Efficient Zero-Knowledge Proof Systems

Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth

University College London

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 want to show it is following a protocol correctly but does 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, Micali and Rackoff [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 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 0s and 1s (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 verticies 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 $R = \{((G_0, G_1), f) : G_1 = f(G_0)\}$.

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 $b \leftarrow \{0, 1\}$.

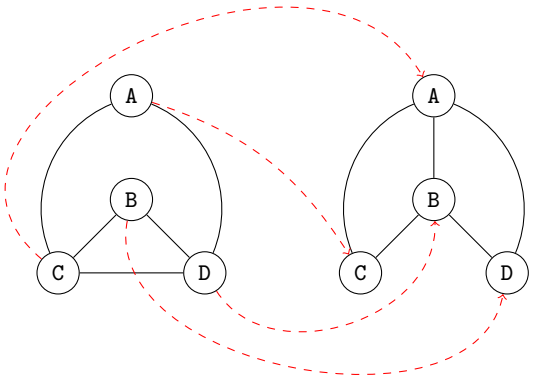

Fig. 1: Two isomorphic graphs: Reordering ABCD to CDAB maps the first graph to the second.

- 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 succesful 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-knowlegde 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 Section 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 vaildity 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 verifer.*
- $x \leftarrow S$: *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 verifer 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

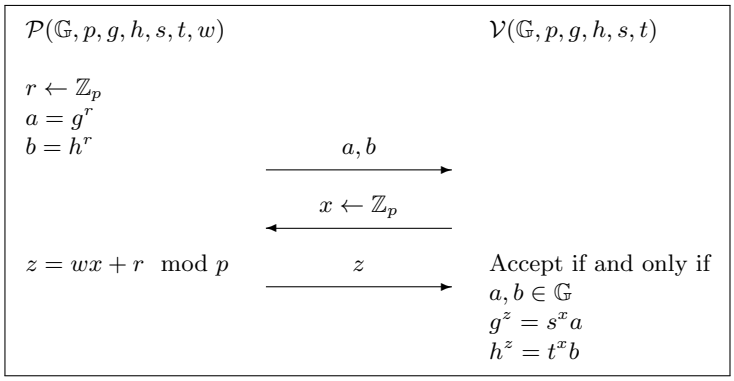

Fig. 2: Σ-protocol for equivalence of discrete logarithm.

the verification equation $g^z = s^x a$ by $g^{z'} = s^{x'} a$ we obtain $g^{z-z'} = s^{x-x'}$. 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* $\mathcal{A}$

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

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.

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

$\text{Gen}(1^\lambda) \to ck$	$\text{Com}_{ck}(m) \to c$
$\cdot\ p \leftarrow \{0,1\}^\lambda$ 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.

$\text{Gen}(1^\lambda) \to ck$	$\text{Com}_{ck}(m) \to c$
$\cdot\ p \leftarrow \{0,1\}^\lambda$ 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.

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.

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 $B = \text{Com}_{ck}(0; s)$ 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.

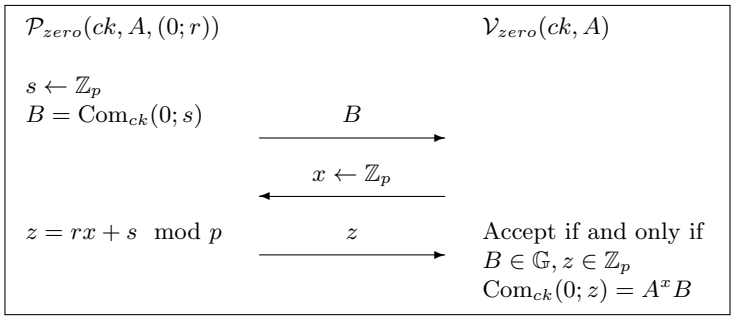

Fig. 5: Σ-protocol for opening a commitment to 0.

This protocol could be used also to prove equality of openings of commitments. Given two commitments A_1 and A_2 it suffices to use Σ_{zero} to show that $A_1 A_2^{-1}$ opens to zero. In the protocol we only require the commitment scheme to be homomorphic, therefore it can be instantiated with both Pedersen and ElGamal commitments. In both cases we get perfect completeness, perfect soundness and perfect special honest verifier zero-knowledge.

Σ_{prod}. For this protocol we focus on the case of Pedersen commitments and refer to [CD98] for the more general case. Let A, B, C be commitments opening to a, b and ab, respectively. Consider a commitment key $ck = (\mathbb{G}, p, g, h)$. The main idea is for the prover to prove knowledge of opening of A and B and showing that C opens to the same value of A when replacing g with B in the commitment key. Let $ck' = (\mathbb{G}, p, B, h)$ be the modified key, thus

$$C = \mathrm{Com}_{ck}(ab; r_c) = g^{ab} h^{r_c} = B^a h^{r_c - a r_b} = \mathrm{Com}_{ck'}(a; r_c - a r_b)$$

The full description of the protocol is in Fig. 6. The protocol, Σ_{prod} achieves perfect completeness, perfect SHVZK and computational special soundness.

2.5 Composition of Σ-protocols

One of the characteristics that makes Σ-protocols very appealing is that it is easy to combine several of them together to obtain a Σ-protocol for compound relations. This allows a very modular design of complex Σ-protocols starting from simple building blocks.

