Abstract

Weakly verifiable puzzles can be seen as problems where a solution cannot be efficiently verified by the solver. They are introduced by Canetti et al. (Theory of Cryptography, 2005), who give a first hardness amplification result. Dodis et al. (Theory of cryptography, 2009) define dynamic weakly verifiable puzzles that generalize, for example, MACs and signature schemes. They prove a hardness amplification result for the case where the solver must solve some fraction of puzzles. Holenstein and Schönebeck (Theory of Cryptography, 2011) introduce interactive weakly verifiable puzzles that generalize, for example, commitment schemes. They study hardness amplification for these puzzles and give the right bound even for the setting where a monotone function applied to the output of the checking circuit of the puzzle has to evaluate to one.

In this thesis we introduce dynamic interactive weakly verifiable puzzles by combining the definitions of Dodis et al. and Holenstein et al. We combine their proofs to obtain a hardness amplification result for monotone functions in this case.

Contents

Contents **iii**

1 Introduction **1**
1.1 Weakly Verifiable Puzzles 1
1.2 Hardness Amplification 1
1.3 Previous works 2
1.4 Contribution of the Thesis 3
1.5 Organization of the Thesis 3

2 Notation and Definitions **5**

3 Dynamic Interactive Weakly Verifiable Puzzles **9**
3.1 The Definition 9
3.2 The Hardness Amplification Theorem 10

4 Examples of Weakly Verifiable Cryptographic Primitives **15**
4.1 Message Authentication Codes 15
4.2 Public Key Signature Schemes 17
4.3 Bit Commitment Protocols 18
4.4 Automated Turing Tests 20

5 Previous Results **21**
5.1 Weakly Verifiable Puzzles 21
5.1.1 The definition 21
5.1.2 The hardness amplification 22
5.2 Dynamic Weakly Verifiable Puzzles 26
5.2.1 The definition 26
5.2.2 The hardness amplification theorem 27
5.3 Interactive Weakly Verifiable Puzzles 30
5.3.1 The definition 30
5.3.2 The hardness amplification theorem 31

6 Hardness Amplification for Dynamic Interactive Weakly Verifiable Puzzles **33**
6.1 Proof of Hardness Amplification 33
6.1.1 The hardness amplification theorem 33
6.1.2 The intuition behind the proof 35
6.1.3 The proof of hardness amplification theorem 36
6.1.4 Domain partitioning 41
6.1.5 The hardness amplification proof for partitioned domains 45
6.2 Discussion 55
6.2.1 Optimality of the result 55

A Appendix **57**
A.1 Concentration Bounds 57
A.2 The Proof of Lemma 6.3 Under the Simplifying Assumptions . 58

Bibliography **59**

Chapter 1

Introduction

1.1 Weakly Verifiable Puzzles

The notion of a *weakly verifiable puzzle* was introduced by Cannetti et al. [CHS05]. Intuitively, it is a problem generated by a party called a poser and solved by a solver. One does not require the solver to have an efficient way to verify a solution. On the other hand, the poser has an access to some secret information which makes the task of verifying a solution easy.

An example of a weakly verifiable puzzle is a CAPTCHA (an automated Turing test) defined by Ahn et al. [VABHL03]. It is easily solvable by humans but is at least mildly harder to solve for computer programs. The poser generates an instance of a CAPTCHA together with the unique correct solution. Therefore, it can trivially verify solutions. The solution space is often relatively small, thus it must be harder for the solver being a computer program to verify a solution.

1.2 Hardness Amplification

An important task in cryptography is to turn a certain problem that is only mildly hard into one that is substantially harder. An example is the well known hardness amplification lemma for a one-way function by Yao [Yao82]. This result implies that it is possible to build a strong one-way function from a function that is only weakly one-way.

A similar hardness amplification statement can be studied for weakly verifiable puzzles. Given a puzzle that can be solved with substantial probability, we want to construct a puzzle that is considerably harder to solve.

There are two approaches to amplify hardness for weakly verifiable puzzles. Both base on combining several weakly-hard puzzles into a construction that is strongly-hard. First, one can use *sequential repetition* where puzzles are solved

in rounds that start one after another. Ahn et al. [VABHL03] observed that sequential repetition amplifies hardness for weakly verifiable puzzles. However, this approach may be inefficient as it increases the number of communication rounds. Often a better solution from the practical reasons is to amplify hardness by *parallel repetition* where several independent puzzles are sent in one round. We note that Bellare et al. [BIN97] show that parallel repetition may fail to amplify hardness in certain settings.

To prove hardness amplification one has to show that the following implication holds

$$A \implies B,$$

where A is a statement that a problem P is hard, and B denotes that a problem Q is hard. It turns out that it is often easier to consider the following logically equivalent implication

$$\neg B \implies \neg A.$$

This approach is used in the thesis. More precisely, we assume existence of an efficient algorithm that successfully solves parallel repetition of weakly verifiable puzzles with substantial probability. Under this assumption we construct an efficient algorithm that success probability in solving a single puzzle is substantial.

1.3 Previous works

The Yao's proof of hardness amplification for one-way functions relies to a great extent on the fact that it is possible for a solver to easily verify the correctness of a solution. Therefore, to show hardness amplification for weakly verifiable puzzles a different approach has to be developed.

Cannetti et al. [CHS05] define weakly verifiable puzzles and give the hardness amplification proof for these puzzles.

Hardness amplification for weakly verifiable puzzles in a situation where the solver can make several mistakes but still successfully solves parallel repetition of puzzles is studied by Impagliazzo et al. [IJK07]. They introduce a *threshold function* that determines the minimal fraction of puzzles that must be successfully solved.

Holenstein and Schoenebeck [HS11] give a more general proof for hardness amplification of weakly verifiable puzzles. They use arbitrary *binary monotone functions* in place of threshold functions. Additionally, they introduce *interactive weakly verifiable puzzles* where a puzzle is defined by an interactive protocol between the poser and the solver. These puzzles generalize, for example, the binding property of commitment protocols.

Dodis et al. [DIJK09] extend the proof of hardness amplification of Imaglizazzo et al. Not only, they consider threshold functions but also introduce *dynamic weakly verifiable puzzles* where a puzzle is defined together with a set of indices $\mathcal{Q}$. The solver must give a correct answer on any $q \in \mathcal{Q}$. Furthermore, it can verify the correctness of a limited number of solutions and obtain several hints understood as a correct solution on some $q \in \mathcal{Q}$. The solver must solve a puzzle on q for which it did not ask a hint before. A dynamic weakly verifiable puzzle can be seen as, for example, a game of breaking security of MAC. In this setting $\mathcal{Q}$ is a set of messages. The task of the solver is to provide a valid tag for $q \in \mathcal{Q}$. The solver can obtain tags for messages of his choice, but must provide a correct tag for a message for which it did not obtain a hint before.

1.4 Contribution of the Thesis

In this thesis we introduce the notion of *dynamic interactive weakly verifiable puzzles* that generalizes dynamic and interactive puzzles. We give a proof of hardness amplification for this type of puzzles by combining techniques used by Dodis et al. [DIJK09] and Holenstein et al. [HS11]. As a result we show that it is possible to amplify hardness for dynamic interactive weakly verifiable puzzles where to create an instance of a puzzle the solver and the poser engage in an interactive protocol. Furthermore, the solver can verify a limited number of solutions and obtain several correct solutions on some $q \in \mathcal{Q}$. Finally, the proof applies also to a setting where a monotone binary function is used to decide which puzzles of parallel repetition have to be successfully solved.

1.5 Organization of the Thesis

In Chapter 2 we set up the notation and terminology used in the thesis. We also define a *monotone binary function* and a *pairwise independent family of efficient hash functions*.

Next, in Chapter 3 we introduce *dynamic interactive weakly verifiable puzzles* and state the hardness amplification theorem for this type of puzzles. Proving this theorem is the main focus of the thesis.

Chapter 4 is devoted to the different types of cryptographic primitives that can be seen as weakly verifiable. We briefly describe MACs, signature schemes, commitment protocols, and CAPTCHAs and explain why these constructions are generalized by dynamic interactive weakly verifiable puzzles.

Then, in Chapter 5 we give an outline of the earlier studies of weakly verifiable puzzles and compare them with the results obtained in this thesis.

Finally, in Chapter 6 we give the proof of hardness amplification for dynamic interactive weakly verifiable puzzles and discuss this result.

Chapter 2

Notation and Definitions

In this chapter we set up the notation and definitions used in the thesis.

Bitstrings and tuples. We denote a tuple $(x_1, \ldots, x_n)$ by $x^{(n)}$. The i-th element of $x^{(n)}$ is denoted by $x_i^{(n)}$. Furthermore, for $x^{(n)}$, $y^{(k)}$ we use $x^{(n)} \circ y^{(k)}$ to denote the concatenation of $x^{(n)}$ and $y^{(k)}$ which results in a tuple $(x_1, \ldots, x_l, y_1, \ldots, y_k)$. We denote a bitstring $(b_1, \ldots, b_n) \in \{0,1\}^n$ as a n-tuple of bits.

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

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

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

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

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.

For a function $f(n) : \mathbb{R} \to \mathbb{R}$ we write $f(n) = poly(n)$ to denote that there exists a polynomial $p(n)$ such that $f(n) = O(p(n))$.

Definition 2.1 (Binary monotone function)**.** For $n \in \mathbb{N}$ we say that $g : \{0,1\}^n \to \{0,1\}$ is a *binary monotone function* if for any two bitstrings $(b_1, \ldots, b_n) \in \{0,1\}^n$ and $(b'_1, \ldots, b'_n) \in \{0,1\}^n$ and sets $\mathcal{I} := \{i \in \{1, \ldots, n\} :$

$b_i = 1\}$ and $\mathcal{I}' := \{i \in \{1, \ldots, n\} : b'_i = 1\}$ it holds that if $\mathcal{I} \subseteq \mathcal{I}'$ and $g(b_1, \ldots, b_n) = 1$ then $g(b'_1, \ldots, b'_n) = 1$. $\Diamond$

Algorithms and Circuits. We define a *probabilistic circuit* as a Boolean circuit $C_{m,n} : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^*$ and write $C_{m,n}(x; \rho)$ to denote a probabilistic circuit that takes as input $x \in \{0,1\}^m$ and the randomness $\rho \in \{0,1\}^n$. If a probabilistic circuit takes as input only the randomness, we slightly abuse the notation and write $C_n(\rho)$. We make sure that it is clear from the context that probabilistic circuits that take as input only the randomness are not confused with deterministic circuits. We use $\{C_n\}_{n \in \mathbb{N}}$ to denote a family of probabilistic circuits. For a (probabilistic) circuit C we write $\mathit{Size}(C)$ to denote the total number of vertices of C. W define a *family of (probabilistic) polynomial size circuits* as a family of (probabilistic) circuits where the size of a circuit belonging to this family is polynomial in the number of input vertices.

Sometimes we talk about interactive protocols consisting of the two phases. In this settings we define a *two-phase circuit* $C := (C_1, C_2)$ as a circuit where in the first phase the circuit C_1 is used and in the second phase the circuit C_2. We write $C(\delta) := (C_1, C_2)(\delta)$ to denote that C_1 and C_2 use the same randomness $\delta \in \{0,1\}^*$.

