit-length of all messages exchanged by the two parties. We often compare the communication to the size of the statement as an indication of relative efficiency.

In the graph isomorphism proof, the statement is two graphs of k vertices G_0, G_1, which we can represent with two adjacency matrices using less than k^2 bits since they are symmetric. The communication consists of the graph H (less than $\frac{1}{2}k^2$ bits), the reply b (1 bit) and a description of g (in the order of $k \log k$ bits). The communication cost is thus linear in the size of the statement.

Computation. We usually distinguish between the computation cost of the prover and that of the verifier. We often opt to make verification quicker at the expense of the prover. First, in some settings, such as voting, a non-interactive proof for a ballot being valid is only created once but may be seen and verified multiple

times. Second, in applications such as verifiable computation the verifier is much weaker than the prover and it is only natural to try and lessen the computational load of the verifier. Finally, one may argue that being computationally bounded is a core characteristic of the verifier. If the verifier was computationally unbounded, she could check whether a statement $u \in L$ directly. This would eliminate the need for a proof in many cases.

Security Setting. Most protocols do not exist in vacuum; their security is based on a number of assumptions. These assumptions may be computational in nature, where a certain mathematical problem is considered hard to solve. There are also zero-knowledge protocols making stronger assumptions on the underlying primitives, e.g., many zero-knowledge proofs rely on the random oracle model where a cryptographic hash-function is assumed to behave like a truly random function [BR93].

A potential resource but at the same time potential security liability is the environment in which the zero-knowledge proof is executed. Interactive zero-knowledge proofs can be executed without any setup but the availability of a common reference string, e.g., a bit-string with a certain probability distribution, may improve performance. For non-interactive zero-knowledge proofs it is necessary and unavoidable to have a common reference string or some other form of assistance.

1.4 Notation

In the next two sections, we will give an overview of main ideas in Σ-protocols, which yield efficient interactive zero-knowledge proofs, and non-interactive zero-knowledge proofs. It will be useful to establish some common notation.

We write $y = A(x; r)$ when an algorithm A on input x and randomness r, outputs y. We write $y \leftarrow A(x)$ for the process of picking randomness r at random and setting $y = A(x; r)$. We also write $y \leftarrow S$ for sampling y uniformly at random from a set S. We will for convenience assume uniform random sampling from various types of sets is possible; there are easy ways to amend our protocols to the case where the sets are only sampleable with a distribution that is statistically close to uniform.

We assume all algorithms and parties in a cryptographic protocol will directly or indirectly get a security parameter λ as input (which for technical reasons will be written in unary 1^λ to ensure the running time is polynomial). The intuition is that the higher the security parameter the more secure should the scheme be. We will define security in terms of experiments that define the execution of a scheme in the presence of the adversary, and predicates that define whether the adversary succeeded in breaking the scheme. We are interested in the probability that the adversary breaks the scheme, for which we use the notation

$$\Pr[\text{output} \leftarrow \text{Experiment}(1^\lambda) : \text{Predicate}(\text{output})].$$

We will use the notation $\mathcal{A}$ for the adversary and assume it is either unbounded or efficient, where we define an efficient adversary as one that runs in probabilistic polynomial time.

Given two probability functions in the security parameter $f, g : \mathbb{N} \to [0, 1]$ we say that they are *close* and write $f(\lambda) \approx g(\lambda)$ when $|f(\lambda) - g(\lambda)| = O(\lambda^{-c})$ for every constant $c > 0$. We say that f is *negligible* if $f(\lambda) \approx 0$, and we say that f is *overwhelming* if $f(\lambda) \approx 1$. We will in many security definitions want the adversary's success probability to be negligible in the security parameter.

2 Σ-Protocols

In the previous section we discussed an interactive proof system for graph isomorphism. In the example the verifier picks a random challenge in $\{0, 1\}$ and the prover has probability $\frac{1}{2}$ of convincing the verifier of a false statement. The protocol needs to be iterated many times in order to reduce this probability and achieve good soundness. In this section we describe 3-move interactive proof systems in which the verifier picks a uniformly random challenge from a much larger space. This means a cheating prover has small probability of guessing the verifier's challenge in advance. The size of the challenge space is made big enough so that a single execution of the protocol suffice to convince the verifier. This kind of interactive proof systems often goes under the name of Σ-protocols.