For example, Σ-protocols are closed under parallel composition, therefore we can combine many Σ-protocols together using a unique verifier's challenge to prove that many statements hold simultaneously. Completeness, special soundness and SHVZK of the combined protocol are directly implied by the respective properties of the singular protocols. In particular for special soundness and SHVZK, we can define an extractor and a simulator respectively running in

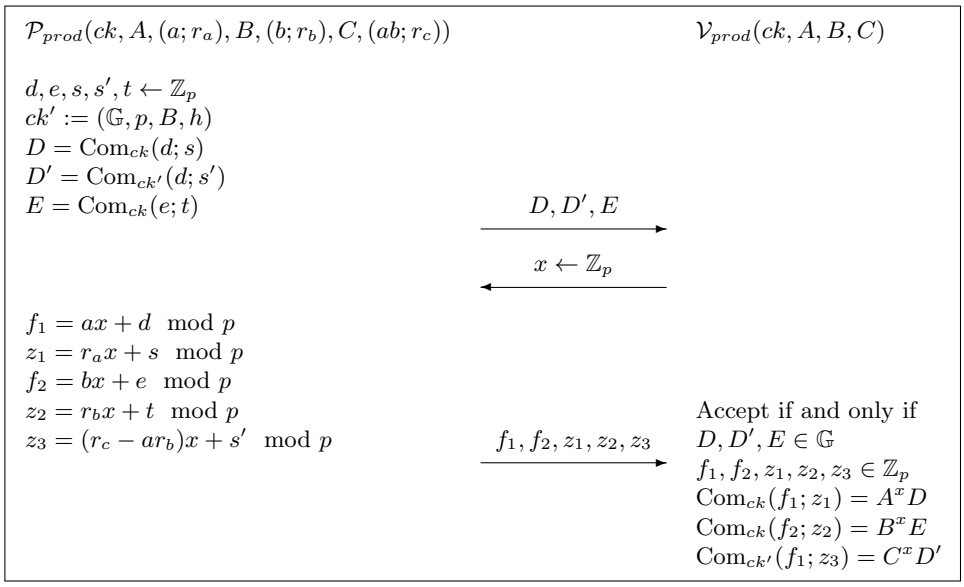

Fig. 6: Σ-protocol for the product of openings of Pedersen commitments.

parallel the extractors and simulators of the constituent Σ-protocols on the same challenge.

Given two protocols Σ_0 and Σ_1 for relations R_0 and R_1, it is possible to combine them to show that one statement out of u_0, u_1 holds, without disclosing which one. This transformation was first introduced in [CDS94] and could be easily extended to prove the validity of one statement out of many. The idea is to allow prover $\mathcal{P}_{OR}$ to *simulate* the transcript for at most one of the two statements. Without loss in generality, consider a prover knowing a witness w_0 for u_0. Then the prover can pick a random challenge x_1 and simulate an accepting transcript (a_1, x_1, z_1) for u_1 by invoking simulator $\mathcal{S}_1$ for Σ_1. In the first move the prover sends to the verifier a_0 generated as in Σ_0 and simulated a_1. Upon receiving a challenge x form the verifier, the prover computes $x_0 = x \oplus x_1$. The prover computes responses z_0 using challenge x_0 and sends z_0, z_1, x_0, x_1 to the verifier. Then, the verifier checks that $x = x_0 \oplus x_1$ and accepts if both transcripts (a_0, x_0, z_0) and (a_1, x_1, z_1) are accepting. As above, completeness special soundness and SHVZK of the combined protocol are directly implied by the respective properties of the individual protocols. In particular for SHVZK, the simulator $\mathcal{S}_{OR}$ receives a challenge x as input, picks a random x_1 and computes x_0 such that $x = x_0 \oplus x_1$. Then, $\mathcal{S}_{OR}$ invokes $\mathcal{S}_0$ and $\mathcal{S}_1$ respectively on x_0 and x_1. The simulated transcript has the same distribution as a real transcript.

2.6 Arithmetic Circuits

To illustrate the capabilities of Σ-protocols, we show how to build a protocol for a more general relation combining several simpler protocols. For example using many parallel executions of the zero and product Σ-protocols in Section 2.4 we can provide Σ-protocols for the satisfiability of arithmetic circuits. To prove satisfiability of an arithmetic circuit the prover has to commit to all the w_i corresponding to wire assignments and then prove consistency of inputs and outputs of addition gates using Σ_{zero} and multiplication gates using Σ_{prod}.

Consider for instance a very simple arithmetic circuit over $\mathbb{Z}_p$ consisting of fan-in-2 addition and multiplication gates, as the one pictured in Fig. 7. The prover computes commitments $W_i = \mathrm{Com}_{ck}(w_i, r_i)$ for random r_i and then shows that both commitments $W_1 \cdot W_2 \cdot W_3^{-1}$ and $W_4 \cdot W_5 \cdot W_7^{-1}$ open to 0 and that W_3 and W_8 open to $w_1 \cdot w_2$ and $w_6 \cdot w_7$, respectively.

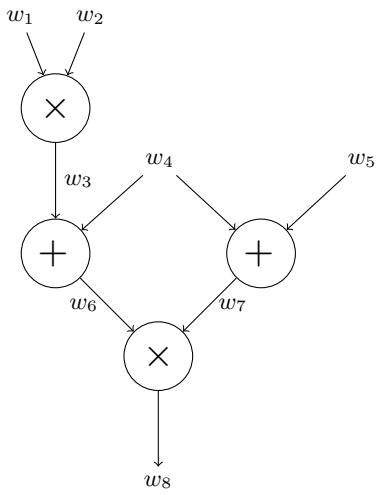

Fig. 7: Example of an arithmetic circuit.

For an arithmetic circuits with N addition and multiplication gates we need to combine N parallel executions of Σ_{zero} and Σ_{prod}. The resulting communication amounts to $O(N)$ commitments and field elements.