We write $\mathit{Time}(A)$ to denote the number of steps it takes to execute an algorithm A as a function of the input length of A. We say that A runs in *polynomial time* if $\mathit{Time}(A) = \mathit{poly}(n)$ where $n \in \mathbb{N}$ is the input length of A.

Similarly as for a probabilistic circuit we often write the randomness used by a probabilistic algorithm explicitly.

Probabilities and distributions. For a finite set $\mathcal{R}$ we write $r \xleftarrow{\$} \mathcal{R}$ 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 whose probability depends on n. We say that E_n happens *almost surely* or *with high probability* if there exists $n_0 \in \mathbb{N}$ such that for all $n \in \mathbb{N} : n > n_0$ there exists $p(n)$ such that $\Pr[E_n] \geq 1 - 2^{-n} p(n)$.

Interactive protocols. We are often interested in situations where two probabilistic circuits interact with each other according to some protocol 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[1] such that $A_n : \{0,1\}^* \to \{0,1\}^*$ and $B_n : \{0,1\}^* \to \{0,1\}^*$.

[1] We represent the input of A_n as some tuple that is encoded into a bitstring. The same

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 a message $m_0 := A_n(\rho_A)$ is sent and in the second round a message $m_1 := B_n(\rho_B, m_0)$. 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 $m_{2k-1} := B_n(\rho_B, m_1, \cdots, m_{2k-2})$. The number of rounds is naturally bounded by the size of A_n and B_n.

The protocol execution between probabilistic circuits A and B is denoted by $\langle A, B\rangle$. The output of A is denoted by $\langle A, B\rangle_A$ and of B by $\langle A, B\rangle_B$. The sequence of all messages sent by A and B is called a *communication transcript* and is denoted by $\langle A, B\rangle_{trans}$. The time complexity of the protocol depends on the size of A_n and B_n. For example, we say that a protocol runs in polynomial time if the size of A_n and B_n are bounded by some $p(n)$.

Oracle algorithms. We use the notion of *oracle circuits* and *oracle algorithms* following the standard definition in the literature [Gol04, AB09]. If a circuit A has oracle access to a circuit B, we write A^B. If additionally B has oracle access to a circuit C, we write A^{B^C}. However, to shorten the notation we often write A^B instead and make sure that it is clear from the context which oracle is accessed by B. We use the same convention for the oracle algorithms.

Often when studying the time complexity of oracle algorithms we count each oracle call as a single step. We emphasize this by writing that an algorithm has a certain time complexity *with oracle calls*. On the other hand, in some settings we are interested in giving a more rigorous bound on the running time of an algorithm. In these situations we compute the running time with regard to the time needed for an oracle access.

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 all $x, y \in \mathcal{D}, x \neq y$ and for all $\alpha, \beta \in \mathcal{R}$, it holds that

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

Polynomial time samplable. The function $hash \in \mathcal{H}$ is samplable 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 $y = hash(x)$. $\Diamond$

approach is used for the input of B_n.

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 known [CW77] that there exists families of functions meeting the criteria stated in Definition 2.2.

Chapter 3

Dynamic Interactive Weakly Verifiable Puzzles

In the previous chapter we set up the notation and terminology used in this thesis. The focus of this chapter is on *dynamic interactive weakly verifiable puzzles*, which we introduce in Section 3.1. Then, in Section 3.2 we define the *direct product of puzzles* and state the hardness amplification theorem that proving is the main goal of this thesis.

3.1 The Definition

Let us combine definitions of puzzles introduced in [DIJK09] and [HS11] and define a *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* $C := (C_1, C_2)$ 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 $C(\rho) := (C_1, C_2)(\rho)$ 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 a result of the interaction $P_n(\pi)$ outputs a *verification circuit* Γ_V and a probabilistic *hint circuit* Γ_H. The circuit $C_1(\rho)$ produces no output. The circuit Γ_V takes as input $q \in \mathcal{Q}$ (for some set $\mathcal{Q}$ of indices), $y \in \{0,1\}^*$ 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 $q \in \mathcal{Q}$ 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 ran-

domness 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 this q. $\Diamond$

If it is clear from the context, we omit the subscript n and write P instead of P_n.

A verification query (q, y) of 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 throughout this chapter, we make the assumption that C does not ask verification queries on q for which a hint query has been asked before. Moreover, we assume that once C asked a verification query that succeeds, it does not ask any further hint or verification queries.

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

We use the term *weakly verifiable* to emphasize that there is no easy way for the solver to check the correctness of a solution except for asking a verification query. We call a weakly verifiable puzzle *dynamic* if the number of hint queries is greater than zero. Furthermore, we say that a weakly verifiable puzzle is *interactive* if in the first phase the number of messages exchanged between the problem poser and the solver is greater than one. Finally, we say that a weakly verifiable puzzle is *non-dynamic* if $|Q| = 1$ and *non-interactive* if the number of messages sent in the first phase is at most one.

Definition 3.1 generalizes and combines the previous approaches that study *weakly verifiable puzzles* [CHS05], *dynamic weakly verifiable puzzles* [DIJK09], and *interactive weakly verifiable puzzles* [HS11].

3.2 The Hardness Amplification Theorem

In this section we define the k-wise direct product of puzzles and state the hardness amplification theorem for dynamic interactive weakly verifiable puzzles.

Definition 3.2 (k-wise direct-product of DIWVPs)**.** Let $g : \{0,1\}^k \to \{0,1\}$ be a binary monotone function and $P_n^{(1)}$ a problem poser as in Definition 3.1.

[1] Theorem 6.10 from [Hol13b] is stated for probabilistic, polynomial time, oracle algorithms with a single bit of output, but it can be adopted to the case where an output is longer than a single bit.

The *k-wise direct product of* $P_n^{(1)}$ is a dynamic interactive weakly verifiable puzzle 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 $\pi_i \in \{0,1\}^n$ for all $1 \leq i \leq n$. Let $(C_1, C_2)(\rho)$ be a probabilistic two-phase circuit called a *solver*. In the first phase, the problem poser $P_{kn}^{(g)}(\pi^{(k)})$ sequentially interacts in k rounds with $C_1(\rho)$. In the i-th round $C_1(\rho)$ interacts with $P_n^{(1)}(\pi_i)$, and as a 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

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

$\Diamond$

For the k-wise direct product of puzzles we require the solver to make a verification query on a single index $q \in \mathcal{Q}$. Otherwise, if verification queries of the form $((q_1, y_1), \ldots, (q_k, y_k))$ were allowed, solving the k-wise direct product of puzzles would be trivial. One could fix some $\bar{q} \in \mathcal{Q}$ and ask k hint queries such that the i-th hint query is of the form $(q_1, \ldots, q_{i-1}, \bar{q}, q_{i+1}, \ldots, q_k)$ where for all $1 \leq j \leq k$ such that $i \neq j$ it holds $q_j \in \mathcal{Q} \setminus \{\bar{q}\}$. The solver obtains correct solutions for $\bar{q}$ to the puzzles on all k positions and can trivially make a successful verification query.

In the following code listing we define an experiment *Success* such that it outputs 1 if and only if C asks a successful verification query.

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

Oracles: 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 ⟨P(π), C_1(ρ)⟩
    (Γ_V, Γ_H) := ⟨P(π), C_1(ρ)⟩_P
    x := ⟨P(π), C_1(ρ)⟩_trans