2.1 Definitions

Σ-protocols are 3-move interactive proof systems that allow a prover to convince a verifier about the validity of a statement. The prover sends an initial message a to the verifier, the verifier replies with a random challenge x, and the prover answers with a final response z. The verifier finally checks the transcript (a, x, z) and decides whether to accept or reject the statement.

A Σ-protocol is *public coin*, which means that the verifier picks the challenge x uniformly at random and independently of the message sent by the prover.

Definition 1 (Σ-protocol). *Let R be a polynomial time decidable binary relation and let L_R be the language of statements u for which there exists a witness w such that $(u, w) \in R$. A Σ-protocol for a relation R is a tuple $(\mathcal{P}, \mathcal{V})$ of probabilistic polynomial time interactive algorithms such that*

- *$a \leftarrow \mathcal{P}(u, w)$: given a statement u and a witness w such that $(u, w) \in R$, the prover computes initial message a and sends it to the verifier.*
- *$x \leftarrow S$: the verifier picks a uniformly random challenge x from a large set S and sends it to the prover.*
- *$z \leftarrow \mathcal{P}(x)$: given challenge x the prover computes a response z and sends it to the verifier.*
- *$1/0 \leftarrow \mathcal{V}(u, (a, x, z))$: the verifier checks the transcript (a, x, z) and returns 1 if she accepts the argument and 0 if she rejects it.*

A pair of efficient algorithms $(\mathcal{P}, \mathcal{V})$ is a Σ-protocol if is complete, special sound and special honest verifier zero-knowledge in the sense of the following definitions.

Completeness guarantees that if both prover and verifier are honest, then the verifier accepts when $u \in L_R$ and the prover knows the corresponding witness.

Definition 2 (Completeness). *$(\mathcal{P}, \mathcal{V})$ is computationally complete if for all probabilistic polynomial time adversaries $\mathcal{A}$*

$$\Pr\left[(u, w) \leftarrow \mathcal{A}(1^\lambda); a \leftarrow \mathcal{P}(u, w); x \leftarrow S; z \leftarrow \mathcal{P}(x) : \mathcal{V}(u, (a, x, z)) = 1\right] \approx 1,$$

where $\mathcal{A}$ outputs $(u, w) \in R$.

If this holds for unbounded adversaries $\mathcal{A}$, we say that $(\mathcal{P}, \mathcal{V})$ is statistically complete. If the probability above is also exactly equal to 1 we says that $(\mathcal{P}, \mathcal{V})$ is perfectly complete.

A Σ-protocol is a form of proof of knowledge, in the sense that a prover should be able to answer random challenges only if she *knows* a witness for a statement u. This is formalised via *special soundness* which says that given two accepting transcripts corresponding to two distinct challenges and the same initial message it is possible to extract a witness for the statement.

Definition 3 (Special Soundness). *$(\mathcal{P}, \mathcal{V})$ is computationally special sound if there exists an efficient extractor algorithm $\mathcal{E}$ such that for all probabilistic polynomial time adversaries $\mathcal{A}$*

$$\Pr\left[\begin{array}{l}(u, a, x, z, x', z') \leftarrow \mathcal{A}(1^\lambda); w \leftarrow \mathcal{E}(a, x, z, x', z') : \\ \mathcal{V}(u, (a, x, z)) = 0 \text{ or } \mathcal{V}(u, (a, x', z')) = 0 \text{ or } (u, w) \in R\end{array}\right] \approx 1.$$

If this holds for unbounded adversaries $\mathcal{A}$, we say that $(\mathcal{P}, \mathcal{V})$ is statistically special sound. If the probability above is also exactly equal to 1 we says that $(\mathcal{P}, \mathcal{V})$ is perfectly special sound.

A Σ-protocol is zero-knowledge if it does not leak information about the witness beyond the membership of u in the language L_R. The definition of zero-knowledge follows the simulation paradigm, which says that if it is possible to simulate an accepting transcript without knowing a witness, then the protocol is not leaking information about the witness. At first, this might seem in contradiction with the soundness requirement, which says that is should be hard to produce an accepting transcript without knowing a witness. However, the simulator is not taking part in the real execution of the protocol, and therefore we can assume it to be less restricted than the parties directly involved in the protocol. We can for example allow the simulator to produce messages forming the transcript in a different order than it happens during the real interaction. In case of special honest verifier zero-knowledge, we restrict the verifier to be a public coin verifier that picks random challenges independently from the messages she receives from the prover. In this setting the simulator is given the verifier's challenge x and has to simulate a conversation between prover and verifier without knowing a witness.

