Contents

Chapter 1

Introduction

1.1 Security Amplification Theorems

Introduction to security amplification theorems and hardness implication statements. Example of classical results. Problems captured by weakly verifiable puzzles. Contribution of this thesis.

1.2 Weakly verifiable puzzles

1.3 Contribution of the Thesis

1.4 Organization of the Thesis

Overview of the content of the succeeding chapters.

Chapter 2

Preliminaries

In this chapter we set up notation and terminology used in the Thesis.

2.1 Notation and Definitions

(Functions) We write $poly(\alpha_1, \ldots, \alpha_n)$ to denote a polynomial on variables $\alpha_1, \ldots, \alpha_n$. We call a function $f : \mathbb{N} \to \mathbb{R}$ *negligible* if for every polynomial $poly(n)$ there exists $n_0 \in \mathbb{N}$ such that for all $n \in \mathbb{N} : n > n_0$ the following holds

$$f(n) < \frac{1}{poly(n)}.$$

On the other hand, we say that a function $f : \mathbb{N} \to \mathbb{R}$ is *non-negligible* if there exists a polynomial $poly(n)$ such that for some $n_0 \in \mathbb{N}$ and for all $n \in \mathbb{N} : n > n_0$ we have

$$f(n) \geq \frac{1}{poly(n)}.$$

We note that it is possible that a function $f : \mathbb{N} \to \mathbb{R}$ is neither negligible nor non-negligible. We say that a function f is *efficiently computable* if there exists a polynomial time algorithm computing f.

(Algorithms, Bitstrings and Circuits) We denote Boolean circuits using capital letters from the Greek or English alphabet. We define a *probabilistic circuit* as a Boolean circuit $C_{m,n} : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^*$. We write $C_{m,n}(x; r)$ to denote a probabilistic circuit taking as input $x \in \{0,1\}^m$ and as auxiliary input $r \in \{0,1\}^n$. If a probabilistic circuit does not take any input, we slightly abuse notation and write $C_n(r)$. Similarly, we use $\{C_n\}_{n \in \mathbb{N}}$ to denote a family of probabilistic circuits that takes only auxiliary input. We make sure that it is clear from the context that probabilistic circuits with only auxiliary input are not confused with deterministic Boolean circuits. For a (probabilistic) circuit C we write $\mathit{Size}(C)$ to denote the total number of vertices of C. A *(probabilistic) polynomial size circuit* is a (probabilistic) circuit of size polynomial in the number of input vertices (including auxiliary input vertices if the circuit is probabilistic). We define a *two phase circuit* $C := (C_1, C_2)$ as a circuit where in the first phase a circuit C_1 is used and in the second phase a circuit C_2.[1] If C_1 and C_2 are probabilistic circuits we write $C(\delta) := (C_1, C_2)(\delta)$ to denote that in both phases C_1 and C_2 take as auxiliary input the same bitstring δ.

For an algorithm A we write $\mathit{Time}(A)$ to denote the number of steps it takes to execute A. We say that A runs in *polynomial time* if $\mathit{Time}(A)$ is bounded by some $\mathit{poly}(|x|)$, where $|x|$ denotes the length of the input that A takes. Similarly, as for probabilistic circuits we often write randomness used by a probabilistic algorithm explicitly as a bitstring provided as an auxiliary input.

TODO: Time of oracle access

We denote a tuple $(x_1, \dots, x_l)$ by $x^{(l)}$. Furthermore, for tuples $x^{(l)}$, $y^{(k)}$ we use $x^{(l)} \circ y^{(k)}$ to denote the concatenation of $x^{(l)}$ and $y^{(k)}$ which results in a tuple $(x_1, \dots, x_l, y_1, \dots, y_k)$.

(Probabilities and distributions) For a finite set $\mathcal{R}$ we write $r \stackrel{\$}{\leftarrow} \mathcal{R}$ to denote that r is chosen from $\mathcal{R}$ uniformly at random. For $\delta \in \mathbb{R} : 0 \leq \delta \leq 1$ we write μ_δ to denote the Bernoulli distribution where outcome 1 occurs with probability δ and 0 with probability $1 - \delta$. Moreover, we use μ_δ^k to denote the probability distribution over k-tuples where each element of a k-tuple is drawn independently according to μ_δ. Finally, let $u \leftarrow \mu_\delta^k$ denote that a k-tuple u is chosen according to μ_δ^k.

Let $(\Omega_n, \mathcal{F}_n, \Pr)$ be a probability space and $n \in \mathbb{N}$. Furthermore, let $E_n \in \mathcal{F}_n$ denote an event that probability depends on n. We say that E_n happens *almost surely* or with *high probability* if $\Pr[E_n] \geq 1 - 2^{-n}\mathit{poly}(n)$.

(Interactive protocols) We are often interested in situations where two probabilistic circuits interact with each other according to some protocol. We limit ourselves to cases where circuits interact by means of messages representable by bitstrings. Let $\{A_n\}_{n\in\mathbb{N}}$ and $\{B_n\}_{n\in\mathbb{N}}$ be families of circuits such that $A_n : \{0,1\}^* \to \{0,1\}^*$ and $B_n : \{0,1\}^* \to \{0,1\}^*$. An *interactive protocol* is defined by $\{A_n\}_{n\in\mathbb{N}}$ and $\{B_n\}_{n\in\mathbb{N}}$ where for random bitstrings $\rho_A \in \{0,1\}^n$, $\rho_B \in \{0,1\}^n$ in the first round $m_0 := A_n(\rho_A)$ and in the second round $m_1 := B_n(\rho_B, m_1)$. In general in the $(2k-1)$-th round we have $m_{2k-2} := A_n(\rho_A, m_1, \cdots, m_{2k-3})$ and in the $2k$-th round $m_{2k-1} := B_n(\rho_B, m_1, \cdots, m_{2k-2})$. A protocol execution between two probabilistic circuits A and B is denoted by $\langle A, B\rangle$. The output of A in a protocol execution

[1] Where what a *phase* means should be clear from the context in which C is used.

is denoted by $\langle A, B\rangle_A$ and of B by $\langle A, B\rangle_B$. A sequence of all messages send by A and B in the protocol execution is called a communication transcript and is denoted by $\langle A, B\rangle_{trans}$.

TODO: Time complexity of the interactive protocol.

(Oracle algorithms) We use notion of *oracle circuits* following the standard definition included in the literature [Gol04]. If a circuit A gains oracle access to a circuit B, we write A^B. If additionally B gains oracle access to a circuit C we write A^{B^C}. However, to shorter notation, we often write A^B instead, and we make sure that it is clear from the context which oracle is accessed by B.

TODO: Exponential factor in what? **TODO:** Cite someone?

Definition 2.1 (Polynomial time sampleable distribution) *We say that a distribution is* polynomial time sampleable *if it can be approximated by an algorithm running in time poly*$(\log|\mathcal{D}|, \log|\mathcal{R}|)$ *up to an exponential factor.*

Definition 2.2 (Pairwise independent family of efficient hash functions) *Let $\mathcal{D}$ and $\mathcal{R}$ be finite sets and $\mathcal{H}$ be a family of functions mapping values from $\mathcal{D}$ to values in $\mathcal{R}$. We say that $\mathcal{H}$ is a* family of pairwise independent efficient hash functions *if $\mathcal{H}$ has the following properties.*

(Pairwise independent) *For $\forall x \neq y \in \mathcal{D}$ and $\forall \alpha, \beta \in \mathcal{R}$, it holds*

$$\Pr_{hash \leftarrow \mathcal{H}}[hash(x) = \alpha \mid hash(y) = \beta] = \frac{1}{|\mathcal{R}|}.$$

(Polynomial time sampleable) *For every $hash \in \mathcal{H}$ the function $hash$ is sampleable in time poly*$(\log|\mathcal{D}|, \log|\mathcal{R}|)$*.*

(Efficiently computable) *For every $hash \in \mathcal{H}$ there exists an algorithm running in time poly*$(\log|\mathcal{D}|, \log|\mathcal{R}|)$ *which on input $x \in \mathcal{D}$ outputs $y \in \mathcal{R}$ such that $y = hash(x)$.*

We note that the pairwise independence property is equivalent to

$$\Pr_{hash \leftarrow \mathcal{H}}[hash(x) = \alpha \wedge hash(y) = \beta] = \frac{1}{|\mathcal{R}|^2}.$$

It is well know [CW77] that there exists families of functions meeting the above criteria.

Chapter 3

Weakly Verifiable Cryptographic Primitives

This chapter gives an overview of dynamic interactive weakly verifiable puzzles. We start by formulating the definition of dynamic interactive weakly verifiable puzzles in Section 3.1. To familiarize the Reader with this notion and provide more intuition, in Section 3.2 we describe a series of well known cryptographic primitives that are captured by this notion. The Section 3.3 is devoted to the previous research concerning weakly verifiable puzzles.

3.1 Dynamic Interactive Weakly Verifiable Puzzles

We define *dynamic interactive weakly verifiable puzzle* as follows.

Definition 3.1 (Dynamic Interactive Weakly Verifiable Puzzle.) *A* dynamic interactive weakly verifiable puzzle (DIWVP) *is defined by a family of probabilistic circuits* $\{P_n\}_{n\in\mathbb{N}}$. *A circuit belonging to* $\{P_n\}_{n\in\mathbb{N}}$ *is called a problem poser. A solver* $C_n := (C_1, C_2)$ *for* P_n *is a probabilistic two phase circuit. We write* $P_n(\pi)$ *to denote the execution of* P_n *with the randomness fixed to* $\pi \in \{0,1\}^n$ *and* $C_n(\rho) := (C_1, C_2)(\rho)$ *to denote the execution of both* C_1 *and* C_2 *with the randomness fixed to* $\rho \in \{0,1\}^*$.

In the first phase, the problem poser $P_n(\pi)$ *and the solver* $C_1(\rho)$ *interact. As the result of the interaction* $P_n(\pi)$ *outputs a verification circuit* Γ_V *and a hint circuit* Γ_H. *The circuit* $C_1(\rho)$ *produces no output. The circuit* Γ_V *takes as input* $q \in Q$, *an answer* $y \in \{0,1\}^*$ *and outputs a bit. We say that an answer* (q, y) *is a correct solution if and only if* $\Gamma_V(q, y) = 1$. *The circuit* Γ_H *on input* $q \in Q$ *outputs a hint such that* $\Gamma_V(q, \Gamma_H(q)) = 1$.

In the second phase, C_2 *takes as input* $x := \langle P_n(\pi), C_1(\rho)\rangle_{trans}$ *and has oracle access to* Γ_V *and* Γ_H. *The execution of* C_2 *with the input* x *and the randomness fixed to* ρ *is denoted by* $C_2(x, \rho)$. *The queries of* C_2 *to* Γ_V *and* Γ_H *are* *called verification queries and hint queries respectively. We say that the circuit* C_2 succeeds *if and only if it makes a verification query* (q, y) *such that* $\Gamma_V(q, y) = 1$ *and it has not previously asked for a hint query on* q.

We use the term *weakly verifiable* to emphasize that there is no easy way for a solver to check correctness of a solution, except asking a verification query.

We call a puzzle *dynamic* if the number of hint queries is greater than zero. Furthermore, we say that a puzzle is *interactive* if in the first phase the number of messages exchanged between a problem poser and a problem solver is greater than one.

Definition 3.1 generalizes and combines previous approaches that studies *weakly verifiable puzzles* [CHS04], *dynamic weakly verifiable puzzles* [DIJK09] and *interactive weakly verifiable puzzles* [HS10].

There is no loss of generality in assuming that a problem poser and a problem solver are defined by probabilistic circuits. Definition 3.1 embraces also a case where a problem poser and a problem solver are probabilistic polynomial time algorithms. We use the well know fact [Hol13b] that a probabilistic polynomial time algorithms can be transformed into an equivalent family of Boolean circuits of polynomial size[1].

3.2 Examples

In this section we give examples of cryptographic constructions that are different types of weakly verifiable puzzles.

3.2.1 Message Authentication Codes

We consider the setting in which two parties a *sender* and a *receiver* communicate over an insecure channel. The messages of the sender may be intercepted, modified, and replaced by a third party called an *adversary*. The receiver needs a way to ensure that received messages have been indeed sent by the sender and have not been modified by the adversary. The solution is to use *message authentication codes*.

Loosely speaking, the message authentication codes may be explained as follows. Let sender, receiver, and adversary be polynomial time algorithms, and messages be represented as bitstrings. Furthermore, we assume that the sender and the receive share a secrete key to which an adversary has no access. The sender appends to every message a tag which is computed as a function of the key and the message. The receiver, using the key, has a way to check whether

[1]Theorem 6.10 from [Hol13b] is stated for probabilistic oracle programs with single bit of outputs, but it can be adopted to a case when an output is longer than a single bit.

an appended tag is valid for a received message. The receiver accepts a message if the tag is valid, otherwise it rejects. We require that it is hard for the adversary to find a tag and a message that is accepted by the receiver with non-negligible probability. We give the following formal definition of *Message Authentication Code* based on [Mau13] and [Gol04].

Definition 3.2 (Message Authentication Codes) *Let $\mathcal{M}$ be a set of messages, $\mathcal{K}$ a set of keys and $\mathcal{T}$ a set of tags where $n \in \mathbb{N}$. We define the* message authentication code (MAC) *as an efficiently computable function $\mathcal{M} \times \mathcal{K} \to \mathcal{T}$. Furthermore, we say that MAC is secure if it satisfies the following condition:*

Let $k \xleftarrow{\$} \mathcal{K}$ be fixed and Γ_H be a polynomial size circuit that takes as input a message $m \in \mathcal{M}$ and outputs a tag $t \in \mathcal{T}$ such that $f(m,k) = t$ where a key k is hard-coded in Γ_H. We say that MAC is secure if there is no probabilistic polynomial time algorithm with oracle access to Γ_H that with non-negligible probability outputs a message $m \in \mathcal{M}$ as well as a corresponding tag $t \in \mathcal{T}$ such that $f(m,k) = t$, and Γ_H has not been queried for a tag of message m.

We show how MAC is captured by notion of dynamic interactive weakly verifiable puzzles. For fixed n the sender corresponds to a problem poser, the adversary to a problem solver, and the key is a bitstring $\pi \in \{0,1\}^n$ taken as auxiliary input by a problem poser. In the first phase, which is non interactive, the problem poser outputs a hint circuit Γ_H and a verification circuit Γ_V where both circuits have hard-coded π. The circuit Γ_H takes as input a message and outputs a tag for this message. The circuit Γ_V that as input $m \in \mathcal{M}$ and $t \in \mathcal{T}$ and outputs a bit 1 if and only if $f(m,\pi) = t$.

In the second phase an adversary takes no input (x^* is empty string) and is given oracle access to Γ_H and Γ_V. A task of finding by an adversary a valid tag $t \in \mathcal{T}$ for a message $m \in \mathcal{M}$ such that a hint for m has no been asked before corresponds to asking a successful verification query by a problem poser to Γ_V.

3.2.2 Public Key Signature Scheme

First we give a definition of public key encryption scheme, and what it means for such a scheme to be secure. These definitions are based on [Gol04].