run C_2^{Γ_V,Γ_H}(x; ρ)
    if C_2^{Γ_V,Γ_H}(x; ρ) asks a verification query (q, y) s.t. Γ_V(q, y) = 1 then
        return 1
return 0
```

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

$$\Pr_{\pi,\rho}[Success^{P,C}(\pi,\rho)=1]. \tag{3.1}$$

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

We now state our main theorem. Loosely speaking, we claim that it is possible to reduce a solver for the k-wise direct product of P to a solver for a single puzzle defined by P. This implies that if there is no good solver for P, then also a good solver for the k-wise direct product of P does not exist.

Theorem 3.3 (Hardness amplification for dynamic interactive 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 $P_n^{(1)}$ as in Definition 3.2. Additionally, let C be a 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 C, a binary monotone function $g:\{0,1\}^k\to\{0,1\}$, and a problem poser $P_n^{(1)}$. The algorithm Gen takes as input parameters ε, δ, n, k, h, v, and outputs a solver circuit D for $P_n^{(1)}$ such that:*
If C satisfies

$$\Pr_{\substack{\pi^{(k)}\in\{0,1\}^{kn}\\ \rho\in\{0,1\}^*}}\Big[Success^{P_{kn}^{(g)},C}(\pi^{(k)},\rho)=1\Big]\geq 16(h+v)\Big(\Pr_{u\leftarrow\mu_\delta^k}[g(u)=1]+\varepsilon\Big), \tag{3.2}$$

then D is a two-phase probabilistic circuit that almost surely over the randomness of Gen satisfies

$$\Pr_{\substack{\pi\in\{0,1\}^{n}\\ \rho\in\{0,1\}^*}}\Big[Success^{P_n^{(1)},D}(\pi,\rho)=1\Big]\geq\delta+\frac{\varepsilon}{6k}. \tag{3.3}$$

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

The above theorem is very general in the sense that it does not pose any constrains on the size of the circuits or the time complexity of the interactive protocol. Additionally, we count each oracle call as a single step.

Let us consider, as an example, the case where C and $P_{kn}^{(g)}$ are polynomial time probabilistic algorithms. Furthermore, we assume that C is such that it satisfies (3.2) and $\frac{1}{\varepsilon}$ and k are bounded by some polynomial $p(n)$. Clearly there are the polynomial size families of probabilistic circuits that correspond

to $P_{kn}^{(g)}$ and C. The running time of Gen is polynomial, therefore the size of D must also be polynomial. Finally, the circuit D can be executed by a polynomial time algorithm. Thereby, we obtain a polynomial time reduction as described in the literature [AB09, Mau13].

Theorem 3.3 holds with high probability over the randomness of Gen. More precisely, the circuit output by Gen satisfies the condition of the theorem with probability at least $1 - p(k, n, \frac{1}{\varepsilon}) \cdot 2^{-n}$. Therefore, Theorem 3.3 is meaningful if there exists $p(n)$ that bounds k and $\frac{1}{\varepsilon}$.

We emphasize that the number of hint queries asked by D is greater than the number of hint queries asked by C whereas the number of verification queries is limited to at most one. For many cryptographic constructions making such an assumption about the number of hint and verification queries is reasonable. In particular, we cannot assume that a solver for a single puzzle may ask more verification queries than a solver for a k-wise direct product of puzzles.

In Chapter 2 we defined a binary monotone function. The monotone restriction on g in Theorem 3.3 is essential. For $g(b) := 1 - b$ a solver circuit that deliberately gives incorrect answers satisfies g with probability 1 whereas a circuit that solves a puzzle successfully with probability $\gamma > 0$ succeeds only with probability $1 - \gamma$.

We prove Theorem 3.3 in Chapter 6.

Chapter 4

Examples of Weakly Verifiable Cryptographic Primitives

In the previous chapter we formally defined dynamic interactive weakly verifiable puzzles and stated the hardness amplification theorem for this type of puzzles. In this chapter we give an overview of cryptographic primitives that can be seen as various types of weakly verifiable puzzles. First, in Section 4.1 we describe message authentication codes which the game based security definition can be seen being a dynamic non-interactive weakly verifiable puzzle. In Section 4.2 we give the chosen-message attack security definition for public key signature schemes which is yet another example of a dynamic non-interactive weakly verifiable puzzle. Then, in Section 4.3 we focus on commitment protocols, which binding property can be seen as a non-dynamic interactive weakly verifiable puzzle. Finally, in Section 4.4 we describe a problem of solving CAPTCHAs that can be considered to be a non-dynamic and non-interactive weakly verifiable puzzle.

4.1 Message Authentication Codes

In this section we describe message authentication codes (MACs) and conclude that the game based definition of MAC security can be seen being a dynamic non-interactive weakly verifiable puzzle.

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

Loosely speaking, MAC can be explained as follows. Let the sender, receiver, and adversary be polynomial time algorithms and messages be represented as

bitstrings. Furthermore, we assume that the sender and the receiver share a secret key to which the 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 efficiently check whether 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 was not sent before and with non-negligible probability is accepted by the receiver.

Below we give the formal definition of a *message authentication code* based on [Mau13, Gol04].

Definition 4.1 (Message Authentication Code)**.** Let $\mathcal{M}$ be a set of messages, $\mathcal{K}$ a set of keys, and $\mathcal{T}$ a set of tags. A *message authentication code* (MAC) is defined by an efficiently computable function $f : \mathcal{M} \times \mathcal{K} \to \mathcal{T}$. We say that a MAC is *secure* if f satisfies the following condition:

Let k be chosen uniformly at random from $\mathcal{K}$ and H be a polynomial size circuit with hard-coded k that takes as input a message $m \in \mathcal{M}$ and outputs a tag $t \in \mathcal{T}$ such that $f(m, k) = t$. We say that a MAC is secure if there is no probabilistic polynomial time algorithm with oracle access to H that with non-negligible probability outputs $m \in \mathcal{M}$ as well as $t \in \mathcal{T}$ such that $f(m, k) = t$ and H has not been queried on m. $\Diamond$

We describe how the security definition of a MAC can be seen as a dynamic weakly verifiable puzzle where at most one verification query is asked. For fixed f and $n \in \mathbb{N}$ the sender corresponds to the problem poser and the adversary to the solver. Furthermore, the key k_π depends on the randomness $\pi \in \{0, 1\}^n$ used by the problem poser. The set $\mathcal{Q}$ is the set of messages $\mathcal{M}$.

In the first phase, which is non-interactive, neither the problem poser nor the solver send any messages. 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 corresponds to the circuit H from Definition 4.1 and takes as input a message m and outputs a tag t such that $f(m, k_\pi) = t$. The circuit Γ_V takes as input $m \in \mathcal{M}$ and $t \in \mathcal{T}$ and outputs 1 if and only if $f(m, k_\pi) = t$. In the second phase, the solver takes no input (x is empty string) and has oracle access to Γ_H, Γ_V and can make at most one verification query.

Thus, the task of the adversary to find a valid tag $t \in \mathcal{T}$ for a message $m \in \mathcal{M}$ such that the hint query for m has not been asked before corresponds to making a successful verification query by the solver.

4.2 Public Key Signature Schemes

In this section we describe *public key signature schemes* (SIGs). The game-based security definition of SIGs for a chosen message attack can be seen as a dynamic, non-interactive weakly verifiable puzzle.

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

Definition 4.2 (Public key signature scheme)**.** Let $\mathcal{Q}$ be a set of messages. A *public key signature scheme* is defined by a triple of polynomial time algorithms: G – a probabilistic key-generation algorithm, V – a verification algorithm, S – a probabilistic signing algorithm, such that the following conditions are satisfied:

- The key-generation algorithm $G(1^n; \rho)$ outputs a pair of bitstrings $k_{priv} \in \{0,1\}^*$ and $k_{pub} \in \{0,1\}^*$ where $n \in \mathbb{N}$ is a security parameter and ρ_G the randomness used by G. We call k_{priv} a *private key* and k_{pub} a *public key.*
- The signing algorithm S takes as input k_{priv}, $q \in \mathcal{Q}$, uses the randomness ρ_S and outputs a signature $s \in \mathcal{S}$.
- The verification algorithm V takes as input k_{pub}, $q \in \mathcal{Q}$, and $s \in \mathcal{S}$ and outputs $b \in \{0,1\}$[1].
- For every k_{priv}, k_{pub} output by G and every $q \in \mathcal{Q}$ it holds

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

  where the probability is over the randomness of S.

$\Diamond$

We say that $s \in \mathcal{S}$ is a *valid signature* for $q \in \mathcal{Q}$ if and only if $V(k_{pub}, q, s) = 1$. We define the security of public key signature scheme as follows.

Definition 4.3 (Security of a public key signature scheme with respect to a chosen message attack)**.** Let H be a polynomial size probabilistic circuit that has hard-coded k_{priv} and takes as input q and outputs $S(k_{priv}, q)$. An *adversary* A is a probabilistic polynomial time algorithm that takes as input k_{pub} and has oracle access to H. We say that A *succeeds* if it finds a valid signature $s \in \mathcal{S}$ for a message $q \in \mathcal{Q}$, and the oracle H 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. $\Diamond$

It seems that a game of breaking the security of a public key signature scheme is not weakly verifiable as the adversary can use V to efficiently check the correctness of a solution. However, it is easy to see that this situation can be

[1] For simplicity we assume that the verification algorithm is deterministic.

modeled as a dynamic weakly verifiable puzzle where the solver has access to the verification circuit and can ask a polynomial number of verification queries.

More precisely, we now describe how breaking the security of a public key signature scheme can be seen as a dynamic non-interactive weakly verifiable puzzle.

The problem poser corresponds to the key-generation algorithm G and the solver to the adversary. The set $\mathcal{Q}$ is a set of messages. In the first phase, the problem poser obtains k_{pub}, k_{priv} and sends k_{pub} to the solver. Then, the problem poser generates a hint circuit Γ_H and a verification circuit Γ_V. The hint circuit Γ_H corresponds to the circuit H from Definition 4.3 and takes as input $q \in \mathcal{Q}$ and outputs a valid signature for q. The verification circuit Γ_V takes as input $s \in \mathcal{S}$ and $q \in \mathcal{Q}$ and outputs 1 if and only if $s \in \mathcal{S}$ is a valid signature for $q \in \mathcal{Q}$. In the second phase, the solver takes as input a transcript of messages from the first phase which consists solely of k_{pub}. Additionally, the solver has oracle access to Γ_V and Γ_H and can make a polynomial number of queries to Γ_H and Γ_V. The adversary calls to V can be simulated by making a verification query to Γ_V. It is clear that if the solver asks a successful verification query (q, s), then there exists an adversary that also finds a valid signature for q.

Thus, the task of finding a valid signature in a public key signature scheme can be seen as a weakly verifiable puzzle that is dynamic but non-interactive as in the first phase only a single message is sent.

4.3 Bit Commitment Protocols

In this section we describe *bit commitment protocols*. The binding property of a bit commitment protocol can be seen as an interactive non-dynamic weakly verifiable puzzle where $|\mathcal{Q}| = 1$.

Loosely speaking, we 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 $b \in \{0, 1\}$. We require that after the commit phase it is hard for the receiver to guess b correctly. In the reveal phase the sender opens the commitment by sending to the receiver a pair (b', y) where $y \in \{0, 1\}^*$ should convince the receiver that the sender committed to the value $b' \in \{0, 1\}$. A desirable property of a bit commitment protocol is that in the reveal phase it is hard for the sender to find two bitstrings y_0 and y_1 such that the receiver recognizes both $(0, y_0)$ and $(1, y_1)$ to be valid decommitments.

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