Definition 4 (Special Honest Verifier Zero-Knowledge). *A public coin argument $(\mathcal{P}, \mathcal{V})$ is computationally special honest verifier zero-knowledge (SHVZK) if there exists a probabilistic polynomial time simulator $\mathcal{S}$ such that for all probabilistic polynomial time stateful adversaries $\mathcal{A}$*

$$\begin{aligned}&\Pr\left[(u, w, x) \leftarrow \mathcal{A}(1^\lambda); a \leftarrow \mathcal{P}(u, w); z \leftarrow \mathcal{P}(x) : \mathcal{A}(a, x, z) = 1\right] \\ \approx &\Pr\left[(u, w, x) \leftarrow \mathcal{A}(1^\lambda); (a, z) \leftarrow \mathcal{S}(u, x) : \mathcal{A}(a, x, z) = 1\right]\end{aligned}$$

If this holds for unbounded adversaries $\mathcal{A}$, we say that $(\mathcal{P}, \mathcal{V})$ is statistically special honest verifier zero-knowledge. If the probabilities above are also exactly equal we says that $(\mathcal{P}, \mathcal{V})$ is perfectly special honest verifier zero-knowledge.

The above definition of zero-knowledge might not be strong enough for many applications since it is assuming a semi-honest verifier that does not deviate from the protocol. However, there are efficient transformations [DGOW95, Dam00, GMY06] for SHVZK Σ-protocols to obtain full zero-knowledge against malicious verifiers with a small overhead in communication and computation.

2.2 Σ-Protocol for the Equivalence of Discrete Logarithm

Consider two group elements $s, t \in \mathbb{G}$, such that they share the same discrete logarithm with respect to two different generators $g, h \in \mathbb{G}$. We now give a simple Σ-protocol for the equality of discrete logarithms of s and t. More precisely we describe a Σ-protocol for the following relation

$$R = \{(u, w) | u = (\mathbb{G}, p, g, h, s, t); g, h, s, t \in \mathbb{G}; s = g^w; t = h^w\}$$

where $\mathbb{G}$ is a group of prime order p with $|p| = \lambda$.

The prover starts by picking a random field element r from $\mathbb{Z}_p$ and then computes two *blinding* elements $a = g^r, b = h^r$ and sends them to the verifier. The verifier picks a uniformly random challenge $x \leftarrow \mathbb{Z}_p$ and sends it back to the prover. The prover computes the field element $z = wx + r$ and sends it to the verifier. The verifier checks if both of the following verification equations hold

$$g^z = s^x a \qquad\qquad h^z = t^x b$$

in which case accept the proof and otherwise rejects it. The argument is summarised in Fig. 2.

The idea behind the protocol is that if the prover knows the discrete logarithm of s and t, then she can compute the discrete logarithm of s^x and t^x with respect to base g and h. Moreover, if a and b have the same discrete logarithm, then so will $s^x a$ and $t^x b$.

The protocol above is clearly complete. If both the prover and the verifier are honest, the verifier will always accept statements in the language.

For special soundness consider two accepting transcripts (a, b, x, z) and (a, b, x', z') for distinct challenges x, x' and the same initial message (a, b). Dividing the verification equation $g^z = s^x a$ by $g^{z'} = s^{x'} a$ we obtain $g^{z-z'} = s^{x-x'}$.

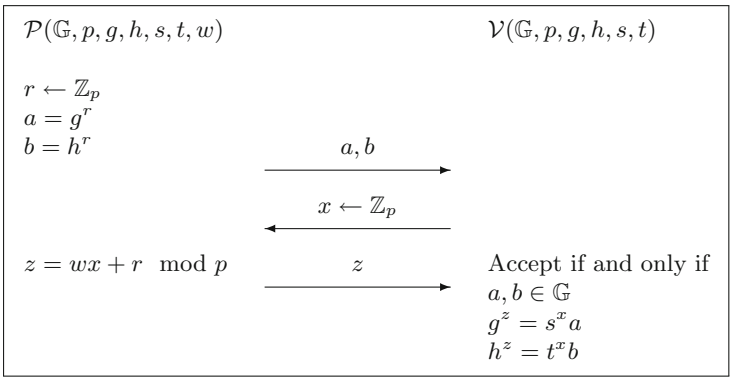


Fig. 2. Σ-protocol for equivalence of discrete logarithm.