Definition 3.3 (Public key signature scheme) *Let $\mathcal{Q}$ be the set of messages. A* public key signature scheme *is defined by a triple of probabilistic polynomial time algorithms: G – the key generation algorithm, V – the verification algorithm, S – the signing algorithm, such that the following conditions are satisfied:*

- *$G(1^n)$ outputs a pair of bitstrings $k_{priv} \in \{0,1\}^n$ and $k_{pub} \in \{0,1\}^n$ where n is a security parameter. We call k_{priv} a private key and k_{pub} a public key.*

- *The signing algorithm S takes as input $k_{priv} \in \{0,1\}^n$, $q \in \mathcal{Q}$ and outputs a signature $s \in S$.*
- *The verification algorithm V takes as input $k_{pub} \in \{0,1\}^n$, $q \in \mathcal{Q}$, and $s \in S$ and outputs a bit $b \in \{0,1\}$.*
- *For every k_{priv}, k_{pub} output by G and every $q \in \mathcal{Q}$ it holds*

$$\Pr[V(k_{pub}, q, S(k_{priv}, q))] = 1,$$

where the probability is over the random coins of V and S.

We say that $s \in S$ is a *valid* signature for $q \in \mathcal{Q}$ if and only if $V(k_{pub}, q, s) = 1$.

Definition 3.4 ***(Security of public key signature scheme with respect to a chosen message attack)*** *Let an* adversary *A be a probabilistic polynomial time algorithm that takes as input k_{pub} and has oracle access to S. We say that A* succeeds *if it finds a signature $s \in S$ for a message $q \in \mathcal{Q}$ such that $V(k_{pub}, q, s) = 1$ and the oracle S has not been queried for a signature of q. The public key encryption scheme is* secure *if there is no polynomial time adversary that succeeds with non-negligible probability.*

We will show now that a public key signature scheme defined as above can be represented as a dynamic weakly verifiable puzzle. Let a problem poser correspond to entity that generates k_{pub} and k_{priv} and a problem solver to an adversary. In the first phase, the problem poser uses algorithm $G(1^n)$ to obtain k_{pub}, k_{priv} and sends to the adversary the public key k_{pub}. Then the problem poser generates a hint circuit Γ_H and a verification circuit Γ_V. The hint circuit Γ_H takes as input $q \in \mathcal{Q}$ and outputs a signature for q. The verification circuit Γ_V takes as input $s \in S$ and $q \in \mathcal{Q}$ and checks whether $s \in S$ is a valid signature for $q \in \mathcal{Q}$. In the second phase, the problem solver takes as input a transcript of message from the first round which consists solely of a single message k_{pub}. Additionally, it is given oracle access to Γ_V and Γ_H. It is clear that if the adversary asks a successful verification query (q, s), then it also forges a signature.

Thus, a game of forging a signature of a public key signature schemes is a weakly verifiable puzzle that is dynamic but not interactive as in the first phase only a single message is sent.

3.2.3 Bit Commitments

Let us consider the following *bit commitment protocol* that involves two parties a *sender* and a *receiver*. We suppose that the sender and the receiver are polynomial time probabilistic algorithms. The protocol consists of a *commit phase* and a *reveal phase*. In the commit phase the sender and the receiver interact, as the result the sender commits to a value $b \in \{0,1\}$. In the reveal phase the sender opens the commitment by sending to the receiver a pair (y, b') where $y \in \{0,1\}^*$ and $b' \in \{0,1\}$. We require that after the commit phase it is hard for the receiver to correctly guess b. Additionally, in the *reveal phase* it should be hard for the sender to persuade the receiver that it was committed to the value $\neg b$.

We base the following definition of *bit commitment protocol* on [Hol13a].

Definition 3.5 (Bit Commitment Protocol) *For a security parameter $n \in \mathbb{N}$ a* bit commitment protocol *is defined by a pair (S_n, R_n) where $S_n = (S_1, S_2)$ is a two phase probabilistic circuit, and R_n is a probabilistic circuit. We call S_n a sender and R_n a receiver. The circuit S_1, used in the commit phase, takes as input a pair (b, ρ_S) where $b \in \{0,1\}$ is interpreted as a bit to which S_n commits, and $\rho_S \in \{0,1\}^n$ is the randomness used by the algorithm S_n. The receiver R_n takes only auxiliary input $\rho_R \in \{0,1\}^*$ that is the randomness used by R_n. The protocol consists of two phases. In the commit phase, circuits S_1 and R_n engage in the protocol execution. As the result S_1 commits to b and R_n generates a verification circuit Γ_V. The circuit Γ_V takes as input a bit $b' \in \{0,1\}$ and a bitstring $y \in \{0,1\}^*$ and outputs a bit. In the reveal phase the circuit S_2 takes as input a transcript communication transcript from the first phase $\langle S_1, R_n\rangle_{trans}$, the bitstring ρ_s and returns (b', y). We require a bit commitment protocol to have the following properties:*

(Correctness) *For a fixed $b \in \{0,1\}$ we have*

$$\Pr_{\substack{\rho_S \in \{0,1\}^*, \rho_R \in \{0,1\}^n \\ \Gamma_V := \langle S_1(b,\rho_S), R_n(\rho_r)\rangle_{R_n} \\ (b',y) := S_2(\langle S_1(b,\rho_s), R_n(\rho_R)\rangle, \rho_S)}} \left[\Gamma_V(b', y) = 1\right] \geq 1 - \varepsilon(n),$$

where $\varepsilon(n)$ is a negligible function of n.

(Hiding)

> ***TODO:*** *Describe it using equations, define somehow the guess of R? Maybe as a last message in the first phase of communication*

Probability over random coins of S_n and R_n that any polynomial size circuit can guess bit b correctly after the commit phase is at most $\frac{1}{2} + \varepsilon(n)$ where $\varepsilon(n)$ is a negligible function of n.

(Binding) *For every polynomial size circuit S_n we have*

$$\Pr_{\substack{\Gamma_V := \langle S_1, R\rangle_R \\ (b',y) := S_2(\rho_S)}} [\Gamma_V(0, y_0) = 1 \wedge \Gamma_V(1, y_1) = 1] \leq \varepsilon(k),$$

where $\varepsilon(k)$ is a negligible function in k.

TODO: This is not clear ... access to the oracle etc.
TODO: Why it is not possible to verify – i.e. the sender does not even know the function that is used by the receiver to validate decommitment.

The bit commitment protocols can be generalized as an interactive weakly verifiable puzzle as follows. The number of hint queries amounts to zero and the number of the verification queries is at most one. The sender corresponds to a problem solver, and the receiver is a problem poser. Additionally, we require the problem solver to ask a verification query (b', y) only on $b' := \neg b$ where b is a bit to which the problem solver is committed after the first phase. The first phase corresponds to the commit phase. The second phase is the reveal phase where the problem poser tries to find a bitstring y such that $\Gamma_V(\neg b, y) = 1$.

3.2.4 Automated Turing Tests

The goal of *Automated Turing Tests* is to distinguish humans from computers which is frequently used to prevent computer programs from accessing resources for humans. An example is *CAPTCHA* defined first in [VABHL03]. Loossly speaking, CAPTCHA is a test that human can solve with probability close to 1, but it is hard to write a computer program that has a success probability comparable to the one achieved by humans. An example of CAPTACHA is an image depicting a distorted text. Most humans guess the text which is displayed on the image correctly, but it might be hard to write a program for which it would also be easy. We note that the definition of hardness has not been particular well defined , and bases on opinions AI community opinions that distinguish between hard and easy AI problems [VABHL03].

CAPTCHAs based on guessing the distorted text are weakly verifiable puzzles. In the first round the problem poser and problem solver engage in interactive protocol, such that after the execution of the protocol the problem poser has a way to verify the solution. The problem poser in the second round takes as input a distorted image, and try to guess the text that was used to generated it. The standard CAPTCHAs are non-dynamic, as the problem poser does not gain access to the hint oracle and asks only a single verification query.

Our definition captures also the above type of problems, additionally it is also applicable in the broader context for a different AI problems.

As it is not know how good the possible algorithm can be to recognize CAPTCHA it is likely that the gap between human performance and a performance of computer programs may be small. Therefore, it is of interest to find a way to amplify this gap. It turns out that it is indeed possiThe first ble which for not dynamic puzzles was proved in [HS10]. The proof presented in Chapter 4 applies also to the dynamic context.

TODO: Give an optimization problem for gap amplification

3.3 Previous results

Different types of weakly verifiable cryptographic primitives have been studied in a series of works [CHS04, DIJK09, HS10]. This section is intended not only to give a short overview of techniques used in these works, but aims also to provide some intuition and insight into the problem of hardness amplification of dynamic interactive weakly verifiable puzzles.

3.3.1 Result of R.Canetti, S.Halevi, and M.Steiner

The notion of the *weakly verifiable puzzle* has been coined by R.Canetti, S.Halevi, and M.Steiner in the paper *Hardness amplification of weakly verifiable puzzles* [CHS04]. In comparison to Definition 3.1 the puzzles considered in [CHS04] are neither dynamic nor interactive. Moreover, the number of verification queries is limited to one. This constitutes a simplified case to the one considered in this Thesis. In this section we provide the definition of weakly verifiable puzzles (WVP) that closely follows the one contained in [CHS04], and state the theorem of hardness amplification of weakly verifiable puzzles stated in [CHS04]. Finally, we give an intuition behind the proof of this theorem. It is noteworthy that the main proof of this Thesis, contained in Chapter 4, uses many ideas from the paper of R.Canetti, S.Halevi, and M.Steiner [CHS04].

Definition 3.6 (Weakly Verifiable Puzzles) *A* weakly verifiable puzzle *is defined by a pair of polynomial time algorithms: a probabilistic puzzle–generation algorithm G and a deterministic verification algorithm V. We write $G(1^k, \rho)$ to denote that G takes as input a bitstring 1^k, where k is a security parameter, and as auxiliary input a bitstring $\rho \in \{0,1\}^*$ which is the randomness used by G. The algorithm G outputs $p \in \{0,1\}^*$ and a check information $c \in \{0,1\}^*$. The* verifier *V is a deterministic algorithm that takes as input p, c, an answer $a \in \{0,1\}^*$ and outputs $b \in \{0,1\}$.*

A solver *S for G is a polynomial time probabilistic algorithm that takes as input p and outputs a. We denote the randomness used by S as $\pi \in \{0,1\}^*$, and define the* success probability *of S in solving a puzzle defined by P as*

$$\Pr_{\substack{\rho\in\{0,1\}^*,\pi\in\{0,1\}^* \\ (p,c):=G(1^k,\rho) \\ a:=S(p,\pi)}}\Big[V(p,c,a)=1\Big].$$

We write $P := (G, V)$ to denote a weakly verifiable puzzle P defined by algorithms G and V.

We compare the above definition with Definition 3.1. First, we note that probabilistic polynomial time algorithms can be converted into families of polynomial size circuits. Next, we see that in Definition 3.6 the algorithm G is parameterized by a bitstring 1^k meaning that the length of a random bitstring taken by G is bounded by $poly(k)$. For a fixed k, without loss of generality, we can model the algorithm $G(1^k, \rho)$ as a polynomial size circuit that does not take as input the bistring 1^k, but just a bitstring ρ of length $poly(k)$. The security parameters from Definition 3.6 and Definition 3.1 are not equivalent, as in the later definition the security parameter limits the length of the random bistring. Next, in Definition 3.6 a verification algorithm takes as input p, c, a. Again, without loss of generality, we can assume that bitstrings p and c are hard-coded in the circuit Γ_V from Definition 3.1. Hence, the algorithm V corresponds to Γ_V.

Definition 3.7 (n-fold repetition of Weakly Verifiable Puzzles) *Let $n \in \mathbb{N}$, and a weakly verifiable puzzle $P = (G, V)$ be fixed. We define the n-fold repetition of P as a weakly verifiable puzzle where the puzzle–generation algorithm $G^{(n)}$ takes as input 1^k, $\rho \in \{0,1\}^*$ as an auxiliary input and outputs tuples $p^{(n)} := (p_1, \dots, p_n) \in \{0,1\}^*$ and $c^{(n)} := (c_1, \dots, c_k) \in \{0,1\}^*$, where for each $1 \leq i \leq n$ pair (p_i, c_i) is an independent instance of weakly verifiable puzzles defined by G with security parameter k and V. Finally, the verification algorithm $V^{(n)}$ takes as input $p^{(n)}$, $c^{(n)}$, an answer $a^{(n)}$, and outputs $b \in \{0,1\}$ such that $b = 1$ if and only if for all $1 \leq i \leq n$ we have $V(p_i, c_i, a_i) = 1$. We write $P^{(n)}$ to denote the n–fold repetition of P.*

Moreover, the n-fold repetition of weakly verifiable puzzles is solved successfully if and only if all n puzzles are solved successfully. In this Thesis we are interested in a more general situation where a monotone function $g : \{0,1\}^n \to \{0,1\}$ is used to decide which coordinates of the n-fold repetition of puzzles have to be solved correctly. A precise definition is given in Section 4.1. Clearly, we can assume that g is such that all coordinates have to be solved successfully which matches the case considered in Definition 3.7.

The main theorem proved in [CHS04] states that it is possible to turn a good solver for $P^{(n)}$ to a good solver for P.

Theorem 3.8 (Hardness amplification of Weakly Verifiable Puzzles) *Let $n : \mathbb{N} \to \mathbb{N}$ and $\delta : \mathbb{N} \to (0,1)$ be efficiently computable functions and $q \in \mathbb{N}$ a slackness parameter. Moreover, let $P = (G, V)$ be a weakly verifiable puzzle as in Definition 3.6. We denote the running time of the puzzle–generation algorithm G for P by T_G, and of the verification algorithm V for P by T_V. If $S^{(n)}$ is a solver for the n-fold repetition of P that success probability is at least δ^n and running time is T, then there exists a solver S for G with success probability at least $\delta(1 - \frac{1}{q})$ and with running time $O\Big(\frac{nq^3}{\delta^{2n-1}}(T + nT_G + nT_V)\Big)$.*

The following algorithm is used in R.Cannetti, S.Halevi, and M.Steiner to prove Theorem 3.8. It uses a solver $S^{(n)}$ for the n-fold repetition of P that succeeds with probability at least δ^n to solve a single puzzle with probability at least $\delta(1-\frac{1}{q})$. Use a slackness parameter q as it is not possible to achieve the perfect hardness amplification. We note that in the analysis of running time of *CHS–solver* we take into account time needed for oracle calls to $S^{(n)}, V, G$. To make the notation shorter in the following code excerpts we do not write randomness used by the generator G explicitly.

Algorithm: $\mathit{CHS\text{–}solver}^{S^{(n)},V,G}(p, n, k, q, \delta)$

Oracle: A solver $S^{(n)}$ for $P^{(n)}$, a verification algorithm V for P, a puzzle–generation algorithm G for P.
Input: A bistring $p \in \{0,1\}^*$, parameters n, k, q, δ.

$\mathit{prefix} := \emptyset$
for $i = 1$ **to** $n - 1$ **do:**
  $p^* := \mathit{ExtendPrefix}^{S^{(n)},V,G}(\mathit{prefix}, i, n, k, q, \delta)$
  if $p^* = \perp$ **then return** $\mathit{OnlinePhase}^{S^{(n)},V,G}(\mathit{prefix}, p, i, n, k, q, \delta)$
  else $\mathit{prefix} := \mathit{prefix} \circ p^*$
$a^{(n)} := S^{(n)}(\mathit{prefix} \circ p)$
return a_n

Algorithm: $\mathit{OnlinePhase}^{S^{(n)},V,G}(\mathit{prefix}, p, v, n, k, q, \delta)$

Oracle: A solver algorithm $S^{(n)}$ for $P^{(n)}$, a puzzle–generation algorithm G for P, a verification algorithm V for P.
Input: A $(v-1)$–tuple of bitstrings *prefix*, a bitstring $p \in \{0,1\}^*$, parameters v, n, k, q, δ.