Definition 4.4 (Bit Commitment Protocol)**.** A *bit commitment protocol* is defined by families of circuits $\{S_n\}_{n\in\mathbb{N}}$ and $\{R_n\}_{n\in\mathbb{N}}$ where $S_n = (S_1, S_2)$ is a two-phase probabilistic circuit, R_n is a probabilistic circuit, and $n \in \mathbb{N}$ is a security parameter. We call S_n a *sender* and R_n a *receiver*. The circuit S_1 takes as input a pair (b, ρ_S) where $b \in \{0,1\}$ is interpreted as a bit to which S commits, and $\rho_S \in \{0,1\}^*$ is the randomness used by S. The receiver R_n uses the randomness $\rho_R \in \{0,1\}^*$. The protocol consists of two phases. In the *commit phase*, S_1 and R_n engage in the protocol execution. As the result, S_n commits to b and R_n generates a circuit V. The circuit V takes as input $b' \in \{0,1\}$ and $y \in \{0,1\}^*$ and outputs a bit. In the reveal phase, the circuit S_2 takes as input a communication transcript from the commitment phase $\langle S_1, R_n\rangle_{trans}$, the randomness 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

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

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

Hiding. For every $b \in \{0,1\}$ the probability over the randomness of S_n and R_n that any polynomial size circuit can guess 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 probabilistic polynomial size two-phase circuit $S_n^*(\rho_S) := (S_1^*, S_2^*)(\rho_S)$ we have

$$\Pr_{\substack{\rho_S\in\{0,1\}^*,\rho_R\in\{0,1\}^* \\ V:=\langle S_1^*(b,\rho_S),R_n(\rho_R)\rangle_{R_n} \\ ((0,y_0),(1,y_1)):=S_2^*(\rho_S)}} [V(0, y_0) = 1 \wedge V(1, y_1) = 1] \leq \varepsilon(n),$$

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

$\Diamond$

Breaking the binding property of a bit commitment protocol can be seen as solving an interactive weakly verifiable puzzle. The problem poser corresponds to the receiver and the solver to the circuit that tries to break the binding property of the bit commitment protocol. When the interaction in the commit phase is completed the problem poser generates circuits Γ_V, Γ_H. The circuit Γ_V takes as input $y_0 \in \{0,1\}^*$, $y_1 \in \{0,1\}^*$ and outputs 1 if and only if $V(0, y_0) = 1 \wedge V(1, y_1) = 1$. The hint circuit Γ_H outputs $\perp$ for any query. Thus, without loss of generality, we assume that the solver does not ask any verification queries. Furthermore, $|\mathcal{Q}| = 1$, therefore we do not write explicitly on which $q \in \mathcal{Q}$ the solver asks a verification query. In the second phase, the

solver is given oracle access to Γ_V and is allowed to ask at most one verification query. Finally, we conclude that for the problem poser and the solver defined as above making a successful verification query corresponds to breaking the binding property of a bit commitment protocol.

4.4 Automated Turing Tests

In this section we describe CAPTCHAs which can be seen as non-dynamic and non-interactive weakly verifiable puzzles.

The goal of *automated Turing tests* is to distinguish humans from computer programs. An example of such a test is a *CAPTCHA*, which is formally defined in [VABHL03]. An example of a CAPTCHA is an image depicting a distorted text where the task is to guess the text used to generate the image.

Solving a CAPTCHA can be seen as a non-dynamic, non-interactive weakly verifiable puzzle. In the first phase the problem poser sends the solver an image containing a distorted text. Then, the problem poser generates circuits Γ_V and Γ_H. The circuit Γ_V takes as input a bitstring y and outputs 1 if and only if y correctly encodes the text depicted in the distorted image. A hint circuit Γ_H outputs $\perp$ in response to every query. Therefore, without loss of generality, we assume that the solver makes no hint queries.

In the second phase, the solver takes as input the image, has oracle access to Γ_V and can ask at most a single verification query. In general, for CAPTCHAs checking the correctness of a solution is comparably hard to finding a correct solution. Thus we conclude that a task of correctly guessing a text depicted in the image can be seen as a weakly verifiable non-interactive and non-dynamic puzzle.

It is unknown how good algorithms for solving a CAPTCHA can be. Thus, it is likely that a gap between the human performance and the performance of the best computer programs may be small. We are interested in finding a way to amplify this gap. In the papers [HS11, DIJK09] it was shown that this is possible by means of parallel repetition.

Chapter 5

Previous Results

In the last chapter we gave an overview of different types of cryptographic primitives that motivated studies of weakly verifiable puzzles. The focus of this chapter is on providing an outline of previous research. We give a short overview of techniques used in the series of papers [CHS05, DIJK09, HS11] and provide some intuition and insight into the hardness amplification results for weakly verifiable puzzles. First, we describe weakly verifiable puzzles studied in [CHS05] that are neither interactive nor dynamic. Then, in Section 5.2 we bring our focus on dynamic non-interactive puzzles studied in [DIJK09]. Finally, in Section 5.3 we give an overview of the results of Holenstein and Schoenebeck [HS11] where non-dynamic but interactive weakly verifiable puzzles are studied.

5.1 Weakly Verifiable Puzzles

The notion of *weakly verifiable puzzles* was coined by Canetti et al. in the paper *Hardness amplification of weakly verifiable puzzles* [CHS05]. The puzzles considered there are non-dynamic and non-interactive. Moreover, the number of verification queries is limited to one. This constitutes a special case of Definition 3.1. In this section we provide the definition of weakly verifiable puzzles (WVPs) and state the theorem of hardness amplification for WVPs in a similar vein as in [CHS05]. Finally, we give the intuition behind the proof of this theorem. It is noteworthy that the main proof of this thesis, presented in Chapter 6, uses many ideas of the work of Canetti et al. [CHS05].

5.1.1 The definition

We give the definition of WVP from [CHS05]. However, we use the notation and terminology defined in this thesis.

Definition 5.1 (Weakly Verifiable Puzzle, [CHS05])**.** 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. For a security parameter $k \in \mathbb{N}$ we write $G(1^k; \rho)$ to denote that G takes as input 1^k and uses the randomness $\rho \in \{0,1\}^*$. The algorithm G outputs $p \in \{0,1\}^*$ and a check information $c \in \{0,1\}^*$. The *verifier* V takes as input p, c, an answer $a \in \{0,1\}^*$ and outputs $b \in \{0,1\}$.

A *solver* S for G is a probabilistic polynomial time 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 (V, G) 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. $\Diamond$

Let us argue that Definition 5.1 is a special case of Definition 3.1. First, we note that if G takes as input 1^k, then the length of the randomness used is bounded by some polynomial $p(k)$. For a fixed k, without loss of generality, we can represent $G(1^k; \rho)$ as a probabilistic circuit of polynomial size that takes as input only the randomness ρ. In Definition 5.1 a verification algorithm V 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. Furthermore, for weakly verifiable puzzles we have that $|Q| = 1$ and thus, it is not necessary to pass q as the parameter to Γ_V. Hence, for fixed p and c the algorithm V corresponds to Γ_V. The puzzles considered in Definition 5.1 are non–dynamic. Thus, there is no element corresponding to the hint circuit Γ_H, and without loss of generality, we can assume that Γ_H outputs $\perp$ for every query. Finally, we note that the puzzles described in Definition 5.1 are non-interactive.

5.1.2 The hardness amplification

We give the definition of the n-wise direct product of weakly verifiable puzzles.

Definition 5.2 (n-wise direct product of weakly verifiable puzzles, [CHS05])**.** Let $n \in \mathbb{N}$ and a weakly verifiable puzzle $P := (G, V)$ be fixed. We define the n-wise direct product of P as a weakly verifiable puzzle where the puzzle-generation algorithm $G^{(n)}$ takes as input $1^{k\cdot n}$, uses the randomness $\rho \in \{0,1\}^*$, and outputs tuples $p^{(n)} := (p_1, \ldots, p_n) \in \{0,1\}^*$ and $c^{(n)} := (c_1, \ldots, c_k) \in \{0,1\}^*$ where for each $1 \le i \le n$ a pair (p_i, c_i) is an independent instance of a weakly verifiable puzzle defined by G and V with the security parameter k.

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 $1 \leq i \leq n$ it holds $V(p_i, c_i, a_i) = 1$. $\Diamond$

Let us set up the additional notation and terminology. We write $P^{(n)} := (G^{(n)}, V^{(n)})$ to denote the n-wise direct product of P. For $P^{(n)} := (G^{(n)}, V^{(n)})$ we say *puzzle on the i-th coordinate* to refer to the i-th puzzle of the n-wise direct product of P (this puzzle corresponds to the one generated by $G_i^{(n)}$, $V_i^{(n)}$).

The n-wise direct product of WVP is solved successfully if and only if all n puzzles are solved successfully. In contrast, in Definition 3.2 we are interested in a more general situation where a monotone function $g : \{0,1\}^n \to \{0,1\}$ is used to decide on which coordinates the puzzles of the n-wise direct product have to be solved successfully. Clearly, we can assume that g is such that the puzzles on all coordinates have to be solved successfully which matches the case considered in Definition 5.2.

The main theorem proved in [CHS05] states that it is possible to turn a good solver for $P^{(n)}$ into a good solver for P.

Theorem 5.3 (Hardness amplification for weakly verifiable puzzles, [CHS05])**.** *Let $n, q \in \mathbb{N}$ and $\delta : \mathbb{N} \to (0,1)$ be an efficiently computable function. Moreover, let $P := (G, V)$ be a weakly verifiable puzzle. We denote the running time of the puzzle-generation algorithm G by T_G and of the verification algorithm V by T_V. If $S^{(n)}$ is a solver for the n–wise direct product of P that success probability is at least δ^n and the running time is T, then there exists a solver S for P with oracle access to $S^{(n)}$ that success probability is at least $\delta(1 - \frac{1}{q})$ and the running time is $O\Big(\frac{nq^3}{\delta^{2n-1}}(T + nT_G + nT_V)\Big)$.*

The parameter q is introduced as it is not possible to achieve the perfect hardness amplification. The following algorithm CHS-solver has oracle access to $S^{(n)}$ and solves a puzzle defined by P with probability at least $\delta(1 - \frac{1}{q})$. We denote by $p \in \{0,1\}^*$ the output of G, which is also the input taken by the CHS-solver. To make the notation shorter in the following code excerpts we do not write the randomness used by G explicitly. In the analysis of the running time of the CHS-solver we explicitly take into account the time needed for the oracle calls to $S^{(n)}$, V, G.

Algorithm: CHS-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.

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

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

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

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

Algorithm: *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
  $(p^*, c^*) := G(1^k)$
  $\bar{\nu}_i :=$ *EstimateResSuccProb*$^{S^{(n)},G,V}(\mathit{prefix} \circ p^*, i, n, k, q, \delta)$
  if $\bar{\nu}_i \ge \delta^{n-i}$ **then return** p^*
return $\perp$

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

Oracle: A solver algorithm for $P^{(n)}$, a verification algorithm V for P, a puzzle-generation algorithm G for P
Input: A i–tuple of puzzles *prefix*, parameters i, n, k, q, δ.

successes := 0
Repeat $M := \left\lceil \frac{84q^2}{\delta^{n-i}} \ln\left(\frac{18qn \cdot N_i}{\delta}\right)\right\rceil$ times
  $((p_{i+1}, \ldots, p_n), (c_{i+1}, \ldots, c_n)) := G^{(n-i)}(1^k)$
  $a^{(n)} := A(prefix, p_{i+1}, \ldots, p_n)$
  if $\forall_{i+1 \leq j \leq n} : V(p_j, c_j, a_j) = 1$ **then** *successes* := *successes* + *1*
return *successes*/M

A full proof of Theorem 5.3 is presented in [CHS05]. We limit ourselves to providing intuition why the CHS-solver transforms a good solver for the n–wise direct product of P into a good solver for a single puzzle defined by P.

Let us consider the n-wise direct product of P and, for simplicity, a deterministic solver $S^{(n)}$ for $P^{(n)} := (G^{(n)}, V^{(n)})$. We 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 are labeled with all possible tuples $(p_2, \ldots, p_n)$ output by $G^{(n)}$ when executed with the 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 bitstrings inferred by a column and a row of the cell. We make the following observation.

Observation 5.4 ([CHS05]). *Let $S^{(n)}$ be a deterministic polynomial time solver for $P^{(n)}$ that success probability is at least δ^n. Then, the matrix M defined as above has either a column with a $\delta^{(n-1)}$ fraction of cells that are 1^n vectors, or a conditional probability that a cell is of the form 1^n given that the last $(n-1)$ bits of this cell are equal 1 is at least δ.*