Therefore we have that the discrete logarithm of s with respect to g is $w = \frac{z-z'}{x-x'} \mod p$. Similar calculations on the other verification equations tells us that the discrete logarithm of t with respect to base h is w too.

For SHVZK we need to show a simulator that given a uniformly random challenge x as input can produce a transcript indistinguishable from a real transcript. The simulator can pick a uniformly random field element z and compute the first message as $a = g^z s^{-x}$ and $b = h^z t^{-x}$. Note that x and z have the same distribution as in the real execution of the protocol and that a, b are uniquely determined given x and z. Therefore the transcript (a, b, x, z) output by the simulator has the same distribution as an honestly generated transcript.

2.3 Commitment Schemes

Commitment schemes are key primitives for the construction of many cryptographic protocols. They allow a sender to create a commitment to a secret value. The sender may later decide to open the commitment and reveal the value in a verifiable manner. We require two main properties to commitment schemes:

- *Hiding:* a commitment should not reveal the secret value it contains.
- *Binding:* the sender should not be able to open the commitment to a different value.

Non-interactive commitments are a particularly useful type of commitment scheme, for which both committing and verifying the opening of a commitment can be done locally, without any interaction with other parties.

Formally, a non-interactive commitment scheme is a pair of probabilistic polynomial time algorithms $(\mathrm{Gen}, \mathrm{Com})$. The setup algorithm $ck \leftarrow \mathrm{Gen}(1^\lambda)$ generates a commitment key ck. The commitment key specifies a message space $\mathcal{M}_{ck}$, a randomness space R_{ck} and a commitment space $\mathcal{C}_{ck}$. The commitment algorithm combined with the commitment key specifies a function $\mathrm{Com}_{ck} : \mathcal{M}_{ck} \times R_{ck} \to \mathcal{C}_{ck}$. Given a message $m \in \mathcal{M}_{ck}$ the sender picks uniformly at random $r \leftarrow R_{ck}$ and computes the commitment $c = \mathrm{Com}_{ck}(m; r)$.

Definition 5 (Hiding). *A non-interactive commitment scheme* $(\mathrm{Gen}, \mathrm{Com})$ *is computationally hiding if for all probabilistic polynomial time stateful interactive adversaries* $\mathcal{A}$

$$\Pr\left[\begin{array}{l} ck \leftarrow \mathrm{Gen}(1^\lambda); (m_0, m_1) \leftarrow \mathcal{A}(ck); b \leftarrow \{0,1\}; \\ r \leftarrow R_{ck}; c \leftarrow \mathrm{Com}_{ck}(m_b; r) : \mathcal{A}(c) = b \end{array}\right] \approx \frac{1}{2}$$

where $\mathcal{A}$ *outputs* $m_0, m_1 \in \mathcal{M}_{ck}$*. If this holds for unbounded adversaries* $\mathcal{A}$*, we say that* $(\mathrm{Gen}, \mathrm{Com})$ *is unconditionally hiding. If the probability above is also exactly equal to* $\frac{1}{2}$*, we says that* $(\mathrm{Gen}, \mathrm{Com})$ *is perfectly hiding.*

Definition 6 (Binding). *A non-interactive commitment scheme* $(\mathrm{Gen}, \mathrm{Com})$ *is computationally binding if for all probabilistic polynomial time adversaries* $\mathcal{A}$

$$\Pr\left[\begin{array}{l} ck \leftarrow \mathrm{Gen}(1^\lambda); (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(ck) : \\ \mathrm{Com}_{ck}(m_0; r_0) = \mathrm{Com}_{ck}(m_1; r_1) \,\mathrm{and} m_0 \neq m_1 \end{array}\right] \approx 0$$

where $\mathcal{A}$ *outputs* $m_0, m_1 \in \mathcal{M}_{ck}$ *and* $r_0, r_1 \in R_{ck}$*. If this holds for unbounded adversaries* $\mathcal{A}$*, we say that* $(\mathrm{Gen}, \mathrm{Com})$ *is unconditionally binding. If the probability above is also exactly equal to 0, we says that* $(\mathrm{Gen}, \mathrm{Com})$ *is perfectly binding.*