Repeat $\left\lceil \frac{6q\ln(6q)}{\delta^{n-v+1}} \right\rceil$ times
  $((p_{v+1}, \ldots, p_n), (c_{v+1}, \ldots, c_n)) := G^{(n-v-1)}(1^k)$
  $a^{(n)} := S^{(n)}(\mathit{prefix}, p, p_{v+1}, \ldots, p_n)$
  if $\forall_{v+1 \le i \le n} V(p_i, c_i, a_i) = 1$ **then return** a_v
return $\perp$

Algorithm: $\mathit{ExtendPrefix}^{S^{(n)},V,G}(\mathit{prefix}, i, n, k, q, \delta)$

Oracle: A solver algorithm $S^{(n)}$ for $P^{(n)}$, a puzzle–generation algorithm G for P, a verification algorithm V for P.

Input: A $(i-1)$–tuple of puzzles *prefix*, parameters i, n, k, q, δ.

Repeat $\left\lceil \frac{6q}{\delta^{n-v+1}} \ln(\frac{18qn}{\delta}) \right\rceil$ times **do:**

$(p^*, c^*) := G(1^k)$

$\bar{\mu}_i := \mathit{EstimateResSuccProb}^{G,V}(\mathit{prefix} \circ p^*, i, n, k, q, \delta)$

if $\bar{\mu}_i \geq \delta^{n-i}$ **then return** p^*

return $\perp$

Algorithm: $\mathit{EstimateResSuccProb}^{S^{(n)},V,G}(\mathit{prefix}, i, n, k, q, \delta)$

Oracle: A solver algorithm for $P^{(n)}$, a verification algorithm V for P, a generation algorithm G for P

Input: A $(n-i)$–tuple of puzzles *prefix*, parameters i, n, k, q, δ.

$\mathit{successes} := 0$

Repeat $M := \left\lceil \frac{84q^2}{\delta^{n-i}} \ln\left(\frac{18qn \cdot N_i}{\delta}\right) \right\rceil$ times

$((p_{i+1}, \cdots, p_n), (c_{i+1}, \cdots, c_n)) := G^{(n-i)}(1^k)$

$a^{(n)} := A(\mathit{prefix}, p_{i+1}, \cdots, p_n)$

if $\forall_{1 \leq i \leq n} : V(p_i, c_i, a_i) = 1$ **then** $\mathit{successes} := \mathit{successes} + 1$

return $\mathit{successes}/M$

A detail proof of Theorem 3.8 is presented in [CHS04]. We limit ourselves to providing an intuition why the above algorithm to transfrom a good solver for n–wise direct product of P to a good solver for P.

Let us consider the n-fold repetition of P, and a deterministic solver $S^{(n)}$ for $P^{(n)}$. Furthermore, we denote by write $p^{(n)}, c^{(n)}$ to denote the output of $G^{(n)}$. We define a matrix M as follows. The columns of M are labeled with all possible bitstrings p_1 whereas the rows corresponds to all possible tuples $(p_2, \ldots, p_n)$ where $G^{(n)}$ is executed with different randomness. A cell of M contains a binary n-tuple such that the i-th bit equals 1 if and only if $V_i(p_i, c_i, a_i) = 1$, where $a^{(n)} := S^{(n)}(p^{(n)})$ and $p^{(n)}$ is a tuple of bitstring inferred by a column and row of the cell. We make the following observation, with a simplifying assumption that the problem solver is deterministic.

Observation 3.9 *For a deterministic polynomial time algorithm $S^{(n)}$ that successfully solves the n-fold repetition of P with probability at least δ^n, the matrix M defined as above has either a column with $\delta^{(n-1)}$ fractions of cells that are all one vectors, or a conditional probability that a cell is of the form 1^n given that the last $(n-1)$ bits of the cell are equal 1 is at least δ.*

We show how the algorithm *CHS–solver* uses Observation 3.9 to solve WVP with probability at least $\delta(1 - \frac{1}{q})$. The algorithm starts with the first position and tries to fix a puzzle such that the success probability of $S^{(n)}$ on the remaining $(n-1)$ position is at least $\delta^{(n-1)}$. If it is possible to find p^* such that this condition is satisfied, then we fix this p^* on this position and repeat the whole procedure again in the consecutive iteration for the next position. If *CHS–solver* fails to find such a bitstring p^*, then we assume that there is no column of M that contains $\delta^{(n-1)}$ fraction of cells that are of the form 1^n. We use Observation 3.9 and hope that the conditional probability of solving the first puzzle given that all puzzles on the remaining position are solved successfully is at least δ. We place p (which denotes the input puzzle) on this position and note that all remaining puzzles are generated by *CHS–solver*. Thus, it is possible to efficiently verify whether these puzzles are successful solved by $S^{(n)}$.

Obviously, the algorithm *CHS–solver* can still fail. First, it may happen that it does not find a column with a high fraction of puzzles that are solved successfully, although such a column exists. Secondly, we can not exclude a situation where no such column exists, but the algorithm fails to find a cell such that last $(n-1)$ bits are 1. Finally, it is also possible that an estimate returned by *EstimateResSuccProb* is incorrect.

It is possible to show that all these events happen with negligible probability. Therefore, at least intuitively we see that the algorithm *CHS–solver* solves a single WVP puzzle successfully with probability at least $\delta(1-\frac{1}{q})$ almost surely.

In Chapter 4 we study a more general class of puzzles that are not only weakly verifiable but also dynamic and interactive. Furthermore, we allow a more general situation when a solver successfully solves the n-fold repetition of puzzles [2] although it solved successfully only on a subset $S \subset \{1, 2, \ldots, n\}$ of puzzles P.

It turns out that it is possible to use a similar technique of fixing puzzles on consecutive positions of the n-fold repetition of puzzles to prove the hardness amplification in this more general case.

3.3.2 Results of Y.Dodis, R.Impagliazzo, R.Jaiswal, V.Kabanets

Some of the cryptographic constructions presented in Section 3.2 are not only weakly verifiable but also dynamic (MAC and SIG). This type of puzzles are defined and studied in [DIJK09]. We give a short overview of this work, state the definition of a *dynamic weakly verifiable puzzle* that closely follows the one included in [DIJK09]. Finally, we provide intuition for the proof of the hardness amplification of DWVP included in [DIJK09].

[2] Actually, in Chapter 4 we define the k-wise repetition of puzzles which in case of WVP is equivalent to the n-fold repetition of puzzles.

Definition 3.10 (Dynamic Weakly Verifiable Puzzle.) *A* dynamic weakly verifiable puzzle *(DWVP) is defined by a distribution $\mathcal{D}$ on pairs (x, α) where $\alpha \in \{0,1\}^*$ is an advice used to generate and evaluate responses to $x \in \{0,1\}^*$. Furthermore, we consider a set $\mathcal{Q}$ of indices and a probabilistic polynomial–time computable relation R such that $R(\alpha, q, r) = 1$ if and only if r is a correct answer to $q \in \mathcal{Q}$ on the set of puzzle determined by α. Finally, let $H(\alpha, q)$ be a probabilistic polynomial–time computable* hint *relation.*

A solver S takes as input x and can ask hint queries on $q \in \mathcal{Q}$ which are answered using $H(\alpha, q)$ and verification queries (q, r) that answers whether $R(\alpha, q, r) = 1$. We say that S succeeds if and only if it makes a verification query on (q, r) such that it has not previously asked for a hint query on this q. We write $P := (\mathcal{D}, R, H)$ to denote a DWVP with a distribution $\mathcal{D}$ of pairs (x, α), and R, H being a verification and hint relations respectively.

The definition stated in Section 3.1 generalizes for dynamic weakly verifiable puzzles. Instead of considering a distribution on pairs (x, α) we use a problem poser. Furthermore, the problem poser may interact in the first phase with the problem solver. Thus, in our case the distribution of puzzles is defined by both the problem poser and the problem solver. The hint and verification relations corresponds to the hint and verification circuits generated by the problem poser.

We define the *n-wise direct product of DWVPs*, which is conceptually similar to the n-fold repetition of WVPs.

Definition 3.11 (n-wise direct product of DWVPs) *For a dynamic weakly verifiable puzzle $P := (\mathcal{D}, R, H)$ we define the n-wise direct product of P as a DWVP with a distribution $\mathcal{D}^n$ on tuples $(x_1, \alpha_1), \ldots, (x_n, \alpha_n)$. Furthermore, the hint relation is defined by $H^n(q, \alpha_1, \ldots, \alpha_n) := (H(\alpha_1, q), \ldots, H(\alpha_n, q))$ and the verification relation $V^n(q, x_1, \ldots, x_n)$ evaluates to 1 if and only if for $1 \leq i \leq n$ at least $n - (1-\gamma)\delta n$ is such that $V(q, x_i, \alpha_i) = 1$ where $0 \leq \gamma, \delta \leq 1$. We write $P^{(n)} := (D^{(n)}, H^{(n)}, V^{(n)})$ to denote the n-wise direct product of $P := (D, H, V)$.*

In contrast to the n-fold repetition of puzzles defined in previous section, here we require a solver to succeed only on the fraction of puzzles.

Theorem 3.12 *Let $S^{(n)}$ be a probabilistic algorithm for the $P^{(n)}$ that succeeds with probability at least ε, where $\varepsilon \geq (800/\gamma\delta) \cdot (h+v) \cdot e^{-\gamma^2 \delta n/40}$, and h and v denote the number of hint and verification queries asked by $S^{(n)}$ respectively. Then there exists a probabilistic algorithm S that succeeds in solving P with probability at least $1 - \delta$ making $O(h(h+v)/\varepsilon) \cdot \log(1/\gamma\delta)$ hint queries and at most one verification query. Furthermore, the running time poly$(h, v, \frac{1}{\varepsilon}, t, \omega, \log(1/\gamma\delta))$ where ω is time needed to ask a single hint query.*

We define the *success probability* of a solver S for DWVP as

TODO: Define success probability

It worth seeing why the approach presented in the previous section can not be applied in the setting of dynamic weakly verifiable puzzles (moving aside for a moment the issue of solving only a fraction of puzzle successfully). For DWVP the algorithm *CHS–solver* breaks in the *OnlinePhase* where the solver $S^{(n)}$ can be called multiple times. It is possible that in one of these runs $S^{(n)}$ asks a hint query on q for which in one of the later runs a verification query (q, r) is asked that would make *CHS–solver* successfully solves the input puzzle. Thus, we can not dismiss the situation where the success probability of the solver $S^{(n)}$ decrease in after some iterations.

The solution proposed in [DIJK09] is to partition the set Q into a set of *attacking queries* Q_{attack} and a set of *advice queries* Q_{adv}. The idea is to allow a solver for the n-wise direct product to ask a hint queries only on $q \in Q_{adv}$, and to halt the execution whenever a hint query is asked on $q \in Q_{attack}$.

It is possible, for a solver S that asks at most h hint queries and v verification queries, to find a function $Q \to \{0, 1, \ldots 2(h+v)\}$ such that the success probability of S with respect to Q_{attack} and Q_{adv} is multiplied by $O(\frac{1}{h+v})$. If h and v are not to big then the success probability of S can still be substantial.

Additionally, we define a canonical success probability with respect to a *hash* function as

TODO: Define success probability and canonical success probability

More formally, in [DIJK09] the following lemma is proved.

Lemma 3.13 *Let S be a solver for DWVP which success probability is at least ε, the running time is at most t, and the number of hint and verification queries is at most h and v respectively, there exists a probabilistic algorithm that runs in time $poly(h, v, \frac{1}{\varepsilon}, t)$ that outputs a function $hash : Q \to \{0, 1, \ldots, 2(h+v)\}$ such that the canonical success probability of S with respect to hash is at least $\frac{\varepsilon}{8(h+v)}$.*

A function *hash* can be found by using a natural sampling technique. We follow the same approach of domain partitioning in Section 4.1.3. Let $H_\alpha(q)$ denote a polynomial time probabilistic algorithm that takes as input q, has hard-coded α and outputs $H(\alpha, q)$.

Algorithm: $DWVP\text{–}solver^{S^{(n)}, hash, H_\alpha^{(n)}, R_\alpha^{(n)}}(x)$

Oracle: A solver $S^{(n)}$ for $P^{(n)}$, a function $hash : Q \to \{0, 1, \ldots, 2(h+v)\}$.

Input: A bistring $x \in \{0,1\}^*$.

Repeat at most $O(\frac{h+v}{\varepsilon} \cdot \log(\frac{1}{\gamma\delta}))$ times
  Choose uniformly at random position $i \in \{1,\ldots,n\}$ for x.
  Generate $(x_1,\alpha_1),\ldots,(x_{i-1},\alpha_{i-1}),(x_{i+1},\alpha_{i+1}),\ldots,(x_n,\alpha_n)$ using $(n-1)$ calls to P each time with the fresh randomness.
  run $S^{(n)}(x_1,\ldots,x_{i-1},x,x_{i+1},\ldots,x_n)$
    if $S^{(n)}$ asks a hint query on q **then**
      if $hash(q) \neq 0$ **then** abort current run of $S^{(n)}$
      Ask a verification query $r := H(q)$
      Let $(r_1,\ldots,r_{i-1},r_{i+1},\ldots,r_n)$ be hints for query q for puzzle sets $(x_1,\ldots,x_{i-1},x_{i+1},x_n)$
      Answer the hint query of $S^{(n)}$ using $(r_1,\ldots,r_{i-1},r,r_{i+1},r_n)$
    if $S^{(n)}$ asks a verification query $(q,r_1,\ldots,r_n)$ **then**
      if $hash(q) = 0$ **then** answer the query with 0
      Let $m := |j : V(q,r_j) = 1, j \neq i|$
      if $m \geq n - n(1-\gamma)\delta$ **then**
        make a verification query (q,r_i) and halt.
      else with probability $\rho^{m-n(1-\gamma)\delta}$ ask a verification query (q,r_i) and halt.
      Halt the current run of $S^{(n)}$ and go to the next iteration.
return $\perp$

The above algorithm substantially differ from the one used in [CHS04].

In the above algorithm we execute multiple times a solver $S^{(n)}$ for the k-wise direct product of DWVPs. In each iteration the position for $x \in \{0,1\}^*$ is chosen uniformly at random. The remaining $(n-1)$ puzzles are generated by the algorithm, thus it is possible to answer all hint and verification queries for these puzzles. We use a function *hash* to partition the query domain. We assume that *hash* is such that the success probability of $S^{(n)}$ with respect to *hash* is at least $\frac{\delta}{8(h+v)}$. We check on which q the solver $S^{(n)}$ asks hint and verification queries. If a hint query is asked on q such that $hash(q) = 0$ then the execution of $S^{(n)}$ is aborted and we goto to the next iteration. This way me make sure that the algorithm never asks a hint query that could prevent a verification query to succeed.

If a verification query is asked on q such that $hash(q) \neq 0$ we answer such a verification query with 0.

Finally, in case when $S^{(n)}$ asks a verification query using index q such that $hash(q) = 0$, then we use a soft decision system to decide whether to ask a verification query. The idea is that if there are many puzzles among the ones generated by the algorithm is solved correctly then it is likely that also the input puzzle is solved correctly. We also have to discount $\gamma\delta n$ to take into account that we require not all puzzles to be solved successfully. The detail calculations provided in [DIJK09] shows that this approach yields a demanded result. More details can be found in [DIJK09, IJK07].

TODO: Say why it makes sense to consider a threshold function

In case of weakly verifiable primitives like CAPTCHAs we want to distinguish between humans and computers. We assume that most people have slightly higher probability of solving these kind of puzzles than the best algorithms. Still, it may happen that humans do not solve all puzzles. Therefore, we would like to introduce a threshold function such that on average solutions of humans are treated as solved successfully but the ones of computer programs are classified as not successful. This is a motivation to study the situations where only some fractions of puzzles is solved successfully.