2.7 Batching

When proving the same relation many times, there are more efficient solutions than executing many Σ-protocols in parallel. As a simple example, if we have several commitments $A_1, \ldots, A_n$ and want to prove all of them contain zero, we can use the protocol in Fig. 8. The underlying idea is to use the homomorphic property to build a committed degree n polynomial in the challenge x, with

the committed values as coefficients. This committed polynomial has negligible probability of evaluating to 0 in the random challenge x unless it is the zero polynomial, i.e., all the committed values are 0. Using this protocol we only communicate a constant number of elements to prove a statement of size n elements is true.

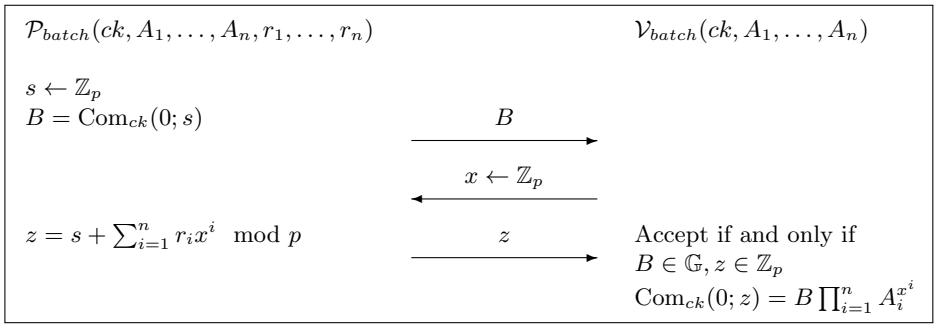

Fig. 8: Batch Σ-protocol for opening of many commitments to 0.

Another way to batch arguments together is to commit to many values at once. We can build commitments to vectors rather than single elements and extend the previous techniques to vector commitments. We can for instance extend Pedersen commitments to allow openings in $\mathbb{Z}_p^n$, as described in Fig. 9. This extension preserves the same properties of the standard Pedersen commitment scheme but committing to n elements only requires sending a single group element.

$\mathrm{Gen}(1^\lambda) \to ck$	$\mathrm{Com}_{ck}(m) \to c$
$\cdot\ p \leftarrow \{0,1\}^\lambda$ s.t. p is prime	$\cdot$ If $m \notin \mathbb{Z}_p^n \to \perp$
$\cdot\ \mathbb{G}$ of order p	$\cdot\ m = (m_1, \ldots, m_n)$
$\cdot\ h \leftarrow \mathbb{G}$ s.t. $\langle h \rangle = \mathbb{G}$	$\cdot\ r \leftarrow \mathbb{Z}_q$
$\cdot\ (x_1, \ldots, x_n) \leftarrow \mathbb{Z}_p^n$	$\cdot\ c := h^r \prod_{i=1}^{n} g_i^{m_i}$
$\cdot\ g_i = h^{x_i}$ for $i \in [n]$	
$\cdot\ ck := (\mathbb{G}, p, h, g_1, \ldots, g_n)$	

Fig. 9: Pedersen commitment for vectors of length n.

Groth [Gro09] used batching techniques and vector commitments together to give zero-knowledge arguments for linear algebra relations over vectors. These techniques make it possible to give arguments for the satisfiability of arithmetic circuits with an overall communication of $O(\sqrt{N})$ group and field elements. So arithmetic circuit satisfiability and many other relevant relations can be proved with sublinear communication.

3 Non-interactive zero-knowledge proofs

In interactive zero knowledge proofs, the prover and the verifier interact over multiple rounds, and can vary their responses depending on the messages that they have received so far. By contrast, non-interactive zero knowledge proofs consist of a single message sent by the prover to the verifier. Non-interactive proofs are typically more difficult to construct than interactive proofs, and often rely on stronger assumptions. However, they are useful in settings where interaction cannot or should not take place, such as digital signatures and encryption schemes.

Non-interactive zero-knowledge proofs were introduced by Blum, Feldman and Micali [BFM88], who produced a proof for the 3-colourability of graphs under a number-theoretic assumption.

3.1 Formal Definitions

A non-interactive proof system for a relation R consists of three probabilistic polynomial time algorithms. There are the common reference string generator Gen, the prover $\mathcal{P}$, and the verifier $\mathcal{V}$. The common reference string generator takes the security parameter as input and produces a common refrence string σ. The prover takes (σ, x, w) as input and produces a proof π. The verifier takes (σ, x, π) as input and outputs 1 if accepting the proof as valid, and 0 if rejecting the proof.

We call $(\text{Gen}, \mathcal{P}, \mathcal{V})$ a non-interactive proof system for R if it has the completeness and perfect soundness properties to be defined below. If $(\text{Gen}, \mathcal{P}, \mathcal{V})$ has completeness and computational soundness, we call it a non-interactive argument system.

Completeness. As with interactive proofs, completeness states that a prover should be able to prove a true statement.

Definition 7 (Completeness). *We say the proof system is perfectly complete if for all $\lambda \in \mathbb{N}$ and all $(u, w) \in R$*

$$\Pr\left[\sigma \leftarrow \text{Gen}(1^\lambda); \pi \leftarrow \mathcal{P}(\sigma, u, w) \ : \ \mathcal{V}(\sigma, u, \pi) = 1\right] = 1.$$

For statistical completeness, the definition is changed so that the probability is close 1*. For computational completeness, we restrict to probabilistic polynomial time adversaries $\mathcal{A}$, and change the definition so that the probability close to* 1*.*

Soundness. Soundness states that a cheating prover should not be able to prove a false statement; even when deviating from the protocol.