Many examples of commitment schemes have been proposed in the literature. Two well-known examples are Pedersen [Ped91] and Elgamal [EG85] commitments, which are based on the discrete logarithm assumption. In addition to the above properties, both commitment schemes are also additively homomorphic, which means that multiplying two commitments produces a commitment to the sum of the openings. More precisely, we say a commitment scheme is additively homomorphic if for all valid keys ck the message, randomness and commitment spaces are abelian groups and for all messages $m_0, m_1 \in \mathcal{M}_{ck}$ and randomness $r_0, r_1 \in R_{ck}$ we have

$$\mathrm{Com}_{ck}(m_0; r_0) \cdot \mathrm{Com}_{ck}(m_1; r_1) = \mathrm{Com}_{ck}(m_0 + m_1; r_0 + r_1).$$

Pedersen Commitments. Consider a group $\mathbb{G}$ of prime order p and let g, h be random generators of the group. Message and randomnesses are in $\mathbb{Z}_p$ and the commitment space is the group $\mathbb{G}$. The sender commits to an element $m \in \mathbb{Z}_p$ by picking a uniformly random r from $\mathbb{Z}_p$ and computing $c = g^m h^r$. The scheme is perfectly hiding and computationally binding, assuming that the discrete logarithm assumption holds (Fig. 3).

ElGamal Commitments. The commitment key, the message space and the randomness space are defined as for Pedersen commitments. The commitment space is $\mathbb{G} \times \mathbb{G}$. Commitments are generated by picking a random $r \leftarrow \mathbb{Z}_p$ and computing $(g^r, g^m h^r)$. The ElGamal commitment scheme is perfectly binding and computationally hiding given that the decision Diffie-Hellman assumption holds (Fig. 4).

$\mathrm{Gen}(1^\lambda) \to ck$	$\mathrm{Com}_{ck}(m) \to c$
$\cdot\ p \leftarrow \{0,1\}^\lambda$ s.t. p is prime	$\cdot$ If $m \notin \mathbb{Z}_p \to \perp$
$\cdot\ \mathbb{G}$ of order p	$\cdot\ r \leftarrow \mathbb{Z}_q$
$\cdot\ h \leftarrow \mathbb{G}$ s.t. $\langle h \rangle = \mathbb{G}$	$\cdot\ c := g^m h^r$
$\cdot\ g = h^x$ for $x \leftarrow \mathbb{Z}_p$	
$\cdot\ ck := (\mathbb{G}, p, g, h)$	

Fig. 3. Pedersen commitment.

$\mathrm{Gen}(1^\lambda) \to ck$	$\mathrm{Com}_{ck}(m) \to c$
$\cdot\ p \leftarrow \{0,1\}^\lambda$ s.t. p is prime	$\cdot$ If $m \notin \mathbb{Z}_p \to \perp$
$\cdot\ \mathbb{G}$ of order p	$\cdot\ r \leftarrow \mathbb{Z}_q$
$\cdot\ h \leftarrow \mathbb{G}$ s.t. $\langle h \rangle = \mathbb{G}$	$\cdot\ c := (g^r, g^m h^r)$
$\cdot\ g = h^x$ for $x \leftarrow \mathbb{Z}_p$	
$\cdot\ ck := (\mathbb{G}, p, g, h)$	

Fig. 4. ElGamal commitment.

After seeing the above examples one might wish to build a commitment scheme that achieves both hiding and binding properties unconditionally. Unfortunately, this is not achievable. The reason is that an unbounded adversary $\mathcal{A}$ is always able to compute an opening to a commitment. If the scheme is such that there exists only one possible opening, then the scheme cannot be hiding. On the other hand, if there are several distinct openings to a commitment then an unbounded adversary can compute all of them and break the binding property.

2.4 Two Useful Examples of Σ-Protocols

Commitment schemes and Σ-protocols are closely related. It is possible in fact to construct commitment schemes out of Σ-protocols for hard relations as described in [Dam90]. It is also convenient to rethink the interaction of Σ-protocols in terms of committing and opening. A general way to build Σ-protocols is to let the prover commit to some values in the first move, and to open some commitments depending on the challenge in the last move.

We try to illustrate this general approach by showing two examples of Σ-protocols. The first one is a protocol for showing that a commitment opens to 0. The second protocol is for proving that a commitment opens to the product of the openings of two other commitments.

Σ_{zero}. Consider a commitment A opening to 0 to be part of the statement. The prover computes a random commitment $B = \mathrm{Com}_{ck}(0; s)$ and sends it to the verifier, which answer with a random challenge x. The prover then sends opening information z to the verifier, which checks the commitment $A^x B$ opens to 0 using randomness z. The full description of the protocol is in Fig. 5.