Let us explain, at least intuitively using Observation 5.4, how the CHS-solver that has oracle access to $S^{(n)}$ for $P^{(n)}$ can be used to solve a puzzle defined by P with substantial probability. We refer to a puzzle solved by the $CHS-solver$ as an *input puzzle*. The CHS-solver starts with the first position and tries to fix a bitstring p^* used to generate the puzzle on this position 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 use p^* to generate the puzzle on this position and repeat the whole process again in the consecutive iteration for the next position. If the CHS-solver fails to find a bitstring p^*, then we assume that there is no column of M that contains

a $\delta^{(n-1)}$ fraction of cells that are of the form 1^n. We use Observation 5.4 to conclude 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 the input puzzle p on this position and note that the remaining puzzles are generated by the CHS-solver. Thus, it is possible to efficiently verify whether these puzzles are successfully solved by $S^{(n)}$.

Obviously, the 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 cannot 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 the estimate returned by *EstimateResSuccProb* is incorrect.

It is possible to show that all these events happen with small probability such that the success probability of the CHS-solver is at least $\delta(1-\frac{1}{q})$ almost surely.

In Chapter 6 we study a more general class of puzzles that are not only weakly verifiable but also dynamic and interactive. Furthermore, we allow a situation where the solver successfully solves the n-wise direct product of P, although it succeeds only on a subset of coordinates of the n-wise direct product of P. It turns out that it is possible to use a similar technique of fixing puzzles on the consecutive positions of the n-wise direct product of puzzles to prove hardness amplification in this more general setting.

5.2 Dynamic Weakly Verifiable Puzzles

Some of the cryptographic primitives presented in Chapter 4 are not only weakly verifiable but also dynamic (MAC and SIG). This type of puzzles is defined and studied in [DIJK09]. We give a short overview of this work and state the definition of a *dynamic weakly verifiable puzzle* (DWVP) that closely follows the one included in [DIJK09]. Finally, we describe important parts of the proof of hardness amplification for DWVPs.

5.2.1 The definition

Definition 5.5 (Dynamic Weakly Verifiable Puzzle, [DIJK09])**.** A *dynamic weakly verifiable puzzle* is defined by a distribution $\mathcal{D}$ on pairs (x, α) where α is secret information and x is a bitstring. Furthermore, we consider a set $\mathcal{Q}$ (for some set of indices $\mathcal{Q}$) and a probabilistic polynomial time computable verification relation V such that $V(\alpha, q, r) = 1$ if and only if $r \in \{0,1\}^*$ is a correct answer to $q \in \mathcal{Q}$ on the set determined by α. Finally, let H be a probabilistic polynomial time computable *hint* function that on input α, q outputs a bitstring $\{0,1\}^*$.

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

Dynamic weakly verifiable puzzles generalize games of breaking the security of message authentication codes and public key signature schemes. In the case of MAC x is an empty bitstring. For the public key signature schemes x is a public key.

The above definition is a special case of Definition 3.1. To see this we observe that instead of considering a distribution $\mathcal{D}$ on pairs (x, α) we can use a probabilistic problem poser that outputs circuits Γ_H and Γ_V with hard-coded α that correspond to H and V, respectively. It is clear that the solver S from Definition 5.5 can be turned into a family of probabilistic polynomial size circuits with oracle access to Γ_H and Γ_V. Furthermore, in the first phase, which is non-interactive, a bitstring x is sent by the problem poser to the solver.

5.2.2 The hardness amplification theorem

Definition 5.6 (n-wise direct product of DWVPs, [DIJK09])**.** For a dynamic weakly verifiable puzzle $P := (\mathcal{D}, V, H)$ we define the n-wise direct product of P as a DWVP with a distribution $\mathcal{D}^{(n)}$ on tuples $(x_1, \alpha_1), \ldots, (x_n, \alpha_n)$ where each (x_i, α_i) is drawn from the distribution $\mathcal{D}$. Furthermore, a hint relation is defined by $H^{(n)}(q, \alpha_1, \ldots, \alpha_n) := (H(\alpha_1, q), \ldots, H(\alpha_n, q))$ and a verification relation $V^{(n)}(\alpha_1, \ldots, \alpha_n, r_1, \ldots, r_n, q)$ evaluates to 1 if and only if for at least $n - (1 - \gamma)\delta n$ of bitstrings $r_1, \ldots, r_n$ it holds $V(\alpha_i, q, r_i) = 1$ where $0 \leq \gamma, \delta \leq 1$. $\Diamond$

We write $P^{(n)} := (D^{(n)}, H^{(n)}, V^{(n)})$ to denote the n-wise direct product of P.

The above definition is more general than the one given in the previous section of the n-wise direct product of WVP. In Definition 5.2 it is enough if the solver succeeds only on a fraction of coordinates.

We write $(\mathcal{H}_{hint}, \mathcal{V}_{verif}) \leftarrow S(x; \delta)$ to denote the execution of S with the input $x \in \{0, 1\}^*$ and using the randomness $\delta \in \{0, 1\}^*$ where $\mathcal{H}_{hint}$ is the set of all hint queries asked by S, and $\mathcal{V}_{verif}$ is the set of all verification queries asked in the execution of S.

With no loss of generality, we make the assumption that once S made a successful verification query it does not ask any further hint and verification queries.

We define the *success probability* of S as

$$\Pr_{\substack{\delta\in\{0,1\}^* \\ (x,\alpha)\leftarrow\mathcal{D} \\ (\mathcal{H}_{hint},\mathcal{V}_{verif})\leftarrow S(x;\delta))}} \left[\exists(q,r)\in\mathcal{V}_{verif} : q\notin\mathcal{H}_{hint}\wedge V(\alpha,q,r)=1\right]$$

Theorem 5.7 (Hardness amplification for dynamic weakly verifiable puzzles [DIJK09]). *Let $S^{(n)}$ be a probabilistic algorithm for $P^{(n)}$ 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. There exists a probabilistic algorithm S that solves a puzzle defined by P with probability at least $1-\delta$. Furthermore, S makes $O(h(h+v)/\varepsilon)\cdot\log(1/\gamma\delta)$ hint queries and at most one verification query. The running time of S is poly$(h,v,\frac{1}{\varepsilon},t,\omega,\log(1/\gamma\delta))$ where ω is the time needed to make a single hint or verification query.*

It is worth seeing why the approach presented in the previous section that works well for the direct product of WVPs cannot be applied for the direct product of DWVPs (moving aside for a moment the issue of solving only a fraction of puzzles successfully). Loosely speaking, for DWVP the algorithm CHS-solver breaks in the *OnlinePhase* where $S^{(n)}$ can be called multiple times. It is possible that in one of these calls $S^{(n)}$ asks a hint query on q which prevents in one of the later runs a verification query (q,r) from succeeding. The fact that a hint query on q has been asked before makes it impossible to make a successful verification query on this q. Thus, we cannot dismiss a situation where the success probability of $S^{(n)}$ decreases with the number of iterations.

The solution proposed in [DIJK09] is to partition $\mathcal{Q}$ into a set of *attacking queries* $\mathcal{Q}_{attack}$ and a set of *advice queries* $\mathcal{Q}_{adv}$. The idea is to allow a solver to ask hint queries only on $q\in\mathcal{Q}_{adv}$, and to halt the execution whenever a hint query is asked on $q\in\mathcal{Q}_{attack}$. More formally, for a function $hash : \mathcal{Q}\to\{0,1,\ldots,2(h+v)-1\}$ we define $\mathcal{Q}_{attack} := \{q\in\mathcal{Q} : hash(q)=0\}$ and $\mathcal{Q}_{adv} := \{q\in\mathcal{Q} : hash(q)\neq 0\}$.

It is possible, for a fixed solver S that asks at most h hint queries and v verification queries, to find a function $hash : \mathcal{Q}\to\{0,1,\ldots 2(h+v)-1\}$ such that the success probability of S with respect to $\mathcal{Q}_{attack}$ and $\mathcal{Q}_{adv}$ is multiplied by $\frac{1}{8(h+v)}$. If h and v are not too big, then the success probability of S can be still substantial. More formally, the following lemma is proved in [DIJK09].

Lemma 5.8 (Success probability in solving a dynamic weakly verifiable puzzle with respect to a function hash, [DIJK09]). *Let S be a solver for DWVP which success probability is at least δ, 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$(h,v,\frac{1}{\delta},t)$ that outputs a function $hash : \mathcal{Q}\to\{0,1,\ldots,2(h+v)-1\}$ that partitions $\mathcal{Q}$ into $\mathcal{Q}_{attack}$ and $\mathcal{Q}_{adv}$*

such that with probability at least $\frac{\delta}{8(h+v)}$ *the first successful verification query* (q', a) *asked by* S *is such that* $q' \in \mathcal{Q}_{attack}$ *and all previous hint and verification queries have been asked on* $q \in \mathcal{Q}_{adv}$.

A function *hash* can be found by means of a natural sampling technique. We follow exactly the same approach of partitioning $\mathcal{Q}$ in Section 6.1.4.

Let $H_\alpha(q)$ be a polynomial time probabilistic algorithm that takes as input q, has hard-coded α and outputs $H(\alpha, q)$. Similarly, we use $V_\alpha(q, r)$ to denote a polynomial time probabilistic algorithm that computes $V(\alpha, q, r)$ and has hard-coded α. The algorithm DWVP-solver, from [DIJK09], has oracle access to $S^{(n)}$, V_α and H_α as well as a function *hash* as in Lemma 5.8.

Algorithm: DWVP-solver$^{S^{(n)}, hash, H_\alpha^{(n)}, V_\alpha^{(n)}}(x)$

Oracle: A solver $S^{(n)}$ for $P^{(n)}$, algorithms V_α and H_α, a function $hash : \mathcal{Q} \to \{0, 1, \ldots, 2(h+v)-1\}$.
Input: A bistring $x \in \{0,1\}^*$.

Repeat at most $O(\frac{h+v}{\varepsilon} \cdot \log(\frac{1}{\gamma\delta}))$ times
  Let $i \xleftarrow{\$} \{1, \ldots, n\}$ be a position 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 the 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 the 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 $\perp$

In each iteration of the DWVP-solver a position for the input puzzle is chosen uniformly at random. The remaining $(n-1)$ puzzles are generated by the

DWVP-solver, thus it is possible to answer all hint and verification queries for these puzzles. The DWVP-solver calls $S^{(n)}$ multiple times, but the function *hash* is used to partition the query domain such that if a hint query is asked on q for which $hash(q) = 0$ then the current execution of $S^{(n)}$ is aborted, and the DWVP-solver goes to the next iteration. In this way, we ensure that the DWVP-solver never asks a hint query that could prevent a verification query from succeeding. If a verification query is asked on q such that $hash(q) \neq 0$ we answer this query with 0.

Finally, in the case where $S^{(n)}$ asks a verification query on q such that $hash(q) = 0$, a *soft decision system* is used to decide whether to ask a verification query. The idea is that if there are many puzzles among the ones generated by the algorithm that are solved successfully, then it is likely that also the input puzzle is solved successfully. We discount $\gamma\delta n$ to take into account that not all puzzles have to be solved successfully. Detail calculations provided in [DIJK09] show that this approach yields a demanded result.

In Chapter 6 we consider weakly verifiable puzzles that are not only dynamic but also interactive. We use a very similar technique to partition $\mathcal{Q}$ into advice and attacking queries. Instead of the requirement to succeed on at least a fraction of puzzles we consider a more general approach where an arbitrary binary monotone function $g : \{0,1\}^n \to \{0,1\}$ is used to determine on which coordinates the solver has to succeed in order to successfully solve the n-wise direct product of puzzles.

5.3 Interactive Weakly Verifiable Puzzles

Hardness amplification for interactive but non-dynamic weakly verifiable puzzles has been studied by Holenstein and Schoenebeck in [HS11]. We will now give an overview of this work and compare it with our approach.

5.3.1 The definition

Holenstein and Schoenebeck study hardness amplification for puzzles defined as follows.

Definition 5.9 (Interactive Weakly Verifiable Puzzle, [HS11])**.** An *interactive weakly verifiable puzzle* is defined by a protocol given by two probabilistic algorithms P and S. The algorithm P is called a *problem poser* and produces as output a verification circuit Γ. The algorithm S called *a solver* produces no output. Furthermore, the *success probability* of the algorithm S^* in solving an interactive weakly verifiable puzzle defined by (P, S) amounts