Definition 8 (Soundness). *We say the proof system has (adaptive) perfect soundness if for all $\lambda \in \mathbb{N}$ and all adversaries $\mathcal{A}$*

$$\Pr\left[\sigma \leftarrow \text{Gen}(1^\lambda); (u, \pi) \leftarrow \mathcal{A}(\sigma, u) \ : u \notin L_R \ \text{ and } \ \mathcal{V}(\sigma, u, \pi) = 1\right] = 0.$$

The definition can be relaxed to statistical soundness by changing the definition such that the probability is close from 1 *instead of requiring exact equality. For computational soundness, we restrict to probabilistic polynomial time adversaries* $\mathcal{A}$*, and change the definition so that the probability is close* 1*.*

A weaker definition is that of *non-adaptive* soundness. Here the adversary $\mathcal{A}$ is given a false statement $u \notin L_R$ independently of the common reference string σ. Adaptively sound proofs are harder to construct, but are more versatile since in many cases the false statement could be chosen after seeing the common reference string. Adaptively sound proofs also have the advantage that the same common reference string can be reused to prove different statements u from the same language.

Proof of Knowledge. A non-interactive proof system is a proof of knowledge if it is possible to recover the witness w from the proof. More formally, we say that a non-interactive proof system is a proof of knowledge if there exists a probabilistic polynomial time knowledge extractor $E = (E_1, E_2)$ such that E_1 produces a correctly generated common reference string with extraction key ξ, which E_2 uses to extract a valid witness from a proof.

Definition 9 (Knowledge Extraction). *We say the proof system has perfect knowledge extraction if for all* $\lambda \in \mathbb{N}$*, and all adversaries* $\mathcal{A}$

$$\Pr\left[\sigma \leftarrow \mathrm{Gen}(1^\lambda) \ : \ \mathcal{A}(\sigma) = 1\right] \ = \ \Pr\left[(\sigma, \xi) \leftarrow \mathcal{E}_1(1^\lambda) \ : \ \mathcal{A}(\sigma) = 1\right]$$

and

$$\Pr\left[(\sigma, \xi) \leftarrow E_1(1^\lambda); (u, \pi) \leftarrow \mathcal{A}(\sigma); w \leftarrow E_2(\sigma, \xi, u, \pi) \ : \ (u, w) \in R \text{ if } V(\sigma, u, \pi) = 1\right] = 1.$$

For statistical knowledge extraction, the definition is changed so that the first two probabilities are close to each other, and the third is close to 1. For the computational version, we restrict to probabilistic polynomial time adversaries $\mathcal{A}$*.*

Perfect knowledge extraction implies perfect soundness. This is because if a valid proof π is given for statement u, we can extract a witness w with $(u, w) \in R$, so in particular, $x \in L_R$.

Zero-Knowledge. The zero-knowledge property ensures that a non-interactive proof reveals nothing except for the truth of the statement being proved. As with interactive zero-knowledge proofs, this is achieved using the simulation paradigm. There must be an efficient simulator for the proof, so that any information computed from a real proof could also be computed from a simulated proof. There is no witness available when simulating the proof, so no information about the witness is leaked. However, the simulator must have access to more than just the statement u and a common reference string σ when simulating a proof. Otherwise, anybody would be able to create a convincing proof, even without the witness! To this end, the simulator is allowed to produce the common reference string for itself, along with some extra information τ, the 'simulation trapdoor'. This trapdoor is used by the simulator, but is unavailable to an adversary against the protocol.

Definition 10 (Zero-Knowledge). *We call* $(\mathrm{Gen}, \mathcal{P}, \mathcal{V})$ *an NIZK proof for* R *with perfect zero-knowledge if there exists a simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that for all* $\lambda \in \mathbb{N}$*, and all adversaries* $\mathcal{A}$

$$\begin{aligned} &\Pr\left[\sigma \leftarrow \mathrm{Gen}(1^\lambda); (u, w) \leftarrow \mathcal{A}(\sigma); \pi \leftarrow \mathcal{P}(\sigma, u, w) \ : \ \mathcal{A}(\sigma, \pi) = 0\right] \\ = &\Pr\left[(\sigma, \tau) \leftarrow \mathcal{S}_1(1^\lambda); (u, w) \leftarrow \mathcal{A}(\sigma); \pi \leftarrow \mathcal{S}_2(\sigma, \tau, u) \ : \ \mathcal{A}(\sigma, \pi) = 0\right] \end{aligned}.$$

For statistical zero-knowledge, the definition is changed so that the probabilities are close to each other. For computational zero-knowledge, we restrict to probabilistic polynomial time adversaries $\mathcal{A}$*, and change the definition so that the probabilities are close to each other.*

3.2 The Common Reference String

The definitions of an NIZK proof system require a common reference string to be available to the prover and verifier. It would be desirable to try and remove this requirement and obtain a proof system where the prover sends a single message to the verifier, with no setup. Unfortunately, it can be shown that any such proof system can only be used for languages that are easy to decide [Gol01] so the verifier does not need the prover to be convinced that $u \in L_R$. In order to construct NIZK proof systems for non-trivial languages, a common reference string or some other type of assistance is necessary.

A common reference string can be made up of uniformly random bits. In this case, it is often referred to as a common random string. However, in many NIZK proof systems, a more structured common reference string is generated according to a different probability distribution.

The common reference string can be honestly generated by a trusted party. Another solution is to use the multi-string model of Groth and Ostrovsky [GO14], where random strings are produced by several authorities, and a majority of strings are assumed to be honestly generated. This removes the need to completely trust any single party. Secure multi-party computation can also be used to ensure that the common reference string is generated correctly.