In Chapter 4 we consider a weakly verifiable puzzles that are interactive and dynamic. We use similar technique to partition domain Q into advice and hint queries as presented in [DIJK09]. Instead of the requirement to succeed only on the fraction of puzzles we consider an arbitrary, monotone function $g : \{0,1\}^n \to \{0,1\}$ that determine on which coordinates the solver has to succeed in order to successfully solve the n-wise direct product of puzzles.

To show the hardness amplification for the n-wise direct product with the domain partitioned we use the approach similar to the one presented in Section 3.3.1. Namely, we try to find a good position for the input puzzle instead of choosing this position on random as in [DIJK09].

TODO: Give more intuition behind the proof i.e. why it works? why it is complicated.

3.3.3 Results of T.Holenstein and G.Scheonebeck

The interactive weakly verifiable puzzles have been studied in [HS10]. An example of interactive weakly verifiable puzzles are bit commitments and CAPTCHAs. Additionally, in [HS10] a more general case is introduced when which puzzles of the n-fold repetition should be solved is determined by a monotone function.

Interactive weakly verifiable puzzles has been defined in [HS10] as follows:

Definition 3.14 *An interactive weakly verifiable puzzle is defined by a protocol given by two algorithms P and S, where P, called the problem poser, produces as output a verification circuit Γ. The algorithm S called the problem* *solver produces no output. Furthermore, the* success probability *of algorithm S in solving a wealky verifiable puzzle is (P, S) is:*

$$\Pr\Big[y := \langle P, S\rangle_S ; \Gamma(y) = 1\Big].$$

Definition 3.15 (k-wise direct product of interactive weakly verifiable puzzles) *Let $g : \{0,1\}^k \to \{0,1\}$ be a monotone function, and (P, S) be a fixed interactive weakly verifiable puzzle. The k-wise direct product of (P, S) is an interactive weakly verifiable puzzles in which the sender and the receiver sequentially create k instance of (P, S), which yields circuits $\Gamma^{(1)}, \ldots, \Gamma^{(k)}$ for P. Finally, $P^{(g)}$ outputs the circuit $\Gamma^{(g)}(y_1, \ldots, y_k) := g(\Gamma^{(1)}(y_1), \ldots, \Gamma^{(k)}(y_k))$.*

In comparison to our definition the puzzles considered in [HS10] are clearly not dynamic. The circuit Γ generated by P corresponds to the verification circuit in Definition 3.1. Clearly, the hint circuit is not created.

Finally, T.Holenstein and G.Schonebeck proves the following hardness amplification therorem.

Theorem 3.16 *There exists an algorithm $Gen(C, g, \varepsilon, \delta, n)$ which takes as input a solver circuit C for k-wise direct product of P, a monotone function $g : \{0,1\}^* \to \{0,1\}$, and parameters ε, δ, n. The algorithm Gen outputs a solver circuit D for P such that the following holds. If C is such that*

$$\Pr\Big[\Gamma^{(g)}(\Big\langle P^{(g)}, C\Big\rangle_C = 1)\Big] \geq \Pr_{u \leftarrow \mu_\delta^{(k)}}\Big[g(u) = 1\Big] + \varepsilon,$$

then, D satisfies almost surely,

$$\Pr\Big[\Gamma^{(1)}(\rangle P^{(1)}, D\rangle_D) = 1\Big] \geq \delta + \frac{\varepsilon}{6k}.$$

Additionally, Gen and D only require oracle access to g and C. Furthermore, $Size(D) \leq Size(C) \cdot \frac{6k}{\varepsilon}\log(\frac{6k}{\varepsilon})$, and $Time(Gen) = poly(k, \frac{1}{\varepsilon}, n)$ with oracle calls to C.

First, we notice that the above intuition does not impose any restrictions on the time complexity of the poser and the solver. In previous sections we considered solvers for the n-wise direct product that were compared with the algorithms that either solves all puzzles (in [CHS04]), or allowed a fraction of puzzles to be solved incorrectly ([DIJK09]). In the above definition a monotone function g is considered which generalizes the previous approaches as in [CHS04] the function g is a multiplication of all bits, and in [DIJK09] g is a threshold function that takes value one if and only if a given fraction of puzzles is solved successfully. Furthermore, we emphasize that a monotone restriction on g is essential. Otherwise, for example in case of $g(b) := 1 - b$ an algorithm that deliberately give incorrect solutions satisfies g with probability one whereas an algorithm that solves a puzzle successfully with probability γ succeeds only with probability $1-\gamma$. A desirable property is that an algorithm that solves puzzles with the higher probability performance better than algorithm that success probability is lower.

The proof of T.Holenstein and G.Schoenebeck is similar to the one of R.Canettif and S.Halevi and M.Steiner presented in Section 3.3.1. Similarly, we use the technique to fix consecutive coordinates such that according invariant is maintained. First in order to estimate how much better a solver circuit C for n-wise direct product performs when a puzzle on the first position is fixed a notion of a surplus $S_{\pi^*,b}$ is introduces:

$$S_{\pi^*,b} := \Pr_{\pi^{(k)}}[c \in \mathcal{G}_b | \pi_1 = \pi^*] - \Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_b],$$

which intuitively tells us how much better a solver C performs when a first puzzle is always solved correctly (case when $b = 1$) or is always solved incorrectly (when $b = 0$). Now we observe the following fact. If there exists a puzzle which is fixed on the first position for which the surplus is bigger than $(1-\frac{1}{k})\varepsilon$ then we can fixed a this first puzzle and inductively solve the problem for the $(k-1)$-direct product of puzzles.

TODO: what do it mean

On the other hand, if we there is no such puzzle to fix on the first position it means that when we fix the first bit of g then the performance between the solver C and an algorithm that solves puzzle on each position independently with probability δ is similar. However, we know that when the first bit of a function g is not fixed then the solver C is better. Thus, we draw a conclusion that the puzzle on the first position has to be solved unusually often.

In a case it would be possible to fix all $(k-1)$ puzzles except the last one, the proof become trivial as we know that a function g with the first $k-1$ bits fixed is either the identity or a constant function.

Thus, it is enough to show that if it is not possible to find an estimate that is low then if we place an input puzzle on this position and we can find remaining $k-1$ puzzle such that $c \in \mathcal{G}_1 \setminus \mathcal{G}_0$ then this puzzle is solved with substantial probability. The whole proof is given in [HS10], and requires some probability manipulations. Here, we give a present a simplified version where we assume that once we find a good position on which we can place the input puzzle then we can always find remaining $k-1$ puzzles.

Out proof of hardness amplification for dynamic interactive weakly verifiable puzzles closely follows the one given in [HS10].

TODO: Why do we consider fixing 0/1 on the first position **TODO:** How the technique is generalized to approach of CHS
TODO: Explain why we compare to such a probability i.e. why we consider with μ
TODO: Why the requirement for g being a monotone function is interesting
TODO: Give the intuition behind the proof.
TODO: Give a proof under the simplified assumptions?
1) The algorithm always output an answer
2) For every pi the surpluses $S_{\pi^*,0}$ and $S_{\pi^*,1}$ re less than $(1 - \frac{1}{k})\varepsilon$.

3.4 Limitations of Security Amplification

Chapter 4

Security amplification for dynamic weakly verifiable puzzles

In this chapter we show that it is possible to amplify security of dynamic weakly verifiable puzzles. In section 4.1 we state the theorem, which is next proved in three steps. First, in Section 4.1.3, we show how to use to partition the domain on which hint and verification queries are asked, next we give a prove of security amplification under the simplifying assumption that there is no collisions of hint and verification queries. Finally, in Section 4.1.5 we combine both former steps which yields the desirable result.

4.1 Main theorem

We start by giving the definition of the k-wise direct product of weaky verifiable puzzles.

4.1.1 The k-wise direct product of weakly verifiable puzzle

Definition 4.1 (k-wise direct-product of DWVPs.) *Let $g : \{0,1\}^k \to \{0,1\}$ be a monotone function and $P_n^{(1)}$ a problem poser as in Definition 3.1. The k-wise direct product of $P_n^{(1)}$ is a DWVP defined by a circuit $P_{kn}^{(g)}$. We write $P_{kn}^{(g)}(\pi^{(k)})$ to denote the execution of $P_{kn}^{(g)}$ with the randomness fixed to $\pi^{(k)} := (\pi_1, \ldots, \pi_k)$ where for each $1 \leq i \leq n : \pi_i \in \{0,1\}^n$. Let $(C_1, C_2)(\rho)$ be a solver for $P_{kn}^{(g)}$ as in Definition 3.1. In the first phase, the algorithm $C_1(\rho)$ sequentially interacts in k rounds with $P_{kn}^{(g)}(\pi^{(k)})$. In the i-th round $C_1(\rho)$ interacts with $P_n^{(1)}(\pi_i)$, and as the result $P_n^{(1)}(\pi_i)$ generates circuits Γ_V^i, Γ_H^i. Finally, after k rounds $P_{kn}^{(g)}(\pi^{(k)})$ outputs a verification circuit*

$$\Gamma_V^{(g)}(q, y_1, \ldots, y_k) := g(\Gamma_V^1(q, y_1), \ldots, \Gamma_V^k(q, y_k))$$

and a hint circuit

$$\Gamma_H^{(k)}(q) := (\Gamma_H^1(q), \ldots, \Gamma_H^k(q)).$$

If it is clear from the context, we omit the subscript n and write $P(\pi)$ instead of $P_n(\pi)$ where $\pi \in \{0,1\}^n$.

A verification query (q, y) of a solver C for which a hint query on this q has been asked before cannot be a verification query for which C succeeds. Therefore, without loss of generality, we make the assumption that C does not ask verification queries on q for which a hint query has been asked before. Furthermore, we assume that once C asked a verification query that succeeds, it does not ask any further hint or verification queries.

Experiment $\mathit{Success}^{P,C}(\pi, \rho)$

Oracle: A problem poser P, a solver $C = (C_1, C_2)$ for P.
Input: Bitstrings $\pi \in \{0,1\}^n$, $\rho \in \{0,1\}^*$.
Output: A bit $b \in \{0,1\}$.

---

run $\langle P(\pi), C_1(\rho)\rangle$
    $(\Gamma_V, \Gamma_H) := \langle P(\pi), C_1(\rho)\rangle_P$
    $x := \langle P(\pi), C_1(\rho)\rangle_{trans}$

run $C_2^{\Gamma_V,\Gamma_H}(x, \rho)$
    if $C_2^{\Gamma_V,\Gamma_H}(x, \rho)$ asks a verification query (q, y) s.t. $\Gamma_V(q, y) = 1$ **then**
        return 1
return 0

We define the *success probability* of C in solving a puzzle defined by P as

$$\Pr_{\pi,\rho}[\mathit{Success}^{P,C}(\pi, \rho) = 1]. \tag{4.1}$$

Furthermore, we say that C *succeeds* for π, ρ if $\mathit{Success}^{P,C}(\pi, \rho) = 1$.

Theorem 4.2 (Security amplification for dynamic weakly verifiable puzzles.) *Let $P_n^{(1)}$ be a fixed problem poser as in Definition 3.1 and $P_{kn}^{(g)}$ a problem poser for the k-wise direct product of $P_n^{(1)}$. Additionally, let C be a problem solver for $P_{kn}^{(g)}$ asking at most h hint queries and v verification queries. There exists a probabilistic algorithm Gen with oracle access to a solver circuit C, a monotone function $g : \{0,1\}^k \to \{0,1\}$ and problem posers $P_n^{(1)}$, $P_{kn}^{(g)}$. Furthermore, Gen takes as input parameters ε, δ, n, k, h, v, and outputs a solver circuit D*

for $P_n^{(1)}$ such that the following holds:
If C is such that

$$\Pr_{\substack{\pi^{(k)}\in\{0,1\}^{kn}\\ \rho\in\{0,1\}^*}}\left[\mathit{Success}^{P_{kn}^{(g)},C}(\pi^{(k)},\rho)=1\right]\geq 16(h+v)\Big(\Pr_{u\leftarrow\mu_\delta^k}[g(u)=1]+\varepsilon\Big)$$

then D is a two phase probabilistic circuit and satisfies almost surely

$$\Pr_{\substack{\pi\in\{0,1\}^{n}\\ \rho\in\{0,1\}^*}}\left[\mathit{Success}^{P_{n}^{(1)},D}(\pi,\rho)=1\right]\geq \delta+\frac{\varepsilon}{6k}.$$

Additionally, D requires oracle access to g, $P_n^{(1)}$, C, hint and verification circuits and asks at most $\frac{6k}{\varepsilon}\log\left(\frac{6k}{\varepsilon}\right) h$ hint queries and one verification query. Finally, $\mathit{Time}(\mathit{Gen}) = \mathit{poly}(k, \frac{1}{\varepsilon}, n, v, h)$ with oracle access to C.

4.1.2 Intuition

TODO: add intuition for hashing

The idea of the algorithm *Gen* is to output a circuit D that solves the input puzzle often. We know that C has high success probability in solving the k-wise direct product of $P^{(1)}$. The algorithm *Gen* tries to find a puzzle such that when C runs with this puzzle fixed on the first position and disregards whether this puzzle is correctly solved then the assumptions of Theorem 4.2 are true for the $(k-1)$-wise direct product. If it was possible to find such a puzzle, then *Gen* could recurse and solve a smaller problem. In the optimistic case we can reach $k=1$, which means that we found a good circuit for solving a single puzzle by just fixing the initial puzzles of C.

Otherwise, when the first position is disregarded then the success probability of C is not substantially better. This is remarkable, as we know that C performs good for the k-wise direct product. It means that the first position is important, in the sense that C solves the puzzle on that position unusually often. Therefore, it is reasonable to construct the circuit D using C by placing the input puzzle of D on that position, and then finding remaining $k-1$ puzzles. The $(k-1)$ remaining puzzles are generated by the circuit D, hence it is possible to check whether they are correctly solved by the circuit C. We know that circuit C has good success probability, and the puzzle on the first position is important. Therefore, if we are able to find a $(k-1)$ puzzles such that the fact whether the k-wise direct product is correctly solved depends on whether the puzzle on the first position is correctly solved then we can assume that C is often correct on this first position.

There are some problems with this approach, first we have to ensure that we can make a decision when the algorithm *Gen* should recurse and when not correctly with high probability. Then, we have to show that it is possible to find a puzzles such that C is often correct on the first position. Finally, we also have to be sure that we do not ask a hint query, on the final verification query to the oracle. To satisfy the last requirement we split the set Q.

4.1.3 Domain partitioning

Let $hash : Q \to \{0, 1, \ldots, 2(h + v) - 1\}$, the idea is to partition Q such that the set of preimages of 0 for $hash$ contains $q \in Q$ on which C is not allowed to ask hint queries, and the first successful verification query (q, y) of C is such that $hash(q) = 0$. Therefore, if C makes a verification query (q, y) such that $hash(q) = 0$, then we know that no hint query is ever asked on this q.

We denote the i-th query of C by q_i if it is a hint query, and by (q_i, y_i) if it is a verification query. We define the following experiment *CanonicalSuccess* in which the set Q is partitioned using a function *hash*. We say that a solver circuit *succeeds* in the experiment *CanonicalSuccess* if it asks a successful verification query (q_j, y_j) such that $hash(q_j) = 0$, and no hint query q_i is asked before (q_j, y_j) such that $hash(q_i) = 0$.