$$\Pr_{\substack{\rho\in\{0,1\}^n,\pi\in\{0,1\}^n \\ \Gamma:=\langle P(\rho),S^*(\pi)\rangle_P}}\Big[\Gamma(\langle P(\rho), S^*(\pi)\rangle_{S^*}) = 1\Big].$$

$\diamondsuit$

It is not hard to see that Definition 5.9 is a special case of Definition 3.1. The puzzles in Definition 5.9 are non-dynamic, thus $|Q| = 1$. Furthermore, we can assume that Γ corresponds to the verification circuit from Definition 3.1 and the circuit Γ_H always outputs $\perp$. Finally, at most one verification query is allowed.

5.3.2 The hardness amplification theorem

In this section we briefly describe approach of Holenstein and Schoenebeck used to amplify hardness for interactive weakly verifiable puzzles.

Similarly as in the previous sections, we define the k-wise direct product of puzzles.

Definition 5.10 (k-wise direct product of interactive weakly verifiable puzzles, [HS11])**.** 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) denoted by $(P^{(g)}, S^{(g)})$ is an interactive weakly verifiable puzzle where the problem poser and the solver sequentially interact in k rounds. In each round (P, S) is used to produce an instance of the puzzle. As the result circuits $\Gamma^{(1)}, \ldots, \Gamma^{(k)}$ for P are generated. Finally, $P^{(g)}$ outputs the circuit $\Gamma^{(g)}(y_1, \ldots, y_k) := g(\Gamma^{(1)}(y_1), \ldots, \Gamma^{(k)}(y_k))$. $\diamondsuit$

The following hardness amplification theorem is proved in [HS11].

Theorem 5.11 (Hardness amplification for interactive weakly verifiable puzzles, [HS11])**.** *There exists an algorithm $Gen(C, g, \varepsilon, \delta, n)$ which takes as input a solver circuit C for the k-wise direct product of (P, S), 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

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

where the probability is over the randomness of $P^{(g)}$ and C, then D satisfies almost surely

$$\Pr\left[\Gamma(\langle P, D\rangle_D) = 1\right] \geq \delta + \frac{\varepsilon}{6k},$$

where the probability is over the randomness of P and D. Additionally, Gen and D only require oracle access to g and C. Furthermore, the running time of Gen is $poly(k, \frac{1}{\varepsilon}, n)$ with oracle calls and $Size(D) \leq Size(C) \cdot \frac{6k}{\varepsilon} \log(\frac{6k}{\varepsilon})$.

The above definition does not impose any restrictions on the time complexity of the poser and the solver. The algorithm *Gen* is used to define a polynomial time reduction between a solver for the k-wise direct product of puzzles to a solver for a single puzzle. In the previous sections we considered solvers for the k-wise direct product that must solve all puzzles [CHS05] or allow a fraction of puzzles to be solved incorrectly [DIJK09]. In the above definition a more general approach is considered where a binary monotone function g is used.

In Chapter 6 we use very similar approach as Holenstein and Schoenebeck. The difference is that the hardness amplification theorem proven in this thesis captures dynamic and interactive puzzles.

Chapter 6

Hardness Amplification for Dynamic Interactive Weakly Verifiable Puzzles

In the previous chapter we gave an overview of the former studies of different types of weakly verifiable puzzles. The focus of this chapter is on proving the hardness amplification theorem for dynamic interactive weakly verifiable puzzles. In Section 6.1.1 we remind the hardness amplification theorem stated in Section 3.2. We give the intuition behind the proof of this theorem in Section 6.1.2. The theorem is proved in Section 6.1.3. The proof requires two additional lemmas. In Section 6.1.4 we prove the first one that states that it is possible to efficiently partition the query domain. Next, in Section 6.1.5 we prove the latter lemma that shows that is it possible to amplify hardness for dynamic interactive weakly verifiable puzzles under the assumption that no hint queries prevent verification queries from succeeding. Finally, in Section 6.2.1 we discuss our result.

6.1 Proof of Hardness Amplification

In this section we give a proof of hardness amplification for dynamic interactive weakly verifiable puzzles.

6.1.1 The hardness amplification theorem

Let us remind the experiment *Success* defined in Chapter 3. We denote the problem poser by P and the solver by C. Additionally, let π and ρ be the randomness used by the problem poser and solver, respectively.

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

Oracles: 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

Without loss of generality throughout this chapter we make the assumption that C does not ask verification queries on $q \in \mathcal{Q}$ 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.

We state the hardness amplification theorem for dynamic interactive weakly verifiable puzzles as in Chapter 3.

Theorem 3.3 (Hardness amplification for dynamic interactive 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 $P_n^{(1)}$ as in Definition 3.2. Additionally, let C be a 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 C, a binary monotone function $g : \{0,1\}^k \to \{0,1\}$, and a problem poser $P_n^{(1)}$. The algorithm Gen takes as input parameters ε, δ, n, k, h, v, and outputs a solver circuit D for $P_n^{(1)}$ such that:*
If C satisfies

$$\Pr_{\substack{\pi^{(k)} \in \{0,1\}^{kn} \\ \rho \in \{0,1\}^*}} \left[Success^{P_{kn}^{(g)},C}(\pi^{(k)}, \rho) = 1 \right] \geq 16(h+v)\Big(\Pr_{u \leftarrow \mu_\delta^k}[g(u) = 1] + \varepsilon \Big), \tag{3.2}$$

then D is a two-phase probabilistic circuit that almost surely over the randomness of Gen satisfies

$$\Pr_{\substack{\pi \in \{0,1\}^n \\ \rho \in \{0,1\}^*}} \left[Success^{P_n^{(1)},D}(\pi, \rho) = 1 \right] \geq \delta + \frac{\varepsilon}{6k}. \tag{3.3}$$

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

6.1.2 The intuition behind the proof

In this section we give the intuition behind the proof of Theorem 3.3. We refer to a puzzle solved by $\widetilde{D}$ as an *input puzzle* and fix the notation as in Theorem 3.3. First, we give the intuition behind the part of the proof that bases on [CHS05, HS11]. Then, we describe methods of [DIJK09] used in our proof.

Technique of [CHS05, HS11] The idea is to use the solver circuit C to solve k puzzles among which is the input puzzle, and the remaining $k-1$ puzzles are generated. If a position for the input puzzle is carefully chosen as well as the randomness used to generate the remaining puzzles, then the input puzzle can be solved with the substantial probability almost surely.

More precisely, we try to fix the randomness used to generate a puzzle on the first position such that C satisfies the assumptions of Theorem 3.3 for the $(k-1)$-wise direct product of puzzles (where the fact whether the puzzle on the first position is correctly solved is neglected). The success probability of C on the remaining puzzles can be estimated using a natural sampling technique.

When it is possible to find this randomness, then we recursively solve the subproblem for the $(k-1)$-wise direct product of puzzles. If the recursion reaches $k = 1$, then we can use C directly to solve the input puzzle.

We cannot exclude the situation where in one of the recursive calls we cannot find the randomness for the first position such that the success probability of C for the $(k-1)$-wise direct product of puzzles is substantial. Nevertheless, we know that C has substantial success probability when all positions are considered. Hence, we conclude that the first coordinate is somehow important in the sense that C solves a puzzle on this coordinate unusually often. Therefore, it makes sense to place the input puzzle on this position. All that is left is to find the randomness used to generate puzzles on the remaining positions such that the input puzzle is solved correctly often.

In Section 5.1 we describe a similar approach for the special case where non-interactive puzzles are considered and the function g is such that all puzzles have to be correctly solved. In that section we use Observation 5.4 to conclude that the remaining puzzles should be chosen such that they are all correctly solved by C.

Here we generalize this approach. Namely, we search for the randomness used to generate the remaining $k-1$ puzzles such that when a puzzle on the first po-

sition was solved successfully, then the k-wise direct product of puzzles would be solved successfully, and if a puzzle on the first position was unsuccessfully solved, then the k-wise direct product of puzzles would be also solved unsuccessfully. These are the only cases where solving a puzzle on the first position makes a difference. For a function g which requires all puzzles to be solved correctly this approach corresponds exactly to the one presented in Section 5.1 for weakly verifiable puzzles.

All puzzles except for the input puzzle are generated by Gen or D. Therefore, it is possible to answer all hint and verification queries concerning these puzzles. For the input puzzle to answer hint and verification queries we have to use the hint and verification oracles, respectively.

The above described approach cannot be directly used in the context of dynamic puzzles. After choosing a position for the input puzzle we have to find the randomness from which the remaining puzzles are generated. This requires running C several times. In one of this runs C can ask a hint query which prevents a later verification query from succeeding. Therefore, it may happen that the success probability of C in the consecutive runs decreases.

Technique of [DIJK09] We would like to ensure that no hint query is ever asked that could prevent a verification query from being successful. To satisfy this requirement we use the technique described in [DIJK09]. The set Q is partitioned into two sets. A set of advice queries contains these $q \in \mathcal{Q}$ on which C is allowed to ask hint queries. A set of attacking queries is the set on which C asks successful verification queries. It is possible to use a natural sampling technique to find a function that partitions Q such that the success probability of C is still substantial.

6.1.3 The proof of hardness amplification theorem

In this section we prove Theorem 3.3 and state two additional lemmas that help us to prove this theorem. Informally speaking, Lemma 6.1 states that it is possible to partition $\mathcal{Q}$ such that the solver C has still substantial success probability and there are no conflicting hint and verification queries. In Lemma 6.3 we state that it is possible to amplify hardness for dynamic interactive puzzles under the assumption that there are no conflicting hint and verification queries.

Domain partitioning Let $hash : \mathcal{Q} \to \{0, 1, \ldots, 2(h+v)-1\}$. The idea is to partition $\mathcal{Q}$ such that the set of preimages of 0 for $hash$ contains $q \in \mathcal{Q}$ 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. Let us define the following experiment *CanonicalSuccess* in which $\mathcal{Q}$ is partitioned using a function *hash*. We say that a solver *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 has been asked before (q_j, y_j) such that $hash(q_i) = 0$.

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

Oracle: A problem poser P, a solver circuit $C = (C_1, C_2)$ for P,
a function $hash : \mathcal{Q} \to \{0, 1, \ldots, 2(h + v) - 1\}$.
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

$$\Pr_{\pi,\rho}[CanonicalSuccess^{P,C,hash}(\pi, \rho) = 1]. \tag{6.1}$$

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

We now give Lemma 6.1. Loosely speaking, it states 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* provided that the number of hint and verification queries is not too large. This lemma is analogous to Lemma 4 from [DIJK09].

Lemma 6.1 (Canonical success probability with respect to a function *hash*, [DIJK09])**.** *For a fixed problem poser P_n let C be a solver for P_n with the*

success probability at least γ, asking at most h hint queries and v verification queries. Moreover, let $\mathcal{H}$ be a family of pairwise independent efficient hash functions $\mathcal{Q} \to \{0, 1, \ldots, 2(h+v)-1\}$. 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 $\frac{\gamma}{16(h+v)}$.

We prove Lemma 6.1 in Section 6.1.4.

Hardness amplification Let $C := (C_1, C_2)$ be a two-phase solver circuit 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 oracles. Instead, whenever C_2 asks a hint or verification query, 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 $hash : \mathcal{Q} \to \{0, 1, \ldots, 2(h+v)-1\}$.
Input: Bitstrings $x \in \{0,1\}^*$, $\rho \in \{0,1\}^*$.
Output: A pair (q, y) where $q \in \mathcal{Q}$ and $y \in \{0,1\}^*$.