3.3 Public and Private Verifiability

In the original definitions, the verification algorithm takes σ, u and π as input. This means that the proof can be verified by anybody. One variation is a designated-verifier proof system. In this case, the setup algorithm Gen outputs a verifier key ω as well as the common reference string, and the verification algorithm takes ω, u and π as input. Now, proofs can only be privately verified. Public verifiability corresponds to the special case where $\omega = \sigma$.

Designated-verifier non-interactive proof systems are generally easier to construct, and can be more efficient than publicly verifiable proofs. This is because unlike publicly verifiable proofs, the verifier has ω, which is not available to the prover. Designated verifier proofs can only be used to convince somebody

in posession of ω. This is in contrast with publicly-verifiable proofs, where a single proof can be copied and sent to other recipients, and suffices to convince everybody.

Private verifiability is sufficient for some applications, such as CCA-secure encryption schemes including the Cramer-Shoup cryptosystem. However, public-verifiability is necessary for many other applications such as signatures, and universally-verifiable voting systems.

3.4 The Fiat-Shamir Heuristic

The Fiat-Shamir heuristic is a method for converting public coin interactive zero-knowledge arguments into NIZK proofs. The first step is to include the description of a cryptographic hash function H in the common reference string. The prover computes their messages as they would in the interactive proof, but replaces the verifier's messages with a hash of the protocol transcript up to that point.

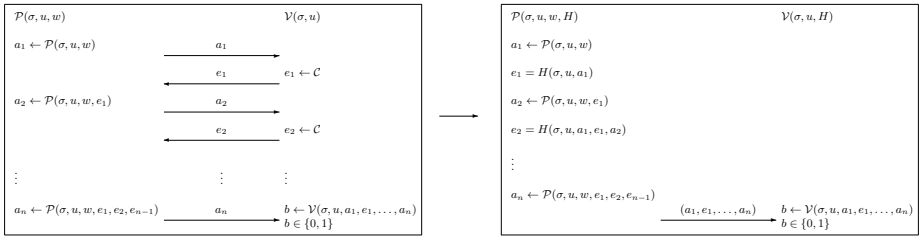

Fig. 10: The Fiat-Shamir Heuristic

The method yields highly efficient NIZK arguments in practice. By modelling the hash function as a truly random function, or 'random oracle', it is possible argue that the resulting arguments are sound [BR93]. Further, in the random oracle model, even if the initial interactive proof only has honest verifier zero-knowledge, the resulting argument will have full zero-knowledge.

However, in reality, hash functions are deterministic. It has been shown [CGH00, **?**] that there are interactive protocols which have soundness when H is modelled as a truly random function, but which are insecure for *any* choice of hash function H. Despite this theoretical problem, the Fiat-Shamir heuristic is still used to produce arguments for practical applications, where the hope is that it does give sound arguments for "natural" problems.

3.5 The Hidden Bits Model

In the hidden bits model, described in [FLS99], the prover uses the common reference string in a particular way that produces some secret bits known only

to the prover. She can then choose to reveal individual bits to the verifier in a verifiable manner.

One way the hidden bits model can be implemented as follows [FLS99, BY96, Gro10a]: The prover chooses a public key for an encryption scheme. She then interprets the common reference string as a sequence of ciphertexts. Since only she knows the secret decryption key, only she knows the corresponding plaintexts. If the encryption scheme allows revealing a plaintext in a verifiable manner, she can now selectively disclose some plaintexts and let other plaintexts remain secret.

A structured hidden bit-string is often more useful than a uniformly random string. In order to create a structured hidden bit-string the prover may discard or reveal certain bits in order to obtain, with good probability, a string with a particular structure. As a simple example, if the prover orders the bits in pairs and reveals the bits in all pairs of the form '00' or '11', then the verifier knows that statistically speaking the remaining hidden bits are structured such that almost all pairs are '01' or '10'.

One technique to use a structured hidden bit string [KP98] is to group bits in pairs, and reveal at most one bit from each pair. We refer to '11' and '00' as 'R' for random, and '10' and '01' as 'W' for wildcard. From 'W', the prover can choose whether to reveal a '0' or a '1'. From an 'R' pair, the prover has no choice in what to reveal since both bits are the same. Using this, and more sophisticated structures, it is possible to set up equations the revealed bits should satisfy. When the statement is true, the structure allows the prover to reveal bits such that all equations are satisfied. When the statement is false, no choice of bits to reveal will satisfy all equations. We will now give a simple example to provide some intuition how this can work.

Example. Consider the formulae

$$x_1 \vee x_2 = 1, \qquad (\neg x_1) \vee x_3 = 1, \qquad (\neg x_2) \vee (\neg x_3) = 1$$

This is a very simple instance of the satisfiability problem and will form the statement for an NIZK proof. Note that each variable and its negation appear exactly once. We will consider blocks of four bits, and assume that every block has exactly one 'W'; we only use the blocks 'WR' and 'RW'. A proof of satisfiability for these formulae will require one block for each variable, 3 blocks in this case.

The idea behind the proof is as follows. For each variable x_i, exactly one of the literals x_i or $\neg x_i$ is equal to 1. This will be assigned to the 'W' part of the block, while the other will be assigned the 'R' part of the block. This means that the prover can choose whether to reveal a '0' or '1' for the true value, which will be the key to completeness, but has no choice about which bit to reveal for the false value, which will help to enforce soundness.

The assignment $x_1 = 1, x_2 = 0, x_3 = 1$ is a solution to the satisfiability problem. This is the witness for the proof.

Suppose for example that the prover receives hidden bits in blocks 'WR','WR','RW'. Variables are assigned to blocks as follows.