Experiment $\mathit{CanonicalSuccess}^{P,C,hash}(\pi, \rho)$

Oracle: A problem poser P, a solver circuit $C = (C_1, C_2)$ for P,
a function $hash : Q \to \{0, \ldots, 2(h + v) - 1\}$.
Input: Bitstrings $\pi \in \{0, 1\}^n$, $\rho \in \{0, 1\}^*$.
Output: A bit $b \in \{0, 1\}$.

---

run $\langle P(\pi), C_1(\rho)\rangle$
  $(\Gamma_V, \Gamma_H) := \langle P(\pi), C_1(\rho)\rangle_P$
  $x := \langle P(\pi), C_1(\rho)\rangle_{trans}$

run $C_2^{\Gamma_V,\Gamma_H}(x, \rho)$
  if $C_2^{\Gamma_V,\Gamma_H}(x, \rho)$ does not succeed for any verification query **then**
    return 0
  Let (q_j, y_j) be the first verification query such that $\Gamma_V(q_j, y_j) = 1$.

if $(\forall i < j : hash(q_i) \neq 0)$ **and** $(hash(q_j) = 0)$ **then**
  return 1
else
  return 0

We define the *canonical success probability* of a solver circuit C for P with respect to a function *hash* as

$$\Pr_{\pi,\rho}[\mathit{CanonicalSuccess}^{P,C,hash}(\pi,\rho)=1]. \tag{4.2}$$

For fixed *hash* and P a *canonical success* of C for bistrings π, ρ is a situation where $\mathit{CanonicalSuccess}^{P,C,hash}(\pi,\rho)=1$.

We show that if a solver circuit C for P often succeeds in the experiment *Success*, then there exists a function *hash* such that C also often succeeds in the experiment *CanonicalSuccess*.

Lemma 4.3 ***(Success probability in solving DWVP with respect to a function hash.)*** *For fixed P_n let C be a solver for P_n with success probability at least γ, asking at most h hint queries and v verification queries. Let $\mathcal{H}$ be an efficient family of pairwise independent hash functions $Q \to \{0,1,\ldots,2(h+v)-1\}$. There exists a probabilistic algorithm FindHash that takes as input parameters γ, n, h, v, and has oracle access to C and P_n. Furthermore, FindHash runs in time $poly(h,v,\frac{1}{\gamma},n)$, and with high probability outputs a function $hash \in \mathcal{H}$ such that the canonical success probability of C with respect to hash is at least $\frac{\gamma}{16(h+v)}$.*

Proof (4.3). We fix a problem poser P and a solver C for P in the whole proof of Lemma 4.3. For $k,l \in \{1,\ldots,(h+v)\}$ and $\alpha,\beta \in \{0,1,\ldots,2(h+v)-1\}$ by the pairwise independence property, we have

$$\begin{aligned}\forall q_k \neq q_l \in Q : \Pr_{hash\leftarrow\mathcal{H}}[hash(q_k)=\alpha \mid hash(q_l)=\beta] &= \Pr_{hash\leftarrow\mathcal{H}}[hash(q_k)=\alpha] \\ &= \frac{1}{2(h+v)}. \end{aligned} \tag{4.3}$$

We write $\mathcal{P}_{Success}$ to denote a set containing all (π,ρ) for which $\mathit{Success}^{P,C}(\pi,\rho)=1$. Let us fix $(\pi^*,\rho^*)\in\mathcal{P}_{Success}$. We are interested in the probability over a choice of function *hash* of the event $\mathit{CanonicalSuccess}^{P,C,hash}(\pi^*,\rho^*)=1$. Let

(q_j, y_j) denote the first query such that $\Gamma_V(q_j, y_j) = 1$. We have

$$
\begin{aligned}
\Pr_{hash \leftarrow \mathcal{H}} & \Big[\mathit{CanonicalSuccess}^{P,C,hash}(\pi^*, \rho^*) = 1 \Big] \\
&= \Pr_{hash \leftarrow \mathcal{H}}[hash(q_j) = 0 \wedge (\forall i < j : hash(q_i) \neq 0)] \\
&= \Pr_{hash \leftarrow \mathcal{H}}[\forall i < j : hash(q_i) \neq 0 \mid hash(q_j) = 0] \Pr_{hash \leftarrow \mathcal{H}}[hash(q_j) = 0] \\
&\overset{(4.3)}{=} \frac{1}{2(h+v)} \left(1 - \Pr_{hash \leftarrow \mathcal{H}}[\exists i < j : hash(q_i) = 0 \mid hash(q_j) = 0] \right) \\
&\overset{(*)}{\geq} \frac{1}{2(h+v)} \left(1 - \sum_{i<j} \Pr_{hash \leftarrow \mathcal{H}}[hash(q_i) = 0 | \, hash(q_j) = 0] \right) \\
&\overset{(4.3)}{=} \frac{1}{2(h+v)} \left(1 - \sum_{i<j} \Pr_{hash \leftarrow \mathcal{H}}[hash(q_i) = 0] \right) \\
&\overset{(4.3)}{\geq} \frac{1}{4(h+v)},
\end{aligned}
\tag{4.4}
$$

where in $(*)$ we used the union bound. Let us denote the set of those (π, ρ) for which $\mathit{CanonicalSuccess}^{P,C,hash}(\pi, \rho) = 1$ by $\mathcal{P}_{\mathit{Canonical}}$. If for π, ρ the circuit C succeeds canonically, then for the same π, ρ we also have $\mathit{Success}^{P,C}(\pi, \rho) = 1$. Hence, $\mathcal{P}_{\mathit{Canonical}} \subseteq \mathcal{P}_{\mathit{Success}}$, and we conclude

$$
\begin{aligned}
\Pr_{\substack{hash \leftarrow \mathcal{H} \\ \pi, \rho}} & \Big[\mathit{CanonicalSuccess}^{P,C,hash}(\pi, \rho) = 1 \Big] \\
&= \Pr_{\substack{hash \leftarrow \mathcal{H} \\ \pi, \rho}} \Big[\mathit{CanonicalSuccess}^{P,C,hash}(\pi, \rho) = 1 \mid (\pi, \rho) \in \mathcal{P}_{\mathit{Success}} \Big] \Pr_{\pi, \rho}[(\pi, \rho) \in \mathcal{P}_{\mathit{Success}}] \\
&\quad + \Pr_{\substack{hash \leftarrow \mathcal{H} \\ \pi, \rho}} \Big[\mathit{CanonicalSuccess}^{P,C,hash}(\pi, \rho) = 1 \mid (\pi, \rho) \notin \mathcal{P}_{\mathit{Success}} \Big] \Pr_{\pi, \rho}[(\pi, \rho) \notin \mathcal{P}_{\mathit{Success}}] \\
&= \Pr_{\substack{hash \leftarrow \mathcal{H} \\ \pi, \rho}} \Big[\mathit{CanonicalSuccess}^{P,C,hash}(\pi, \rho) = 1 \mid (\pi, \rho) \in \mathcal{P}_{\mathit{Success}} \Big] \Pr_{\pi, \rho}[(\pi, \rho) \in \mathcal{P}_{\mathit{Success}}] \\
&\geq \Pr_{\substack{hash \leftarrow \mathcal{H} \\ \pi, \rho}} \Big[\mathit{CanonicalSuccess}^{P,C,hash}(\pi, \rho) = 1 \mid (\pi, \rho) \in \mathcal{P}_{\mathit{Success}} \Big] \cdot \gamma \\
&= \mathop{\mathbb{E}}_{(\pi, \rho) \in \mathcal{P}_{\mathit{Success}}} \left[\Pr_{hash \leftarrow \mathcal{H}}[\mathit{CanonicalSuccess}^{P,C,hash}(\pi, \rho) = 1] \right] \cdot \gamma \\
&\overset{(4.4)}{\geq} \frac{\gamma}{4(h+v)}.
\end{aligned}
\tag{4.5}
$$

Algorithm $\mathrm{FindHash}^{P,C}(\gamma, n, h, v)$

Oracle: A problem poser P, a solver circuit C for P.
Input: Parameters γ, n. The number of hint queries h and of verification queries v.
Output: A function $hash : Q \to \{0, 1, \ldots, 2(h + v) - 1\}$.

```
for i := 1 to 32n(h+v)^2/γ^2 do:
    hash ← H
    count := 0
    for j := 1 to 32n(h+v)^2/γ^2 do:
        π ← {0,1}^n
        ρ ← {0,1}^*
        if CanonicalSuccess^{P,C,hash}(π, ρ) = 1 then
            count := count + 1
    if count ≥ γ/(12(h+v)) · 32(h+v)^2/γ^2 · n then
        return hash
return ⊥
```

We show that FindHash chooses $hash \in \mathcal{H}$ such that the canonical success probability of C with respect to $hash$ is at least $\frac{\gamma}{16(h+v)}$ almost surely. Let $\mathcal{H}_{Good}$ denote a family of functions $hash \in \mathcal{H}$ for which

$$\Pr_{\pi,\rho}\left[\mathit{CanonicalSuccess}^{P,C,hash}(\pi, \rho) = 1 \right] \geq \frac{\gamma}{8(h+v)}, \tag{4.6}$$

and $\mathcal{H}_{Bad}$ be a family of functions $hash \in \mathcal{H}$ such that

$$\Pr_{\pi,\rho}\left[\mathit{CanonicalSuccess}^{P,C,hash}(\pi, \rho) = 1 \right] \leq \frac{\gamma}{16(h+v)}. \tag{4.7}$$

Let N denote the number of iterations of the inner loop of FindHash. We consider a single iteration of the outer loop of FindHash in which $hash$ is fixed. We define independent, identically distributed, binary random variables $X_1, \ldots, X_N$ such that

$$X_i = \begin{cases} 1 & \text{if in the } i\text{-th iteration of the inner loop } \mathit{count} \text{ is increased} \\ 0 & \text{otherwise.} \end{cases}$$

We now turn to the case when $hash \in \mathcal{H}_{Bad}$ and show that it is unlikely that $hash$ is returned by FindHash. From (4.7) it follows that $\mathbb{E}_{\pi,\rho}[X_i] \leq \frac{\gamma}{16(h+v)}$. Therefore, for any fixed $hash \in \mathcal{H}_{Bad}$ using the Chernoff bound we get

$$\begin{aligned} \Pr_{\pi,\rho}\left[\frac{1}{N} \sum_{i=1}^{N} X_i \geq \frac{\gamma}{12(h+v)} \right] &\leq \Pr_{\pi,\rho}\left[\frac{1}{N} \sum_{i=1}^{N} X_i \geq \left(1 + \frac{1}{3}\right) \mathbb{E}[X_i] \right] \\ &\leq e^{-\frac{\gamma}{16(h+v)} N/27} \leq e^{-\frac{2}{27} \frac{(h+v)}{\gamma} n} \overset{(*)}{\leq} e^{-\frac{2}{27} n}, \end{aligned}$$

where in $(*)$ we used the trivial facts that $h+v \geq 1$ and $\gamma \leq 1$. The probability that $hash \in \mathcal{H}_{Good}$, when picked, is not returned amounts

$$\Pr_{\pi,\rho}\left[\frac{1}{N}\sum_{i=1}^{N} X_i \leq \frac{\gamma}{12(h+v)}\right] \leq \Pr_{\pi,\rho}\left[\frac{1}{N}\sum_{i=1}^{N} X_i \leq (1-\frac{1}{3})\mathbb{E}[X_i]\right]$$
$$\leq e^{-\frac{\gamma}{8(h+v)}N/18} \leq e^{-\frac{2}{9}\frac{(h+v)}{\gamma}n} \overset{(*)}{\leq} e^{-\frac{2}{9}n},$$

where we once more used the Chernoff bound. We now show that the probability of picking $hash \in \mathcal{H}_{Good}$ is at least $\frac{\gamma}{8(h+v)}$. To obtain a contradiction suppose that

$$\Pr_{hash \leftarrow \mathcal{H}}[hash \in \mathcal{H}_{Good}] < \frac{\gamma}{8(h+v)}. \tag{4.8}$$

From this it follows that we can bound probability of canonical success as follows

$$\Pr_{\substack{hash \leftarrow \mathcal{H}\\ \pi,\rho}}[CanonicalSuccess^{P,C,hash}(\pi,\rho)=1]$$
$$= \Pr_{\substack{hash \leftarrow \mathcal{H}\\ \pi,\rho}}[CanonicalSuccess^{P,C,hash}(\pi,\rho)=1 \mid hash \in \mathcal{H}_{Good}] \Pr_{hash \leftarrow \mathcal{H}}[hash \in \mathcal{H}_{Good}]$$
$$+ \Pr_{\substack{hash \leftarrow \mathcal{H}\\ \pi,\rho}}[CanonicalSuccess^{P,C,hash}(\pi,\rho)=1 \mid hash \notin \mathcal{H}_{Good}] \Pr_{hash \leftarrow \mathcal{H}}[hash \notin \mathcal{H}_{Good}]$$
$$\leq \Pr_{hash \leftarrow \mathcal{H}}[hash \in \mathcal{H}_{Good}] + \Pr_{\substack{hash \leftarrow \mathcal{H}\\ \pi,\rho}}[CanonicalSuccess^{P,C,hash}(\pi,\rho)=1 \mid hash \notin \mathcal{H}_{Good}]$$
$$\overset{\substack{(4.6)\\(4.8)}}{<} \frac{\gamma}{8(h+v)} + \frac{\gamma}{8(h+v)} = \frac{\gamma}{4(h+v)},$$

which contradicts (4.5). Therefore, we conclude that the probability of choosing a $hash \in \mathcal{H}_{Good}$ amounts at least $\frac{\gamma}{8(h+v)}$.

We show that FindHash picks in one of its iteration $hash \in \mathcal{H}_{Good}$ almost surely. Let K be the number of iterations of the outer loop of FindHash and Y_i be a random variable for the event that in the i-th iteration of the outer loop $hash \notin \mathcal{H}_{Good}$ is picked. We use $\Pr_{hash \leftarrow \mathcal{H}}[hash \in \mathcal{H}_{Good}] \geq \frac{\gamma}{8(h+v)}$ and $K \leq \frac{32(h+v)^2}{\gamma^2}n$, and conclude

$$\Pr_{hash \leftarrow \mathcal{H}}\Big[\bigcap_{1 \leq i \leq K} Y_i\Big] \leq \left(1 - \frac{\gamma}{8(h+v)}\right)^{\frac{32(h+v)^2}{\gamma^2}n} \leq e^{-\frac{\gamma}{8(h+v)}\frac{32(h+v)^2}{\gamma^2}n}$$
$$\leq e^{-\frac{4(h+v)}{\gamma}n} \leq e^{-n}.$$

It is clear that running time of FindHash is $poly(n,h,v,\gamma)$ with oracle access. This finishes the proof of Lemma 4.3. $\square$

4.1.4 Amplification proof for partitioned domain

Let $C = (C_1, C_2)$ be a solver circuit for a dynamic weakly verifiable puzzle as in definition 3.1. We write $C_2^{(\cdot,\cdot)}$ to emphasize that C_2 does not obtain direct access to hint and verification circuits. Instead, whenever C_2 ask hint or verification queries, then it is answered explicitly as in the following code excerpt of the circuit $\widetilde{C}_2$.

Circuit $\widetilde{C}_2^{\Gamma_H, C_2, hash}(x, \rho)$

Oracle: A hint circuit Γ_H, a circuit C_2,
a function $hash : Q \to \{0, 1, \ldots, 2(h+v) - 1\}$.
Input: Bitstrings $x \in \{0,1\}^*$, $\rho \in \{0,1\}^*$.
Output: A pair (q, y).