```
run C_2^(·,·)(x, ρ)
    if C_2^(·,·)(x, ρ) asks a hint query on q then
        if hash(q) = 0 then
            return ⊥
        else
            answer the query of C_2^(·,·)(x, ρ) using Γ_H(q)

    if C_2^(·,·)(x, ρ) asks a verification query (q, y) then
        if hash(q) = 0 then
            return (q, y)
        else
            answer the verification query of C_2^(·,·)(x, ρ) with 0
return ⊥
```

Given $C := (C_1, C_2)$ we define a 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

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

We now show that $\widetilde{C}$ succeeds with probability at least as high as C in the experiment *CanonicalSuccess*.

Lemma 6.2. *For fixed P, $C := (C_1, C_2)$, and hash it holds*

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

Proof of Lemma 6.2. If for some π, ρ, and *hash* the circuit $C := (C_1, C_2)$ succeeds canonically, then for the same π, ρ, and *hash* the circuit $\widetilde{C} := (C_1, \widetilde{C}_2)$ also succeeds. Using this observation we conclude that

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

□

We state the following lemma that is analogous to Theorem 10 from [HS11]. Intuitively, it states that it is possible to amplify hardness where the circuit $\widetilde{C}$, meeting certain criteria, is used.

Lemma 6.3 (Hardness amplification for partitioned domains, [HS11])**.** *Let $g : \{0,1\}^k \to \{0,1\}$ be a binary monotone function, $P_n^{(1)}$ a fixed problem poser and $\widetilde{C} := (C_1, \widetilde{C}_2)$ a probabilistic two-phase circuit with oracle access to a function hash $: \mathcal{Q} \to \{0, 1, \ldots, 2(h + v) - 1\}$ and a solver $C := (C_1, C_2)$ for $P_{kn}^{(g)}$ that asks at most h hint queries and v verification queries. There exists an algorithm $\widetilde{Gen}$ that takes as input parameters ε, δ, n, k, has oracle access to $P_n^{(1)}$, $\widetilde{C}$, hash, g, and outputs a probabilistic two-phase circuit $\widetilde{D} := (D_1, \widetilde{D}_2)$ 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 $\widetilde{D}$ satisfies almost surely over the randomness of $\widetilde{Gen}$

$$\Pr_{\substack{\pi\in\{0,1\}^n,\rho\in\{0,1\}^* \\ x:=\langle P^{(1)}(\pi),D_1^{\widetilde{C}}(\rho)\rangle_{trans} \\ (\Gamma_H,\Gamma_V):=\langle P^{(1)}(\pi),D_1^{\widetilde{C}}(\rho)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2^{P^{(1)},\widetilde{C},hash,g,\Gamma_H}(x,\rho))=1]\geq\delta+\frac{\varepsilon}{6k}.$$

Furthermore, $\widetilde{D}$ asks at most $\frac{6k}{\varepsilon}\log\left(\frac{6k}{\varepsilon}\right)h$ hint queries and no verification queries and outputs a pair (q,y) such that $hash(q)=0$ or $\perp$. Finally, the running time of $\widetilde{Gen}$ is $poly(k,\frac{1}{\varepsilon},n)$ with oracle calls.

We now give the proof of Theorem 3.3 that uses Lemma 6.1 and Lemma 6.3.

Proof of Theorem 3.3. We fix the notation as in Lemma 6.3. Additionally, let Gen and $D:=(D_1,D_2)$ be as in Theorem 3.3. In the following code listing we define Gen that outputs $D:=(D_1,D_2)$. The circuit D_2 is defined in the succeeding code excerpt.

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

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

Let γ be the success probability of C.
$hash:=\mathrm{FindHash}^{P^{(g)},C}(\gamma,n,h,v)$
Let $\widetilde{C}:=(C_1,\widetilde{C}_2)$ be as in Lemma 6.2 with oracle access to C and $hash$.
$\widetilde{D}:=\widetilde{\mathrm{Gen}}^{P^{(1)},\widetilde{C},g,hash}(\varepsilon,\delta,n,k)$
return $D:=(D_1,D_2)$ // *where D has oracle access to $\widetilde{D}_2,hash,g$*

Circuit $D_2^{\widetilde{D},P^{(1)},hash,g,\Gamma_V,\Gamma_H}(x,\rho)$

Oracle: A circuit $\widetilde{D}:=(D_1,\widetilde{D}_2)$ from Lemma 6.3, 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 $x\in\{0,1\}^*$, $\rho\in\{0,1\}^*$.

$(q,y):=\widetilde{D}_2^{P^{(1)},\widetilde{C},hash,g,\Gamma_H}(x,\rho)$
Ask verification query (q,y) to Γ_V.

We consider a solver circuit $C := (C_1, C_2)$ that asks at most h hint queries and v verification queries and satisfies

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

The algorithm Gen can use FindHash to obtain $hash : \mathcal{Q} \to \{0,1,\ldots,2(h+v)-1\}$ such that almost surely over the randomness of FindHash it holds

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

The running time of FindHash is $poly(h, v, \frac{1}{\varepsilon}, n)$ with oracle calls. Applying Lemma 6.2 for $C = (C_1, C_2)$ we obtain a circuit $\widetilde{C} = (C_1, \widetilde{C}_2)$ that satisfies

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

We use the algorithm $\widetilde{\text{Gen}}$ as in Lemma 6.3 that yields a circuit $\widetilde{D} := (D_1, \widetilde{D}_2)$ and almost surely over the randomness of $\widetilde{\text{Gen}}$ satisfies

$$\Pr_{\substack{\pi,\rho\\ x:=\langle P^{(1)}(\pi),D_1^{\widetilde{C}}(\rho)\rangle_{trans}\\ (\Gamma_H,\Gamma_V):=\langle P^{(1)}(\pi),D_1^{\widetilde{C}}(\rho)\rangle_{P^{(1)}}}}[\Gamma_V(D_2^{P^{(1)},\widetilde{C},hash,g,\Gamma_H}(x,\rho))=1] \geq \delta+\frac{\varepsilon}{6k}. \tag{6.2}$$

The running time of Gen is $poly(k, \frac{1}{\varepsilon}, h, v, n)$ with oracle access. Finally, Gen outputs $D := (D_1, D_2)$ with oracle access to $\widetilde{D}_2$, $P^{(1)}$, $hash$, g such that almost surely over the randomness of Gen it holds

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

The running time of $\widetilde{\text{Gen}}$ is $poly(k, \frac{1}{\varepsilon}, n)$ with oracle access

The circuit D asks at most $\frac{6k}{\varepsilon}\log(\frac{6k}{\varepsilon})h$ hint queries and one verification query and $Size(D) \leq Size(C) \cdot \frac{6k}{\varepsilon}$. This finishes the proof of Theorem 3.3. □

6.1.4 Domain partitioning

In this section we use the technique from [DIJK09] to partition $\mathcal{Q}$ into a set $\mathcal{Q}_{adv}$ of advice queries and a set $\mathcal{Q}_{attack}$ of attacking queries. The circuit C is allowed to make hint queries on $q \in \mathcal{Q}_{adv}$ and successful verification queries on $q \in \mathcal{Q}_{attack}$. If C asks a hint query on $q \in \mathcal{Q}_{attack}$, the execution of C is aborted. Similarly, all verification queries on $q \in \mathcal{Q}_{adv}$ are answered as unsuccessful. The idea is to avoid collisions between hint and verification queries in the independent runs of C. Let us now prove Lemma 6.1.

Proof of Lemma 6.1. We fix a problem poser P and a solver C for P in the whole proof of Lemma 6.1. For $k, l \in \{1, \ldots, (h+v)\}$ and $\alpha, \beta \in \{0, 1, \ldots, 2(h+v) - 1\}$ by the pairwise independence property of $\mathcal{H}$ we have that for every $q_k, q_l \in \mathcal{Q}$ such that $q_k \neq q_l$ the following holds

$$\begin{aligned}\Pr_{hash \leftarrow \mathcal{H}}[hash(q_k) = \alpha \mid hash(q_l) = \beta] &= \Pr_{hash \leftarrow \mathcal{H}}[hash(q_k) = \alpha] \\ &= \frac{1}{2(h+v)}. \end{aligned} \tag{6.3}$$

Let $\mathcal{P}_{Success}$ denote a set containing (π, ρ) for which $Success^{P,C}(\pi, \rho) = 1$. We are interested in probability over the choice of *hash* of the event $CanonicalSuccess^{P,C,hash}(\pi^*, \rho^*) = 1$ where $(\pi^*, \rho^*) \in \mathcal{P}_{Success}$ is fixed. Let (q_j, y_j) denote the first verification query for which C succeeds. We calculate this probability

$$\begin{aligned}
&\Pr_{hash \leftarrow \mathcal{H}}\Big[CanonicalSuccess^{P,C,hash}(\pi^*, \rho^*) = 1\Big] \\
&\quad = \Pr_{hash \leftarrow \mathcal{H}}\big[\, hash(q_j) = 0 \wedge (\forall i < j : hash(q_i) \neq 0)\big] \\
&\quad = \Pr_{hash \leftarrow \mathcal{H}}\big[\forall i < j : hash(q_i) \neq 0 \mid hash(q_j) = 0\big] \Pr_{hash \leftarrow \mathcal{H}}[hash(q_j) = 0] \\
&\quad \overset{(6.3)}{=} \frac{1}{2(h+v)}\Big(1 - \Pr_{hash \leftarrow \mathcal{H}}[\exists i < j : hash(q_i) = 0 \mid hash(q_j) = 0]\Big) \\
&\quad \overset{(*)}{\geq} \frac{1}{2(h+v)}\Big(1 - \sum_{i<j} \Pr_{hash \leftarrow \mathcal{H}}[hash(q_i) = 0 | \, hash(q_j) = 0]\Big) \\
&\quad \overset{(6.3)}{=} \frac{1}{2(h+v)}\Big(1 - \sum_{i<j} \Pr_{hash \leftarrow \mathcal{H}}[hash(q_i) = 0]\Big) \\
&\quad \overset{(6.3)}{\geq} \frac{1}{4(h+v)},
\end{aligned} \tag{6.4}$$

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

$$\begin{aligned}
&\Pr_{\substack{hash \leftarrow \mathcal{H} \\ \pi, \rho}}\Big[CanonicalSuccess^{P,C,hash}(\pi, \rho) = 1\Big] \\
&\quad = \Pr_{\substack{hash \leftarrow \mathcal{H} \\ \pi, \rho}}\Big[CanonicalSuccess^{P,C,hash}(\pi, \rho) = 1 \mid (\pi, \rho) \in \mathcal{P}_{Success}\Big] \Pr_{\pi, \rho}[(\pi, \rho) \in \mathcal{P}_{Success}] \\
&\qquad + \Pr_{\substack{hash \leftarrow \mathcal{H} \\ \pi, \rho}}\Big[CanonicalSuccess^{P,C,hash}(\pi, \rho) = 1 \mid (\pi, \rho) \notin \mathcal{P}_{Success}\Big] \Pr_{\pi, \rho}[(\pi, \rho) \notin \mathcal{P}_{Success}] \\
&\quad = \Pr_{\substack{hash \leftarrow \mathcal{H} \\ \pi, \rho}}\Big[CanonicalSuccess^{P,C,hash}(\pi, \rho) = 1 \mid (\pi, \rho) \in \mathcal{P}_{Success}\Big] \Pr_{\pi, \rho}[(\pi, \rho) \in \mathcal{P}_{Success}]
\end{aligned}$$

$$
\begin{aligned}
&\geq \Pr_{\substack{hash \leftarrow \mathcal{H} \\ \pi,\rho}} \left[CanonicalSuccess^{P,C,hash}(\pi,\rho) = 1 \mid (\pi,\rho) \in \mathcal{P}_{Success} \right] \cdot \gamma \\
&= \mathop{\mathbb{E}}_{(\pi,\rho)\in\mathcal{P}_{Success}} \left[\Pr_{hash \leftarrow \mathcal{H}} [CanonicalSuccess^{P,C,hash}(\pi,\rho) = 1] \right] \cdot \gamma \\
&\overset{(6.4)}{\geq} \frac{\gamma}{4(h+v)}. \qquad (6.5)
\end{aligned}
$$

The following algorithm FindHash uses a natural sampling approach to pick a function *hash* that satisfies Lemma 6.1.

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

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