W	R	W	R	R	W
x_1	$\neg x_1$	$\neg x_2$	x_2	$\neg x_3$	x_3

Now, the prover reveals one bit from each pair. For the false values, the prover is forced to reveal a particular bit, and randomly chooses between the first and the second bit from the pair. For the true values, the prover chooses whether to reveal a '0' or a '1' in order to make each formulae from the statement have odd parity. In this example, the prover could reveal bits as follows.

W	R	W	R	R	W
x_1	$\neg x_1$	$\neg x_2$	x_2	$\neg x_3$	x_3
1?	?1	1?	0?	?1	0?

The verifier now checks that the formulae are satisfied when the revealed bits are substituted for each variable. The prover can always reveal bits consistent with the formulae, because at least one 'W' has been assigned to each formula.

The proof does not give away any information about the witness, because the pairs such as '1?' that the verifier sees can originate from several different bit-strings and possible witnesses.

In this example, the statement was true. But let us to argue soundness consider what would happen for an unsatisfiable set of formulae. Then no matter which assignment the prover chooses, there is at least one formulae where all the variables have been assigned an 'R' pair. Now for each pair in this formulae the prover has only one choice of bit to reveal and there is 50% chance of the revealed bits having even parity.

In the example shown above, it is very easy to work out a witness. For more complicated satisfiability problems it is possible to design a similar proof where, if there is no solution, then a large fraction of the verifier's checks will fail [KP98, Gro10a]. By increasing the size of blocks, and imposing further conditions, the prover can only succeed with negligible probability in the case where the statement is an unsatisfiable set of formulae.

3.6 Boneh-Goh-Nissim Encryption

Many public-key cryptosystems have homomorphic properties, but only recenty have we seen the emergence of encryption schemes that are homomorphic with respect to both addition and multiplication. Partial progress was made by Boneh, Goh and Nissim [BGN05], who designed a public-key encryption scheme based on pairings, which are bilinear maps arising from the study of algebraic geometry and elliptic curves. This cryptosystem allows arbitrary additions on the encrypted plaintexts, but only allows a single multiplication.

The BGN encryption scheme uses an group with an element g of order n, where n is a product of two primes p and q, and an efficiently computable pairing. Let $\mathbb{G}$ be the group of order n generated by g. Let h be a random, non-trivial element of order p. The message space is chosen to be small so that it is easy to take discrete logarithms and find the message m, given the element g^{pm}.

Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be the bilinear pairing map.

The public key of the scheme is $\mathbb{G}, n, g, h$, and the secret key is p.

A message m is encrypted with randomness $r \leftarrow \mathbb{Z}_n$ as the ciphertext $c = \mathsf{Enc}(m; r) = g^m h^r$.

Decryption is performed by raising the ciphertext to the power of the secret key, p. Since h has order p, this removes the randomness from the ciphertext. We have $c^p = g^{mp} h^{rp} = g^{pm}$. Then, since the message space is small, it is easy to take discrete logarithms and recover m.

It is easy to see that the scheme is homomorphic with respect to addition:

$$\mathsf{Enc}(m_1; r_1)\mathsf{Enc}(m_2; r_2) = (g^{m_1} h^{r_1}) \cdot (g^{m_2} h^{r_2}) = g^{m_1+m_2} h^{r_1+r_2} = \mathsf{Enc}(m_1+m_2; r_1+r_2)$$

To do a multiplication of encrypted plaintexts, we apply the bilinear map to two ciphertexts. Let $e(g, g) = G$, which has order n, and let $e(h, h) = H$, which has order p. The elements $e(g, h)$ and $e(h, g)$ turn out to be powers of H by properties of the pairing map. This means that

$$e\left(g^{m_1} h^{r_1}, g^{m_2} h^{r_2}\right) = e(g, g)^{m_1 m_2} e(g, h)^{m_1 r_2} e(h, g)^{r_1 m_2} e(h, h)^{r_1 r_2} = G^{m_1 m_2} H^{r'},$$

which is an encryption of $m_1 m_2$ in $\mathbb{G}_T$ with some randomness r'.

3.7 NIZK Proof for Circuit Satisfiability

By using the homomorphic properties of the BGN cryptosystem, we can obtain a zero-knowledge proof of satisfiability for a Boolean circuit. Let $\mathcal{C}$ be a boolean circuit, with $\mathcal{W}$ wires.

The prover begins by encrypting the values of the wires for the circuit to produce $|\mathcal{W}|$ BGN ciphertexts. To show that the encrypted wire values satisfy the circuit, each wire value must be either 0 or 1, and the output of each NAND gate must be correct with respect to the two inputs. Figure 11 provides an example.

Let $c = g^m h^r$ be a ciphertext. Then m is a bit if and only if $m(m-1) = 0$. We get an encryption of $m - 1$ by computing cg^{-1}, and then an encryption of $m(m-1)$ by computing $e(c, cg^{-1})$, which should be an encryption of 0. The prover sends $\pi = (g^{2m-1} h^r)^r \in \mathbb{G}$ as part of the proof. The verifier then checks that $e(c, cg^{-1}) = e(\pi, h)$. As shown in [GOS12], this demonstrates that $e(c, cg^{-1})$ has order p, which means that it must be an encryption of 0. There are $|\mathcal{W}|$ extra group elements which must be added to the proof.