---

run $C_2^{(\cdot,\cdot)}(x, \rho)$
    if $C_2^{(\cdot,\cdot)}(x, \rho)$ asks a hint query on q **then**
        if $hash(q) = 0$ **then**
            return $\perp$
        else
            answer the query of $C_2^{(\cdot,\cdot)}(x, \rho)$ using $\Gamma_H(q)$

    if $C_2^{(\cdot,\cdot)}(x, \rho)$ asks a verification query (q, y) **then**
        if $hash(q) = 0$ **then**
            return (q, y)
        else
            answer the verification query of $C_2^{(\cdot,\cdot)}(x, \rho)$ with 0
return $\perp$

Given $C = (C_1, C_2)$ we define the circuit $\widetilde{C} = (C_1, \widetilde{C}_2)$. Every hint query q asked by $\widetilde{C}$ is such that $hash(q) \neq 0$. Furthermore, $\widetilde{C}$ asks no verification queries. Instead, it returns (q, y) such that $hash(q) = 0$ or $\perp$.

For fixed π, ρ, and $hash$ we say that the circuit $\widetilde{C}$ *succeeds* if for $x := \langle P(\pi), C_1(\rho)\rangle_{trans}$, $(\Gamma_V, \Gamma_H) := \langle P(\pi), C_1(\rho)\rangle_P$, we have

$$\Gamma_V(\widetilde{C}_2^{\Gamma_H, C_2, hash}(x, \rho)) = 1.$$

Lemma 4.4 *For fixed P, C, and hash the following statement is true*

$$\Pr_{\pi,\rho}[CanonicalSuccess^{P,C,hash}(\pi, \rho) = 1] \leq \Pr_{\substack{\pi,\rho \\ x := \langle P(\pi), C_1(\rho)\rangle_{trans} \\ (\Gamma_V, \Gamma_H) := \langle P(\pi), C_1(\rho)\rangle_P}}[\Gamma_V(\widetilde{C}_2^{\Gamma_H, C_2, hash}(x, \rho)) = 1]$$

Proof. If for fixed π, ρ, and *hash* the circuit C succeeds canonically, then for the same π, ρ, and *hash* also $\widetilde{C}$ succeeds. Using this observation, we conclude that

$$\begin{aligned}
&\Pr_{\pi,\rho}\left[\mathit{CanonicalSuccess}^{P,C,hash}(\pi,\rho)=1\right] \\
&\quad\leq \mathop{\mathbb{E}}_{\substack{\pi,\rho \\ x:=\langle P(\pi),C_1(\rho)\rangle_{trans} \\ (\Gamma_V,\Gamma_H):=\langle P(\pi),C_1(\rho)\rangle_P}}\left[\Gamma_V(\widetilde{C}_2^{\Gamma_H,C_2,hash}(x,\rho))=1\right] \\
&\quad= \Pr_{\substack{\pi,\rho \\ x:=\langle P(\pi),C_1(\rho)\rangle_{trans} \\ (\Gamma_V,\Gamma_H):=\langle P(\pi),C_1(\rho)\rangle_P}}\left[\Gamma_V(\widetilde{C}_2^{\Gamma_H,C_2,hash}(x,\rho))=1\right] \qquad \square
\end{aligned}$$

Lemma 4.5 ***(Security amplification for dynamic weakly verifiable puzzles with respect to hash.)*** *Let $g:\{0,1\}^k\to\{0,1\}$ be a monotone function and $P_n^{(1)}$ a fixed problem poser as in Definition 3.1 and $\widetilde{C}:=(C_1,\widetilde{C}_2)$ a circuit with oracle access to a function $hash: Q\to\{0,1,\dots,2(h+v-1)\}$ and a solver circuit $C:=(C_1,C_2)$ for $P_{kn}^{(g)}$ which asks at most h hint queries and v verification queries. There exists an algorithm Gen that takes as input parameters ε, δ, n, k, has oracle access to $P_n^{(1)}$, $\widetilde{C}$, hash, g, and outputs a circuit $D:=(D_1,D_2)$ such that the following holds:*
If $\widetilde{C}$ is such that

$$\Pr_{\substack{\pi^{(k)}\in\{0,1\}^{kn},\rho\in\{0,1\}^* \\ x:=\langle P^{(g)}(\pi^{(k)}),C_1(\rho)\rangle_{trans} \\ (\Gamma_H^{(k)},\Gamma_V^{(g)}):=\langle P^{(g)}(\pi^{(k)}),C_1(\rho)\rangle_{P^{(g)}}}}[\Gamma_V^{(g)}(\widetilde{C}_2^{\Gamma_H^{(k)},C_2,hash}(x,\rho))=1]\geq\Pr_{u\leftarrow\mu_\delta^k}[g(u)=1]+\varepsilon,$$

then D satisfies almost surely

$$\Pr_{\substack{\pi\in\{0,1\}^n,\rho\in\{0,1\}^* \\ x:=\langle P^{(1)}(\pi),D_1^{\widetilde{C}}(\rho)\rangle_{trans} \\ (\Gamma_H,\Gamma_V):=\langle P^{(1)}(\pi),D_1^{\widetilde{C}}(\rho)\rangle_{P^{(1)}}}}[\Gamma_V(D_2^{P^{(1)},\widetilde{C},hash,g,\Gamma_H}(x,\rho))=1]\geq\delta+\frac{\varepsilon}{6k}.$$

Furthermore, D asks at most $\frac{6k}{\varepsilon}\log\left(\frac{6k}{\varepsilon}\right)h$ hint queries and no verification queries. Finally, $Time(Gen)=poly(k,\frac{1}{\varepsilon},n)$ with oracle access to $\widetilde{C}$.

Before we give the proof of Lemma 4.5 we define additional algorithms. First, we are interested in the probability that for $u\leftarrow\mu_\delta^k$ and a bit b we have $g(b,u_2,\dots,u_k)=1$. The estimate of this probability is calculated by EstimateFunctionProbability.

Algorithm EstimateFunctionProbability$^g(b, k, \varepsilon, \delta, n)$

Oracle: A function $g : \{0,1\}^k \to \{0,1\}$.
Input: A bit $b \in \{0,1\}$, parameters k, ε, δ, n.
Output: An estimate $\widetilde{g}_b$ of $\Pr_{u \leftarrow \mu_\delta^k}[g(b, u_2, \ldots, u_k) = 1]$.

---

for $i := 1$ **to** $N := \frac{64k^2}{\varepsilon^2} n$ **do:**
    $u \leftarrow \mu_\delta^k$
    $g_i := g(b, u_2, \ldots, u_k)$
return $\frac{1}{N} \sum_{i=1}^N g_i$

Lemma 4.6 *The algorithm EstimateFunctionProbability$^g(b, k, \varepsilon, \delta, n)$ outputs an estimate $\widetilde{g}_b$ such that $|\widetilde{g}_b - \Pr_{u \leftarrow \mu_\delta^k}[g(b, u_2, \ldots, u_k) = 1]\,| \leq \frac{\varepsilon}{8k}$ almost surely.*

Proof. We fix the notation as in the algorithm EstimateFunctionProbability. Let us define independent, identically distributed binary random variables $K_1, K_2, \ldots, K_N$ such that for each $1 \leq i \leq N$ the random variable K_i takes value g_i. We use the Chernoff bound to obtain

$$\Pr\left[\left|\widetilde{g}_b - \Pr_{u \leftarrow \mu_\delta^k}[g(b, u_2, \ldots, u_k) = 1]\right| \geq \frac{\varepsilon}{8k}\right]$$
$$= \Pr\left[\left|\left(\frac{1}{N}\sum_{i=1}^N K_i\right) - \mathbb{E}_{u \leftarrow \mu_\delta^k}[g(b, u_2, \ldots, u_k)]\right| \geq \frac{\varepsilon}{8k}\right] \leq 2 \cdot e^{-n/3}. \square$$

The algorithm EvalutePuzzles$^{P^{(1)}, \widetilde{C}, hash}(\pi^{(k)}, \rho, n, k)$ evaluates which of the k puzzles of the k-wise direct product defined by $P^{(g)}$ are solved successfully by $\widetilde{C}(\rho) := (C_1, \widetilde{C}_2)(\rho)$. To decide whether the i-th puzzle of the k-wise direct product is solved successfully we need to gain access to the verification circuit for the puzzle generated in the i-th round of the interaction between $P^{(g)}$ and $\widetilde{C}$. Therefore, the algorithm EvalutePuzzles runs k times $P^{(1)}$ to simulate the interaction with $C_1(\rho)$ each time with a fresh random bitstring $\pi_i \in \{0,1\}^n$ where $1 \leq i \leq k$.

Let us introduce some additional notation. We denote by $\langle P^{(1)}(\pi_i), C_1(\rho)\rangle^i$ the execution of the i-th round of the sequential interaction. We use $\langle P^{(1)}(\pi_i), C_1(\rho)\rangle^i_{P^{(1)}}$ to denote the output of $P^{(1)}(\pi_i)$ in the i-th round. Finally, we write $\langle P^{(1)}(\pi_i), C_1(\rho)\rangle^i_{trans}$ to denote the transcript of communication in the i-th round. We note that the i-th round of the interaction between $P^{(1)}$ and C_1 is well defined only if all previous rounds have been executed before.

To make the notation easier in the code excerpts of circuits C_2, D_2 and EvalutePuzzles we omit superscripts of some oracles. Exemplary, we write $\widetilde{C}_2^{\Gamma_H^{(k)}, hash}$ instead of $\widetilde{C}_2^{\Gamma_H^{(k)}, C, hash}$ where the superscript of the oracle circuit C

is omitted. We make sure that it is clear from the context which oracles are used.

Algorithm EvaluatePuzzles$^{P^{(1)},\widetilde{C},hash}(\pi^{(k)},\rho,n,k)$

Oracle: A problem poser $P^{(1)}$, a solver circuit $\widetilde{C}=(C_1,\widetilde{C}_2)$ for $P^{(g)}$, a function $hash : Q \to \{0,1,\dots,2(h+v)-1\}$.
Input: Bitstrings $\pi^{(k)} \in \{0,1\}^{kn}$, $\rho \in \{0,1\}^*$, parameters n, k.
Output: A tuple $(c_1,\dots,c_k) \in \{0,1\}^k$.

---

for $i := 1$ **to** k **do:** *//simulate k rounds of interaction*
  $(\Gamma_V^i,\Gamma_H^i) := \langle P^{(1)}(\pi_i), C_1(\rho)\rangle^i_{P^{(1)}}$
  $x_i := \langle P^{(1)}(\pi_i), C_1(\rho)\rangle^i_{trans}$
$x := (x_1,\dots,x_k)$
$\Gamma_H^{(k)} := (\Gamma_H^1,\dots,\Gamma_H^k)$
$(q,y_1,\dots,y_k) := \widetilde{C}_2^{\Gamma_H^{(k)},hash}(x,\rho)$
if $(q,y_1,\dots,y_k)=\perp$ **then**
  return $(0,\dots,0)$
$(c_1,\dots,c_k) := (\Gamma_V^1(q,y_1),\dots,\Gamma_V^k(q,y_k))$
return $(c_1,\dots,c_k)$

All puzzles used by EvalutePuzzles are generated internally thus the algorithm has access to hint circuit, and can answer itself all queries of $\widetilde{C}_2$.

We are interested in the success probability of $\widetilde{C}$ with the bitstring π_1 fixed to π^* where the fact whether $\widetilde{C}$ succeeds in solving the first puzzle defined by $P^{(1)}(\pi_1)$ is neglected, and instead a bit b is used. More formally, we define the surplus $S_{\pi^*,b}$ as

$$S_{\pi^*,b} = \Pr_{\pi^{(k)},\rho}[g(b,c_2,\dots,c_k)=1 \mid \pi_1=\pi^*] - \Pr_{u\leftarrow\mu_\delta^k}[g(b,u_2,\dots,u_k)=1], \tag{4.9}$$

where $(c_2,c_3,\dots,c_k)$ is obtained as in EvalutePuzzles.

The algorithm EstimateSurplus returns an estimate $\widetilde{S}_{\pi^*,b}$ for $S_{\pi^*,b}$.

Algorithm EstimateSurplus$^{P^{(1)},\widetilde{C},g,hash}(\pi^*,b,k,\varepsilon,\delta,n)$

Oracle: A problem poser $P^{(1)}$, a circuit $\widetilde{C}$ for $P^{(g)}$, functions $g:\{0,1\}^k \to \{0,1\}$ and $hash : Q \to \{0,1,\dots,2(h+v)-1\}$.
Input: A bistring $\pi^* \in \{0,1\}^n$, a bit $b \in \{0,1\}$, parameters k, ε, δ, n.
Output: An estimate $\widetilde{S}_{\pi^*,b}$ for $S_{\pi^*,b}$.

---

for $i := 1$ **to** $N := \frac{64k^2}{\varepsilon^2} n$ **do:**
$(\pi_2, \ldots, \pi_k) \xleftarrow{\$} \{0,1\}^{(k-1)n}$
$\rho \xleftarrow{\$} \{0,1\}^*$
$(c_1, \ldots, c_k) := \text{EvalutePuzzles}^{P^{(1)}, \widetilde{C}, hash}((\pi^*, \pi_2, \ldots, \pi_k), \rho, n, k)$
$\widetilde{s}^i_{\pi^*,b} := g(b, c_2, \ldots, c_k)$
$\widetilde{g}_b := \text{EstimateFunctionProbability}^g(b, k, \varepsilon, \delta, n)$
return $\left(\frac{1}{N}\sum_{i=1}^{N} \widetilde{s}^i_{\pi^*,b}\right) - \widetilde{g}_b$

Lemma 4.7 *The estimate $\widetilde{S}_{\pi^*,b}$ returned by EstimateSurplus differs from $S_{\pi^*,b}$ by at most $\frac{\varepsilon}{4k}$ almost surely.*

Proof. We use the union bound and similar argument as in Lemma 4.6 which yields that $\frac{1}{N}\sum_{i=1}^{N} \widetilde{s}^i_{\pi^*,b}$ differs from $\mathbb{E}[g(b, c_2, \ldots, c_k)]$ by at most $\frac{\varepsilon}{8k}$ almost surely. Together, with Lemma 4.6 we conclude that the surplus estimate returned by EstimateSurplus differs from $S_{\pi^*,b}$ by at most $\frac{\varepsilon}{4k}$ almost surely.□

We define the following solver circuit $C' = (C'_1, C'_2)$ for the $(k-1)$-wise direct product of $P^{(1)}$.

Circuit $C_1'^{\widetilde{C}, P^{(1)}}(\rho)$

Oracle: A solver circuit $\widetilde{C} = (C_1, \widetilde{C}_2)$ for $P^{(g)}$, a poser $P^{(1)}$.
Input: A bitstring $\rho \in \{0,1\}^*$.
Hard-coded: A bitstring $\pi^* \in \{0,1\}^n$.

Simulate $\langle P^{(1)}(\pi^*), C_1(\rho)\rangle^1$
Use $C_1(\rho)$ for the remaining $k-1$ rounds of interaction.