Finally, the prover must demonstrate that the wire values are correct with respect to each NAND gate. For values $m_1, m_2, m_3 \in \{0, 1\}$, we have that $m_1 = \neg(m_2 \wedge m_3)$ if and only if $2m_1 + m_2 + m_3 - 2 \in \{0, 1\}$. The prover uses the homomorphic property of the commitment scheme again to form an encryption of $2m_1 + m_2 + m_3 - 2$. Then, the prover uses the same technique as for the wire values to show that this value is a bit. This adds another $|\mathcal{C}|$ group elements to the proof.

The total communication of the proof is therefore $2|\mathcal{W}| + |\mathcal{C}|$ group elements. The proof has perfect completeness and perfect soundness. It has computational zero-knowledge relying on the semantic security of the BGN encryption scheme.

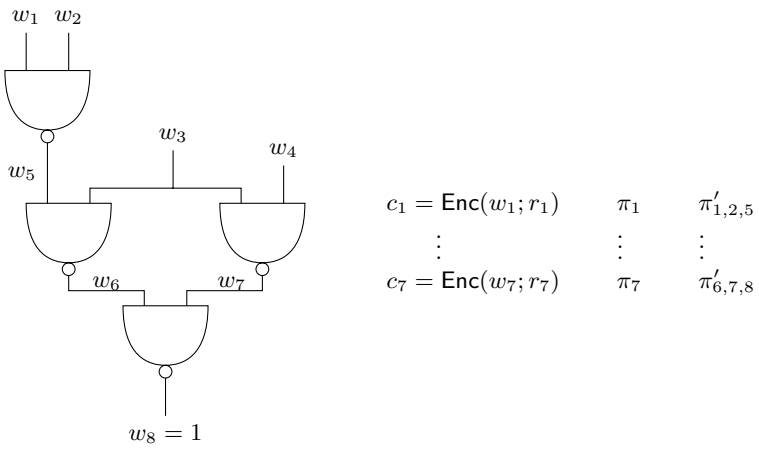

Fig. 11: A simple Boolean circuit. To demonstrate satisfiability using the BGN cryptosystem, the prover encrypts each w_i, and produces an additional group element to prove that they are bits. A further group element per gate used to prove that the wire values respect the logic gates.

Boolean circuit satisfiability is an NP-complete language. Therefore, this example shows that every language in NP has a non-interactive zero-knowledge proof.

3.8 Succinct Non-interactive Arguments and Applications

Having seen a few examples of techniques for constructing NIZK proofs, it is natural to ask how efficient they can be. Micali [Mic00] introduced the notion of computationally sound proofs, which can be much smaller than the statement. Motivated by applications such as verifiable computation, there has recently been a lot of research on reducing the size of the proofs and making the verification process very efficient.

A succinct non-interactive argument (SNARG) is a non-interactive argument system that satisfies an additional succinctness property. SNARGs which are also arguments of knowledge are referred to as SNARKs.

Definition 11 (Succinctness). *A non-interactive argument system is succinct if all proofs π satisfy*

$$|\pi| = \mathrm{poly}(\lambda)\mathrm{polylog}\left(|u| + |w|\right)$$

The development of SNARGs has culminated in pairing-based constructions [Gro10b, Lip12, BCCT12, PHGR13, BCCT13, GGPR13, BSCTV14] which use only a constant number of group elements and are extremely efficient to verify. All arguments rely on very strong assumptions, but there is some evidence that this may be unavoidable [GW11].

Verifiable Computation. The notion of verifiable computation was formalised by Gennaro, Gentry and Parno [**?**]. In a verifiable computation scheme, a client would like to outsource some computation to a worker who has more computing power. However, the client would like some assurance that the worker performed the computation correctly and delivered the correct result. The assurance takes the form of a non-interactive proof produced by the worker to convince the client. For this application, it is essential that proofs should have extremely small size and that the computational cost of verifying a proof should be low.

Pinocchio [PHGR13] is a practical implementation where one can produce schemes to convince the client that a C code was outsourced correctly. Pinocchio takes a program written in C, computes conditions for correct running in the quadratic arithmetic program model of [GGPR13], and outputs a SNARK which can then be used to verify computation. The system has been tested in applications such as image matching, gas simulations, and computing SHA-1 hashes to provide benchmarking data.

Proof-Carrying Data. Proof-carrying data [**?**] is an approach to multi-party construction of proofs that allows the construction of very efficient schemes. Suppose that during a distributed computation, each party would like to be convinced that all previous steps in the computation were performed correctly. One way to do this would be to append to each message a non-interactive proof that the computation was performed correctly so far, and forward on non-interactive proofs for correct computation at previous steps. However, this naive solution adds to the communication of distributed computation at each step. A more sophisticated idea is for each party to include a proof of two facts. Firstly, the party should have performed their step in the computation correctly. Secondly, the party should have seen a valid proof that all previous steps in the computation were correct. For this purpose, we can use SNARKs.

The message size does not increase after each step in the computation as a result of the succinctness property of the SNARKs. Futhermore, SNARKs have knowledge soundness, which means that in principle, knowledge-extractors for the SNARK can be applied repeatedly to the final messages of the distributed computation, to recover the entire history of the computation. This property is essential for the security of the computation.

3.9 Efficiency

We compare the efficiency of some different NIZK proofs and arguments for circuit satisfiability.

NIZK Proofs. By definition, NIZK proofs have perfect or statistical soundness, which means that they are secure against even a computationally unbounded prover. Figure 1 shows the efficiency of some NIZK proofs for circuit satisfiability.

Let λ be the security parameter. Define $k_T = \text{poly}(\lambda)$ to be the size of a trapdoor permutation, $k_G \approx k^3$ to be the size of a suitable group element, and

$|C| = \text{poly}(\lambda)$ to be the size of the circuit. Let $|w| \leq |C|$ be the size of the witness for satisfiability of the circuit.

	CRS in bits	Proof in bits	Assumption
[GOS12]	$O(k_G)$	$O\left(\|C\|k_G\right)$	Pairing-based
[Gro10a]	$\|C\|k_T \text{poly}\log(\lambda)$	$\|C\|k_T \text{poly}\log(\lambda)$	Trapdoor Permutations
[Gro10a]	$\|C\|\text{poly}\log(\lambda)$	$\|C\|\text{poly}\log(\lambda)$	Naccache-Stern
[GGI$^+$14]	$\text{poly}(\lambda)$	$\|w\| + \text{poly}(\lambda)$	FHE and NIZK

Table 1: Performance comparison of NIZK Proofs for Circuit SAT

All the essential details of the proof from [GOS12] were presented earlier in Section 3.7. This proof uses the BGN cryptosystem. The proofs from [Gro10a] using techniques from the hidden bits model, implemented using either trapdoor permutations or the Naccache-Stern cryptosystem.

The proof of [GGI$^+$14] assumes the existence of a fully homomorphic encryption scheme. Fully homomorphic encryption allows for the multiplication and addition of ciphertexts to produce encrypted multiplications and additions of the plaintexts within. At a high level, the idea of this proof is to simply encrypt the witness, and evaluate the circuit on the witness in encrypted form. The prover then gives an NIZK proof that the resulting ciphertext contains a 1.

NIZK Arguments. By definition, NIZK arguments have computational soundness, which means that they are secure assuming a computationally bounded prover. Figure 2 shows the efficiency of some NIZK arguments for circuit satisfiability.

	CRS in group elements	Argument in group elements
[Gro10b]	$O\left(\|C\|^2\right)$	$O(1)$
[Lip12]	$O\left(\|C\|^{1+o(1)}\right)$	$O(1)$
[GGPR13]	$O\left(\|C\| \log \|C\|\right)$	$O(1)$
[**?**]		

Table 2: Performance comparison of NIZK Arguments for Circuit SAT

All of the arguments in the table are SNARKs that rely on strong assumptions and bilinear pairings. In each case, the size of the argument is a constant number of elements in a suitable bilinear group.

References

BCCT12. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. *Proceedings of the 3rd . . .*, 2012.

BCCT13. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing - STOC '13*, (4):111, 2013.

BFM88. Manuel Blum, Paul Feldman, and Silvio Micali. Non-Interactive Zero Knowledge and Its Applications (Extended Abstract) MIT. pages 103–112, 1988.

BGN05. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341, 2005.

BIN97. Mihir Bellare, Russell Impagliazzo, and Moni Naor. Does parallel repetition lower the error in computationally sound protocols? In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 374–383. IEEE, 1997.

BR93. Mihir Bellare and P Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. *Proceedings of the 1st ACM conference on ...*, (November 1993):1–21, 1993.

BSCTV14. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and M Virza. Scalable zero knowledge via cycles of elliptic curves. *Advances in Cryptology– ...*, pages 1–47, 2014.

BY96. Mihir Bellare and Moti Yung. Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation. *J. Cryptology*, 9(3):149–166, 1996.

CD98. Ronald Cramer and Ivan Damgård. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In *Advances in Cryptology - CRYPTO '98*, pages 424–441, 1998.

CDS94. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology - CRYPTO '94*, pages 174–187, 1994.

CGH00. Ran Canetti, Oded Goldreich, and Shai Halevi. The Random Oracle Methodology, Revisited. page 31, 2000.

Cha88. David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

Dam90. Ivan Damgård. On the existence of bit commitment schemes and zero-knowledge proofs. In *Advances in Cryptology ó CRYPTOí 89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 17–27. 1990.

Dam00. Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, pages 418–430, 2000.

DGOW95. Ivan Damgård, Oded Goldreich, Tatsuaki Okamoto, and Avi Wigderson. Honest verifier vs dishonest verifier in public coin zero-knowledge proofs. In *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, pages 325–338, 1995.

EG85. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, 1985.

FLS99. Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999.

GGI+14. Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam Smith. Using Fully Homomorphic Hybrid Encryption to Minimize Non-interative Zero-Knowledge Proofs. *Journal of Cryptology*, pages 1–22, 2014.

GGPR13. R Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic Span Programs and Succinct NIZKs without PCPs. *EUROCRYPT*, 2013.

GK96. Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.

GMR85. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 291–304. ACM, 1985.

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. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.

GMY06. Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *J. Cryptology*, 19(2):169–209, 2006.

GO14. Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. *J. Cryptology*, 27(3):506–543, 2014.

Gol01. Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

GOS12. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM (JACM)*, (0430254):0–32, 2012.

Gro09. Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In *Advances in Cryptology - CRYPTO 2009*, pages 192–208, 2009.

Gro10a. Jens Groth. Short non-interactive zero-knowledge proofs. *Advances in Cryptology-ASIACRYPT 2010*, 2010.

Gro10b. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. *Advances in Cryptology-ASIACRYPT 2010*, 2010.

GW11. Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. *Proceedings of the forty-third annual ACM . . .*, 2011.

KP98. Joe Kilian and Erez Petrank. An Efficient Noninteractive Zero-Knowledge Proof System for NP with General Assumptions. *Journal of Cryptology*, 11:1–27, 1998.

Lip12. Helger Lipmaa. Progression-free sets and sublinear pairing-based noninteractive zero-knowledge arguments. *Theory of Cryptography*, 2012.

Mic00. Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

Ped91. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 129–140, 1991.

PHGR13. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly Practical Verifiable Computation. *2013 IEEE Symposium on Security and Privacy*, pages 238–252, may 2013.