Circuit $\widetilde{C}_2'^{\Gamma_H^{(k-1)}, \widetilde{C}, hash}(x^{(k-1)}, \rho)$

Oracle: A hint oracle $\Gamma_H^{(k-1)} := (\Gamma_H^2, \ldots, \Gamma_H^k)$,
a solver circuit $\widetilde{C} = (C_1, \widetilde{C}_2)$ for $P^{(g)}$,
a function $hash : Q \to \{0, 1, \ldots, 2(h+v) - 1\}$.
Input: A transcript of $k-1$ rounds of interaction
$x^{(k-1)} := (x_2, \ldots, x_k) \in \{0,1\}^*$, a bitstring $\rho \in \{0,1\}^*$
Hard-coded: A bitstring $\pi^* \in \{0,1\}^n$

Simulate $\langle P^{(1)}(\pi^*), C_1(\rho)\rangle^1$
$(\Gamma_H^*, \Gamma_V^*) := \langle P^{(1)}(\pi^*), C_1(\rho)\rangle^1_{P^{(1)}}$
$x^* := \langle P^{(1)}(\pi^*), C_1(\rho)\rangle^1_{trans}$

$\Gamma_H^{(k)} := (\Gamma_H^*, \Gamma_H^2, \ldots, \Gamma_H^k)$
$x^{(k)} := (x^*, x_2, \ldots, x_k)$
$(q, y_1, \ldots, y_k) := \widetilde{C}_2^{\Gamma_H^{(k)}, hash}(x^{(k)}, \rho)$
return $(q, y_2, \ldots, y_k)$

We are ready to define the solver circuit $D = (D_1, D_2)$ for $P^{(1)}$ and the algorithm Gen.

Circuit $D_1^{\widetilde{C}}(r)$

Oracle: A solver circuit $\widetilde{C} = (C_1, \widetilde{C}_2)$ for $P^{(g)}$.
Input: A pair $r := (\rho, \sigma)$ where $\rho \in \{0,1\}^*$ and $\sigma \in \{0,1\}^*$.

Interact with the problem poser $\langle P^{(1)}, C_1(\rho)\rangle^1$.
Let $x^* := \langle P^{(1)}, C_1(\rho)\rangle^1_{trans}$.

Circuit $D_2^{P^{(1)}, \widetilde{C}, hash, g, \Gamma_H}(x^*, r)$

Oracle: A poser $P^{(1)}$, a solver circuit $\widetilde{C} = (C_1, \widetilde{C}_2)$ for $P^{(g)}$,
functions $hash : Q \to \{0, 1, \ldots, 2(h+v)-1\}$, $g : \{0,1\}^k \to \{0,1\}$,
a hint circuit Γ_H for $P^{(1)}$.
Input: A communiation transcript $x^* \in \{0,1\}^*$, a bitstring $r := (\rho, \sigma)$
where $\rho \in \{0,1\}^*$ and $\sigma \in \{0,1\}^*$
Output: A pair (q, y^*).

---

for at most $\frac{6k}{\varepsilon}\log(\frac{6k}{\varepsilon})$ iterations **do:**
  $(\pi_2, \ldots, \pi_k) \leftarrow$ read next $(k-1) \cdot n$ bits from σ
  Use x^* to simulate the first round of interiaction of $C_1(\rho)$ with the problem poser $P^{(1)}$
  for $i := 2$ **to** k **do:**
    run $\langle P^{(1)}(\pi_i), C_1(\rho)\rangle^i$
      $(\Gamma_V^i, \Gamma_H^i) := \langle P^{(1)}(\pi_i), C_1(\rho)\rangle^i_{P^{(1)}}$
      $x_i := \langle P^{(1)}(\pi_i), C_1(\rho)\rangle^i_{trans}$
  $\Gamma_H^{(k)}(q) := (\Gamma_H(q), \Gamma_H^2(q), \ldots, \Gamma_H^k(q))$
  $(q, y^*, y_2, \ldots, y_k) := \widetilde{C}_2^{\Gamma_H^{(k)}, hash}((x^*, x_2, \ldots, x_k), \rho)$
  $(c_2, \ldots, c_k) := (\Gamma_V^2(q, y_2), \ldots, \Gamma_V^k(q, y_k))$
  if $g(1, c_2, \ldots, c_k) = 1$ **and** $g(0, c_2, \ldots, c_k) = 0$ **then**
    return (q, y^*)
return $\perp$

Algorithm $\mathrm{Gen}^{P^{(1)},\widetilde{C},g,hash}(\varepsilon,\delta,n,k)$

Oracle: A poser $P^{(1)}$, a solver circuit $\widetilde{C}$ for $P^{(g)}$, functions $g:\{0,1\}^k \to \{0,1\}$,
$hash : Q \to \{0,1,\dots,2(h+v)-1\}$.
Input: Parameters ε, δ, n, k.
Output: A circuit D.

---

for $i := 1$ **to** $\frac{6k}{\varepsilon}n$ **do:**
$\pi^* \xleftarrow{\$} \{0,1\}^n$
$\widetilde{S}_{\pi^*,0} := \mathrm{EstimateSurplus}^{P^{(1)},\widetilde{C},g,hash}(\pi^*,0,k,\varepsilon,\delta,n)$
$\widetilde{S}_{\pi^*,1} := \mathrm{EstimateSurplus}^{P^{(1)},\widetilde{C},g,hash}(\pi^*,1,k,\varepsilon,\delta,n)$
if $\exists b \in \{0,1\} : \widetilde{S}_{\pi^*,b} \geq (1-\frac{3}{4k})\varepsilon$ **then**
Let C_1' have oracle access to $\widetilde{C}$, and have hard-coded π^*
Let $\widetilde{C}_2'$ have oracle access to $\widetilde{C}$, and have hard-coded π^*.
$\widetilde{C}' := (C_1', \widetilde{C}_2')$
$g'(b_2,\dots,b_k) := g(b,b_2,\dots,b_k)$
return $Gen^{P^{(1)},\widetilde{C}',g',hash}(\varepsilon,\delta,n,k-1)$
// all estimates are lower than $(1-\frac{3}{4k})\varepsilon$
return $D^{P^{(1)},\widetilde{C},hash,g}$

Proof (Lemma 4.5). First let us consider the case where $k=1$. The function $g:\{0,1\}\to\{0,1\}$ is either the identity or a constant function. If g is the identity function, then the circuit D returned by Gen directly uses $\widetilde{C}$ to find a solution. From the assumptions of Lemma 4.5 it follows that $\widetilde{C}$ succeeds with probability at least $\delta+\varepsilon$. Hence, D trivially satisfies the statement of Lemma 4.5. In the latter case g is a constant function, and the statement is vacuously true.

For the general case, we consider two possibilities. Either Gen in one of the iterations finds an estimate $\widetilde{S}_{\pi,b} \geq (1-\frac{3}{4k}\varepsilon)$ or it fails and returns the circuit D.

In the former case we define a new monotone function $g'(b_2,\dots,b_k) := g(b,b_2,\dots,b_k)$ and a new circuit $\widetilde{C}' = (C_1', \widetilde{C}_2')$ with oracle access to $\widetilde{C} := (C_1, \widetilde{C}_2)$. By Lemma 4.7 it follows that $S_{\pi^*,b} \geq \widetilde{S}_{\pi^*,b} - \frac{\varepsilon}{4k} \geq (1-\frac{1}{k})\varepsilon$ almost surely. Therefore, the circuit $\widetilde{C}'$ succeeds in solving the $(k-1)$-wise direct product of puzzles with probability at least $\Pr_{u\leftarrow\mu_\delta^{(k-1)}}[g'(u_1,\dots,u_{k-1})] + (1-\frac{1}{k})\varepsilon$. In this case $\widetilde{C}'$ satisfies the conditions of Lemma 4.5 for the $(k-1)$-wise direct product of puzzles. Therefore, the recursive call to Gen with access to g' and $\widetilde{C}$ returns a circuit $D = (D_1, D_2)$ that with high probability satisfies

$$\Pr_{\substack{\pi,\rho \\ x:=\langle P^{(1)}(\pi), D_1^{\widetilde{C}}(\rho)\rangle_{trans} \\ (\Gamma_H,\Gamma_V):=\langle P^{(1)}(\pi), D_1^{\widetilde{C}}(\rho)\rangle_{P(1)}}}[\Gamma_V(D_2^{P^{(1)},\widetilde{C},hash,g,\Gamma_H}(x,\rho)) = 1] \geq \delta + \left(1 - \frac{1}{k}\right)\frac{\varepsilon}{6(k-1)} = \delta + \frac{\varepsilon}{6k}. \quad (4.10)$$

The only point remaining concerns the behavior of Gen when none of the estimates is greater than $(1 - \frac{3}{4k})\varepsilon$. Assume that...
TODO : Give more intuitoin (and the correct one with references to the equations)
Intuitively this means that $\widetilde{C}$ does not succeed on the remaining $k-1$ puzzles with much higher probability than an algorithm that correctly solves each puzzle with probability δ. However, from the assumptions of Lemma 4.5 we know that on all k puzzles the success probability of $\widetilde{C}$ is higher. Therefore, it is likely that the first puzzle is correctly solved unusually often. It remains to prove that this intuition is indeed correct.

We fix the notation used in the code excerpt of the circuit D_2. We consider a single iteration of the outer loop of D_2, in which values $\pi_2, \ldots, \pi_k$ are fixed. Additionally, we write π_1 to denote the randomness of the problem poser and define $c_1 := \Gamma_V(q, y_1)$, where Γ_V is the verification circuit generated by $P^{(1)}(\pi_1)$ in the first phase when interacting with $D_1(r)$. Finally, we introduce the additional notation $\mathcal{G}_b := \{(b_1, b_2, \ldots, b_k) : g(b, b_2, \ldots, b_k) = 1\}$ and $c = (c_1, c_2, \ldots, c_k)$. Using the new notation the following holds

$$\begin{aligned}
\Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_b] &= \Pr_{u \leftarrow \mu_\delta^k}[g(b, u_2, \ldots, u_k) = 1] \\
\Pr_{\pi^{(k)},\rho}[c \in \mathcal{G}_b] &= \Pr_{\pi^{(k)},\rho}[g(b, c_2, \ldots, c_k) = 1]. \quad (4.11)
\end{aligned}$$

We fix the randomness π_1 of the problem poser $P^{(1)}$ to π^* and use (4.9), (4.11) to obtain

$$\begin{aligned}
&\Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_1] - \Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_0] \\
&= \Pr_{\pi^{(k)},\rho}[c \in \mathcal{G}_1 \mid \pi_1 = \pi^*] - \Pr_{\pi^{(k)},\rho}[c \in \mathcal{G}_0 \mid \pi_1 = \pi^*] - (S_{\pi^*,1} - S_{\pi^*,0}) \quad (4.12)
\end{aligned}$$

We know that the function g is monotone, hence it holds $\mathcal{G}_0 \subseteq \mathcal{G}_1$, and we write (4.12) as

$$\Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_1 \setminus \mathcal{G}_0] = \Pr_{\pi^{(k)},\rho}[c \in \mathcal{G}_1 \setminus \mathcal{G}_0 \mid \pi_1 = \pi^*] - (S_{\pi^*,1} - S_{\pi^*,0}). \quad (4.13)$$

Still having $\pi_1 = \pi^*$ fixed we multiply both sides of (4.13) by

$$\Pr_{\substack{r \\ x^*:=\langle P^{(1)}(\pi^*), D_1(r)\rangle_{trans} \\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*), D_1(r)\rangle_{P(1)}}}[\Gamma_V(D_2(x^*, r)) = 1] / \Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_1 \setminus \mathcal{G}_0],$$

which yields

$$
\begin{aligned}
&\Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(D_2(x^*,r))=1]\\
&\quad= \Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(D_2(x^*,r))=1]\Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_1\setminus\mathcal{G}_0\mid\pi_1=\pi^*]\frac{1}{\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]}\\
&\quad- \Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(D_2(x^*,r))=1](S_{\pi^*,1}-S_{\pi^*,0})\frac{1}{\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]}.
\end{aligned}
\tag{4.14}
$$

We analyze the first summand of (4.14). First, we have

$$
\begin{aligned}
&\Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(D_2(x^*,r))=1]\\
&\quad= \Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(D_2(x^*,r))=1|D_2(x^*,r)\neq\perp]\Pr_{\substack{r\\ x^*=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}}}[D_2(x^*,r)\neq\perp]\\
&\quad\overset{(*)}{=} \Pr_{\pi^{(k)},\rho}[c_1=1\mid c\in\mathcal{G}_1\setminus\mathcal{G}_0,\pi_1=\pi^*]\Pr_{\substack{r\\ x^*=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}}}[D_2(x^*,r)\neq\perp],
\end{aligned}
\tag{4.15}
$$

where in $(*)$ we use the observation that the event $D_2(x^*,r)\neq\perp$ happens if and only if the circuit $D_2(x^*,r)$ finds $\pi^{(k)}$ such that $c\in\mathcal{G}_1\setminus\mathcal{G}_0$. Furthermore, conditioned on $c\in\mathcal{G}_1\setminus\mathcal{G}_0$ we have that $\Gamma_V(D_2(x^*,r))=1$ occurs if and only if $c_1=1$. Inserting (4.15) to the numerator of the first summand of (4.14) yields

$$
\begin{aligned}
&\Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(D_2(x^*,r))=1]\Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_1\setminus\mathcal{G}_0\mid\pi_1=\pi^*]\\
&\quad= \Pr_{\substack{r\\ x^*=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}}}[D_2(x^*,r)\neq\perp]\Pr_{\pi^{(k)},\rho}[c_1=1\mid c\in\mathcal{G}_1\setminus\mathcal{G}_0,\pi_1=\pi^*]\Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_1\setminus\mathcal{G}_0\mid\pi_1=\pi^*].
\end{aligned}
\tag{4.16}
$$

We consider the following two cases. If $\Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_1\setminus\mathcal{G}_0\mid\pi_1=\pi^*]\leq\frac{\varepsilon}{6k}$ then

$$
\Pr_{\pi^{(k)},\rho}[c_1=1\mid c\in\mathcal{G}_1\setminus\mathcal{G}_0,\pi_1=\pi^*]\Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_1\setminus\mathcal{G}_0\mid\pi_1=\pi^*]\leq\frac{\varepsilon}{6k}. \tag{4.17}
$$

When $\Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_1\setminus\mathcal{G}_0\mid\pi_1=\pi^*]>\frac{\varepsilon}{6k}$ the circuit D_2 outputs $\perp$ if and only if it fails in all $\frac{6k}{\varepsilon}\log(\frac{6k}{\varepsilon})$ iterations to find $\pi^{(k)}$ such that $c\in\mathcal{G}_1\setminus\mathcal{G}_0$ which happens with probability

$$\Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}}}[D_2(x^*,r)=\bot] \leq (1-\frac{\varepsilon}{6k})^{\frac{6k}{\varepsilon}\log(\frac{6k}{\varepsilon})} \leq \frac{\varepsilon}{6k}. \tag{4.18}$$

We conclude that in both cases by (4.17) and (4.18) we have

$$\begin{aligned}
&\Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}}}[D_2(x^*,r)\neq\bot] \Pr_{\pi^{(k)},\rho}[c_1=1 \mid c\in\mathcal{G}_1\setminus\mathcal{G}_0, \pi_1=\pi^*] \Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_1\setminus\mathcal{G}_0 \mid \pi_1=\pi^*]\\
&\geq \Pr_{\pi^{(k)},\rho}[c_1=1 \mid c\in\mathcal{G}_1\setminus\mathcal{G}_0, \pi_1=\pi^*] \Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_1\setminus\mathcal{G}_0 \mid \pi_1=\pi^*] - \frac{\varepsilon}{6k}\\
&= \Pr_{\pi^{(k)},\rho}[c_1=1 \wedge c\in\mathcal{G}_1\setminus\mathcal{G}_0 \mid \pi_1=\pi^*] - \frac{\varepsilon}{6k}\\
&= \Pr_{\pi^{(k)},\rho}[g(c)=1 \mid \pi_1=\pi^*] - \Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_0 \mid \pi_1=\pi^*] - \frac{\varepsilon}{6k}\\
&\overset{(4.9)}{=} \Pr_{\pi^{(k)},\rho}[g(c)=1 \mid \pi_1=\pi^*] - \Pr_{u\leftarrow\mu_\delta^{(k)}}[u\in\mathcal{G}_0] - S_{\pi^*,0} - \frac{\varepsilon}{6k}.
\end{aligned} \tag{4.19}$$

We insert (4.19) to (4.14) and calculate the expected value of over π^* which yields

$$\begin{aligned}
\Pr_{\substack{\pi,r\\ x:=\langle P^{(1)}(\pi),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(D_2(x,r))=1] &\geq \mathbb{E}_{\pi^*}\left[\frac{\Pr_{\pi^{(k)},\rho}[g(c)=1|\pi_1=\pi^*] - \Pr_{u\leftarrow\mu_\delta^{(k)}}[u\in\mathcal{G}_0] - \frac{\varepsilon}{6k}}{\Pr_{u\leftarrow\mu_\delta^{(k)}}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]}\right]\\
&- \mathbb{E}_{\pi^*}\left[\left(\Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(D_2(x^*,r))=1](S_{\pi^*,1}-S_{\pi^*,0})+S_{\pi^*,0}\right)\frac{1}{\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]}\right].
\end{aligned} \tag{4.20}$$

We show that if Gen does not recurse, then the majority of estimates is low almost surely. Let us assume that

$$\Pr_{\pi}\left[\left(S_{\pi,0}\leq(1-\frac{1}{2k})\varepsilon\right)\wedge\left(S_{\pi,1}\leq(1-\frac{1}{2k})\varepsilon\right)\right] < 1-\frac{\varepsilon}{6k}, \tag{4.21}$$

then Gen recurses almost surely, because the probability that Gen does not find $\widetilde{S}_{\pi,b}\geq(1-\frac{3}{4k})\varepsilon$ in all of the $\frac{6k}{\varepsilon}n$ iterations is at most

$$\left(1-\frac{\varepsilon}{6k}\right)^{\frac{6k}{\varepsilon}n} \leq e^{-n}$$

almost surely, where we used Lemma 4.7. Therefore, under the assumption that Gen does not recurse with high probability it holds

$$\Pr_{\pi,\rho}\left[\left(S_{\pi,0}\leq(1-\frac{1}{2k})\varepsilon\right)\wedge\left(S_{\pi,1}\leq(1-\frac{1}{2k})\varepsilon\right)\right] \geq 1-\frac{\varepsilon}{6k}. \tag{4.22}$$

Let us define a set

$$\mathcal{W} = \left\{ \pi : \left(S_{\pi,0} \le (1 - \frac{1}{2k})\varepsilon \right) \wedge \left(S_{\pi,1} \le (1 - \frac{1}{2k})\varepsilon \right) \right\}, \tag{4.23}$$

and use $\overline{\mathcal{W}}$ to denote the complement of $\mathcal{W}$. We bound the numerator of the second summand in (4.20)

$$\begin{aligned}
&\mathbb{E}_{\pi^*}[S_{\pi^*,0} + \Pr_{\substack{r \\ x^* := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{trans} \\ (\Gamma_V,\Gamma_H) := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{P(1)}}} [\Gamma_V(D_2(x^*, r)) = 1](S_{\pi^*,1} - S_{\pi^*,0})] \\
&= \mathbb{E}_{\pi^*}\Big[S_{\pi^*,0} + \Pr_{\substack{r \\ x^* := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{trans} \\ (\Gamma_V,\Gamma_H) := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{P(1)}}} [\Gamma_V(D_2(x^*, r) = 1](S_{\pi^*,1} - S_{\pi^*,0}) \mid \pi^* \in \overline{\mathcal{W}}] \Pr_{\pi^*}[\pi^* \in \overline{\mathcal{W}}\Big] \\
&\quad + \mathbb{E}_{\pi^*}\Big[S_{\pi^*,0} + \Pr_{\substack{r \\ x^* := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{trans} \\ (\Gamma_V,\Gamma_H) := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{P(1)}}} [\Gamma_V(D_2(x^*, r)) = 1](S_{\pi^*,1} - S_{\pi^*,0}) \mid \pi^* \in \mathcal{W}] \Pr_{\pi^*}[\pi^* \in \mathcal{W}\Big] \\
&\le \frac{\varepsilon}{6k} + \mathbb{E}_{\pi^*}\Big[S_{\pi^*,0} + \Pr_{\substack{r \\ x := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{trans} \\ (\Gamma_V,\Gamma_H) := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{P(1)}}} [\Gamma_V(D_2^{\widetilde{C}}(x^*, r)) = 1]((1 - \frac{1}{2k})\varepsilon - S_{\pi^*,0}) \mid \pi^* \in \mathcal{W}\Big] \Pr_{\pi^*}[\pi^* \in \mathcal{W}] \\
&\le \frac{\varepsilon}{6k} + (1 - \frac{1}{2k})\varepsilon = (1 - \frac{1}{3k})\varepsilon.
\end{aligned} \tag{4.24}$$

We observe that

$$\begin{aligned}
\Pr_{u \leftarrow \mu_\delta^k}[g(u) = 1] &= \Pr[u \in \mathcal{G}_0 \vee (u \in \mathcal{G}_1 \setminus \mathcal{G}_0 \wedge u_1 = 1)] \\
&= \Pr[u \in \mathcal{G}_0] + \delta \Pr[u \in \mathcal{G}_1 \setminus \mathcal{G}_0].
\end{aligned} \tag{4.25}$$

Finally, we insert (4.24) into (4.20) which yields

$$\Pr_{\substack{\pi,\rho \\ x := \langle P^{(1)}(\pi), D_1(\rho)\rangle_{\text{trans}} \\ (\Gamma_V,\Gamma_H) := \langle P^{(1)}(\pi), D_1(\rho)\rangle_{P(1)}}} [\Gamma_V(D_2(x, \rho)) = 1] \ge \mathbb{E}_{\pi^*}\left[\frac{\Pr_{\pi^{(k)},\rho}[g(c) = 1 \mid \pi_1 = \pi^*] - \Pr_{u \leftarrow \mu_\delta^k}[u \in G_0] - (1 - \frac{1}{6k})\varepsilon}{\Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_1 \setminus \mathcal{G}_0]}\right].$$

From the assumptions of Lemma 4.5 it follows that

$$\Pr_{\pi^{(k)},\rho}[g(c) = 1] \ge \Pr_{u \leftarrow \mu_\delta^{(k)}}[g(u) = 1] + \varepsilon.$$

thus we get

$$\begin{aligned}
\Pr_{\substack{\pi^*,\rho \\ x := \langle P^{(1)}(\pi^*), D_1(\rho)\rangle_{\text{trans}} \\ (\Gamma_V,\Gamma_H) := \langle P^{(1)}(\pi^*), D_1(\rho)\rangle_{P(1)}}} [\Gamma_V(D_2(x, \rho)) = 1] &\ge \frac{\Pr_{u \leftarrow \mu_\delta^k}[g(u) = 1] + \varepsilon - \Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_0] - (1 - \frac{1}{6k})\varepsilon}{\Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_1 \setminus \mathcal{G}_0]} \\
&\overset{(4.25)}{\ge} \frac{\varepsilon + \delta \Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_1 \setminus \mathcal{G}_0] - (1 - \frac{1}{6k})\varepsilon}{\Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_1 \setminus \mathcal{G}_0]} \ge \delta + \frac{\varepsilon}{6k}
\end{aligned} \tag{4.26}$$

Clearly, the running time of Gen is $poly(k, \frac{1}{\varepsilon}, n)$. □

4.1.5 Putting it together

In the previous sections we have shown that it is possible to partition the domain Q such that if the number of hint and verification queries is small, then success probability of a solver for the k-wise direct product is still substantial. As shown in Lemma 4.5 we can build a circuit that returns a solution that is likely to be good. It remains to use these build blocks and prove Theorem 4.2.

Proof (of Theorem 4.2). Let $\widetilde{\text{Gen}}$ be the algorithm from Theorem 4.2, and $\widetilde{D} = (D_1, D_2)$ the corresponding solver circuit for P that is output by $\widetilde{\text{Gen}}$ as in Theorem 4.2. They are defined on the following code excerpts.

Circuit $\widetilde{D}_2^{D,P^{(1)},hash,g,\Gamma_V,\Gamma_H}(x, \rho)$

Oracle: A circuit $D := (D_1, \widetilde{D}_2)$ from Lemma 4.5, a problem poser $P^{(1)}$, functions $hash : Q \to \{0, 1, \ldots, 2(h+v)-1\}$, $g : \{0,1\}^k \to \{0,1\}$ a verification oracle Γ_V, a hint oracle Γ_H.
Input: Bitstrings $x \in \{0,1\}^*$, $\rho \in \{0,1\}^*$.

$(q, y) := D_2^{P^{(1)},\widetilde{C},hash,g,\Gamma_H}(x, \rho)$
Ask verification query (q, y) to Γ_V.

Algorithm $\widetilde{\text{Gen}}^{P^{(1)},g,C}(n, \varepsilon, \delta, k, h, v)$

Oracle: A problem poser $P^{(1)}$, a function $g : \{0,1\}^k \to \{0,1\}$, a solver circuit C for $P^{(g)}$.
Input: Parameters n, ε, δ, k, h, v.
Return: A circuit $\widetilde{D} = (D_1, \widetilde{D}_2)$.

$hash :=$ FindHash$((h+v)\varepsilon, n, h, v)$
Let $\widetilde{C} := (C_1, \widetilde{C}_2)$ be as in Lemma 4.4 with oracle access to C, $hash$.
$D := Gen^{P^{(1)},\widetilde{C},g,hash}(\varepsilon, \delta, n, k)$
return $\widetilde{D} := (D_1, \widetilde{D}_2)$

We show that Theorem 4.2 follows from Lemma 4.3 and Lemma 4.5. We fix $P^{(1)}$, g, $P^{(g)}$ in the whole proof and consider a solver circuit $C = (C_1, C_2)$, asking at most h hint queries and v verification queries, such that

$$\Pr_{\pi^{(k)},\rho}\left[Success^{P^{(g)},C}(\pi^{(k)}, \rho) = 1\right] \geq 16(h+v)\Big(\Pr_{u \leftarrow \mu_\delta^k}[g(u) = 1] + \varepsilon\Big).$$

First, we note that C meets the requirements of Lemma 4.3. Thus, the algorithm $\widetilde{\text{Gen}}$ can call FindHash to obtain *hash* such that with high probability it holds

$$\Pr_{\pi^{(k)},\rho}\left[\mathit{CanonicalSuccess}^{P^{(g)},C,hash}(\pi^{(k)},\rho)=1\right] \geq \Pr_{u\leftarrow\mu_\delta^k}[g(u)=1]+\varepsilon.$$

Applying Lemma 4.4 for C we obtain a circuit $\widetilde{C}=(C_1,\widetilde{C}_2)$ that satisfies

$$\Pr_{\substack{\pi^{(k)},\rho \\ x:=\langle P^{(g)}(\pi^{(k)}),C_1(\rho)\rangle_{trans} \\ (\Gamma_V^{(g)},\Gamma_H^{(k)}):=\langle P^{(g)}(\pi^{(k)}),C_1(\rho)\rangle_{P(g)}}}[\Gamma_V^{(g)}(\widetilde{C}_2^{\Gamma_H^{(k)},C_2,hash}(x,\rho))=1] \geq \Pr_{u\leftarrow\mu_\delta^k}[g(u)=1]+\varepsilon.$$

Now, we use the algorithm Gen as in Lemma 4.5 that yields a circuit $D=(D_1,D_2)$ which with high probability satisfies

$$\Pr_{\substack{\pi,\rho \\ x:=\langle P^{(1)}(\pi),D_1^{\widetilde{C}}(\rho)\rangle_{trans} \\ (\Gamma_H,\Gamma_V):=\langle P^{(1)}(\pi),D_1^{\widetilde{C}}(\rho)\rangle_{P(1)}}}[\Gamma_V(D_2^{P^{(1)},\widetilde{C},hash,g,\Gamma_H}(x,\rho))=1] \geq (\delta+\frac{\varepsilon}{6k}). \tag{4.27}$$

Finally, $\widetilde{\text{Gen}}$ outputs $\widetilde{D}=(D_1,\widetilde{D}_2)$ with oracle access to D, $P^{(1)}$, *hash*, g such that with high probability it holds

$$\Pr_{\pi,\rho}\left[\mathit{Success}^{P^{(1)},\widetilde{D}}(\pi,\rho)=1\right] \geq (\delta+\frac{\varepsilon}{6k}).$$

The running time of FindHash is $poly(h,v,\frac{1}{\varepsilon},n)$ with oracle calls and of Gen $poly(k,\frac{1}{\varepsilon},n)$ with oracle access. Thus, the overall running time of $\widetilde{Gen}$ is $poly(k,\frac{1}{\varepsilon},h,v,n,t)$ with oracle access. Furthermore, the circuit $\widetilde{D}$ asks at most $\frac{6k}{\varepsilon}\log(\frac{6k}{\varepsilon})h$ hint queries and one verification query. Finally, we have $Size(\widetilde{D}) \leq Size(C)\cdot\frac{6k}{\varepsilon}$. This finishes the proof of Theorem 4.2. $\square$

4.2 Discussion

Appendix A

Appendix

A.1 Basic Inequalities

Lemma A.1 (Chernoff Bounds) *For independent, identically distributed Bernoulli random variables* $X_1, \ldots, X_n$ *with* $X := \sum_{i=1}^{n} X_i$ *with* $\Pr[X_i = 1] = p_i$ *and* $\Pr[X_i = 0] = 1 - p_i$ *for all* $1 \leq i \leq n$. *we have the following inequalities for* $0 \leq \delta \leq 1$ *and* $\mathbb{E}[X] = \sum_{i=1}^{n} p_i$*:*

$$\Pr[X \geq (1+\delta)\mathbb{E}[X]] \leq e^{-\mathbb{E}[X]\delta^2/3} \tag{A.1}$$

$$\Pr[X \leq (1-\delta)\mathbb{E}[X]] \leq e^{-\mathbb{E}[X]\delta^2/2} \tag{A.2}$$

$$\Pr[|X - \mathbb{E}[X]| \geq \delta\mathbb{E}[X]] \leq 2e^{-\mathbb{E}[X]\delta^2/3}. \tag{A.3}$$

Bibliography

[CHS04] Ran Canetti, Shai Halevi, and Michael Steiner. Hardness amplification of weakly verifiable puzzles. 2004.

[CW77] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 106–112, New York, NY, USA, 1977. ACM.

[DIJK09] Yevgeniy Dodis, Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. Security amplification for interactive cryptographic primitives. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 128–145, Berlin, Heidelberg, 2009. Springer-Verlag.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[Hol13a] Thomas Holenstein. Lecture notes in complexity theoretic cryptography, Spring 2013.

[Hol13b] Thomas Holenstein. Lecture notes in complexity theory, Spring 2013.

[HS10] Thomas Holenstein and Grant Schoenebeck. General hardness amplification of predicates and puzzles. *CoRR*, abs/1002.3534, 2010.

[IJK07] Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. Chernoff-type direct product theorems. In *Advances in Cryptology-CRYPTO 2007*, pages 500–516. Springer, 2007.

[Mau13] Ueli Maurer. Lecture notes in cryptography, Spring 2013.

[VABHL03] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. Captcha: Using hard ai problems for security. In *Advances in Cryptology—EUROCRYPT 2003*, pages 294–311. Springer, 2003.