```
for i := 1 to 32(h+v)^2/γ^2 n do:
    hash ←$ H
    count := 0
    for j := 1 to N := 32(h+v)^2/γ^2 n 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)) 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

$$\Pr_{\pi,\rho} \left[CanonicalSuccess^{P,C,hash}(\pi,\rho) = 1 \right] \geq \frac{\gamma}{8(h+v)}, \qquad (6.6)$$

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

$$\Pr_{\pi,\rho} \left[CanonicalSuccess^{P,C,hash}(\pi,\rho) = 1 \right] \leq \frac{\gamma}{16(h+v)}. \qquad (6.7)$$

Let N be 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.

Let us define independent and 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 where $\mathit{hash} \in \mathcal{H}_{Bad}$ and show that it is unlikely that $\mathit{hash} \in \mathcal{H}_{Bad}$ is returned by FindHash. From (6.7) it follows that $\mathbb{E}_{\pi,\rho}[X_i] \leq \frac{\gamma}{16(h+v)}$. In the following inequalities (6.8) and (6.9) in the steps denoted with $(*)$ we use the trivial facts $h + v \geq 1$ and $\gamma \leq 1$. For fixed $\mathit{hash} \in \mathcal{H}_{Bad}$ using the Chernoff bound we get

$$\begin{aligned} \Pr_{\pi,\rho}\left[\frac{1}{N}\sum_{i=1}^{N} X_i \geq \frac{\gamma}{12(h+v)}\right] &\leq \Pr_{\pi,\rho}\left[\frac{1}{N}\sum_{i=1}^{N} X_i \geq \big(1+\frac{1}{3}\big)\mathbb{E}[X_i]\right] \\ &\leq e^{-\frac{\gamma}{16(h+v)}N/27} \leq e^{-\frac{2}{27}\frac{(h+v)}{\gamma}n} \overset{(*)}{\leq} e^{-\frac{2}{27}n}. \end{aligned} \tag{6.8}$$

The probability that $\mathit{hash} \in \mathcal{H}_{Good}$ when picked is not returned amounts

$$\begin{aligned} \Pr_{\pi,\rho}\left[\frac{1}{N}\sum_{i=1}^{N} X_i \leq \frac{\gamma}{12(h+v)}\right] &\leq \Pr_{\pi,\rho}\left[\frac{1}{N}\sum_{i=1}^{N} X_i \leq \big(1-\frac{1}{3}\big)\mathbb{E}[X_i]\right] \\ &\leq e^{-\frac{\gamma}{8(h+v)}N/18} \leq e^{-\frac{2}{9}\frac{(h+v)}{\gamma}n} \overset{(*)}{\leq} e^{-\frac{2}{9}n} \end{aligned} \tag{6.9}$$

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

$$\Pr_{\mathit{hash} \leftarrow \mathcal{H}}[\mathit{hash} \in \mathcal{H}_{Good}] < \frac{\gamma}{8(h+v)}. \tag{6.10}$$

Using (6.10) we conclude that it is possible to bound the probability of the canonical success as follows

$$\begin{aligned} &\Pr_{\substack{\mathit{hash} \leftarrow \mathcal{H} \\ \pi,\rho}}[\mathit{CanonicalSuccess}^{P,C,\mathit{hash}}(\pi,\rho) = 1] \\ &\quad = \Pr_{\substack{\mathit{hash} \leftarrow \mathcal{H} \\ \pi,\rho}}[\mathit{CanonicalSuccess}^{P,C,\mathit{hash}}(\pi,\rho) = 1 \mid \mathit{hash} \in \mathcal{H}_{Good}] \Pr_{\mathit{hash} \leftarrow \mathcal{H}}[\mathit{hash} \in \mathcal{H}_{Good}] \\ &\qquad + \Pr_{\substack{\mathit{hash} \leftarrow \mathcal{H} \\ \pi,\rho}}[\mathit{CanonicalSuccess}^{P,C,\mathit{hash}}(\pi,\rho) = 1 \mid \mathit{hash} \notin \mathcal{H}_{Good}] \Pr_{\mathit{hash} \leftarrow \mathcal{H}}[\mathit{hash} \notin \mathcal{H}_{Good}] \\ &\quad \leq \Pr_{\mathit{hash} \leftarrow \mathcal{H}}[\mathit{hash} \in \mathcal{H}_{Good}] + \Pr_{\substack{\mathit{hash} \leftarrow \mathcal{H} \\ \pi,\rho}}[\mathit{CanonicalSuccess}^{P,C,\mathit{hash}}(\pi,\rho) = 1 \mid \mathit{hash} \notin \mathcal{H}_{Good}] \\ &\quad \overset{\substack{(6.6) \\ (6.10)}}{<} \frac{\gamma}{8(h+v)} + \frac{\gamma}{8(h+v)} = \frac{\gamma}{4(h+v)}, \end{aligned}$$

which contradicts (6.5). Hence, we conclude that the probability of choosing $hash \in \mathcal{H}_{Good}$ is at least $\frac{\gamma}{8(h+v)}$.

We now show that FindHash picks in one of its iteration $hash \in \mathcal{H}_{Good}$ almost surely. Let K be the random variable that denotes the number of iterations of the outer loop of FindHash and Y_i be the binary 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 $K \leq 32n(h+v)^2/\gamma^2$ to 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 the running time of FindHash is $poly(n, h, v, \gamma)$ with oracle calls. This finishes the proof of Lemma 6.1. □

6.1.5 The hardness amplification proof for partitioned domains

In this section we give the proof of Lemma 6.3 which in a great extent bases on [HS11].

We need to introduce some additional algorithms and circuits before giving the proof of Lemma 6.3 . The following code listing depicts the algorithm $\widetilde{\text{Gen}}$. The algorithm EstimateSurplus and the circuits $\widetilde{D}$, $\widetilde{C}' := (C'_1, \widetilde{C}'_2)$ used by $\widetilde{Gen}$ are presented on the succeeding code excerpts. Intuitively, the algorithm EstimateSurplus returns the estimate of the advantage of $\widetilde{C}$ over the algorithm that solves each puzzle independently with probability δ where the randomness used to generate a puzzle on the first position is fixed to π^* and the fact of solving this puzzle is disregarded. The circuit $\widetilde{D}$ outputs a solution (q, y) for a puzzle defined by $P^{(1)}$. The circuit $\widetilde{C}'$ is a solver for the $(k-1)$-wise direct product of puzzles that outputs an answer $(q, y_2, \ldots, y_k)$.

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

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

---

for $i := 1$ **to** $\frac{6k}{\varepsilon}n$ **do:**
  $\pi^* \xleftarrow{\$} \{0,1\}^n$
  $\widetilde{S}_{\pi^*,0} := \text{EstimateSurplus}^{P^{(1)}, \widetilde{C}, g, hash}(\pi^*, 0, k, \varepsilon, \delta, n)$

$\widetilde{S}_{\pi^*,1} := \text{EstimateSurplus}^{P^{(1)},\widetilde{C},g,hash}(\pi^*, 1, k, \varepsilon, \delta, n)$
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^*.
$\widetilde{C}' := (C_1', \widetilde{C}_2')$
$g'(b_2, \ldots, b_k) := g(b, b_2, \ldots, b_k)$
return $\widetilde{Gen}^{P^{(1)},\widetilde{C}',g',hash}(\varepsilon, \delta, n, k-1)$
// all estimates are lower than $(1 - \frac{3}{4k})\varepsilon$
return $\widetilde{D}^{P^{(1)},\widetilde{C},hash,g}$

The algorithm EstimateSurplus uses procedures EstimateFunctionProbability and EvalutePuzzles. We state them first before EstimateSurplus.

We are interested in the probability that for $u \leftarrow \mu_\delta^k$ and $b \in \{0,1\}$ we have $g(b, u_2, \ldots, u_k) = 1$. 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 6.4. *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 code listing of EstimateFunctionProbability. Let us define independent and 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

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

□

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 of $P^{(1)}$ 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 $P^{(1)}(\pi_i)$ to simulate the interaction with $C_1(\rho)$.

Let us introduce some additional notation. We write $\langle P^{(1)}(\pi_i), C_1(\rho)\rangle^i$ to denote the i-th round of the sequential interaction. Let $\langle P^{(1)}(\pi_i), C_1(\rho)\rangle^i_{P^{(1)}}$ be 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 if all previous rounds have been executed before.

For simplicity of the notation in the code excerpts of circuits EvalutePuzzles, $\widetilde{C}_2'$, $\widetilde{D}_2$ we omit superscripts of some oracles. Exemplary, in the code listing of EvalutePuzzles for $\widetilde{C}_2^{\Gamma_H^{(k)},C,hash}$ we omit C in the superscript and instead write $\widetilde{C}_2^{\Gamma_H^{(k)},hash}$. We make sure that it is clear from the context which oracles are used.

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

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

---

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

All puzzles are generated by EvalutePuzzles. Thus, the algorithm can answer 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 puzzle placed on the first position is neglected and instead $b \in \{0,1\}$ is used. More formally, we introduce the notion of a surplus $S_{\pi^*,b}$ defined as

$$S_{\pi^*,b} = \Pr_{\pi^{(k)},\rho}\left[g(b, c_2, \ldots, c_k) = 1 \mid \pi_1 = \pi^*\right] - \Pr_{u \leftarrow \mu_\delta^k}\left[g(b, u_2, \ldots, u_k) = 1\right], \tag{6.11}$$

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

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

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

Oracle: A problem poser $P^{(1)}$, a circuit $\widetilde{C}$ for $P^{(g)}$, functions $g : \{0,1\}^k \to \{0,1\}$ and $hash : \mathcal{Q} \to \{0, 1, \ldots, 2(h+v)-1\}$.
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 6.5. *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 6.4 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 6.4 we conclude that the surplus estimate returned by EstimateSurplus differs from $S_{\pi^*,b}$ by at most $\frac{\varepsilon}{4k}$ with probability at least $1 - 2e^{-n}$. □

We define the following circuit $\widetilde{C}' = (C'_1, \widetilde{C}'_2)$ for the $(k-1)$-wise direct product of $P^{(1)}$. Loosely speaking, in the recursive calls of $\widetilde{\text{Gen}}$ we fix the randomness used to generate the puzzle on the first position. The circuit $\widetilde{C}'$ uses the circuit $\widetilde{C}$ for the k-wise direct product where the first puzzle is generated internally using the fixed randomness.

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 $hash : \mathcal{Q} \to \{0, 1, \ldots, 2(h+v)-1\}$.
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 $\pi^* \in \{0,1\}^n$.
Output: A tuple $(q, y_2, \ldots, y_k)$.

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 $(q, y_2, \ldots, y_k)$

Finally, we define the solver circuit $\widetilde{D} = (D_1, \widetilde{D}_2)$ for $P^{(1)}$.

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 $\widetilde{D}_2^{P^{(1)},\widetilde{C},hash,g,\Gamma_H}(x^*, r)$

Oracle: A poser $P^{(1)}$, a solver circuit $\widetilde{C} = (C_1, \widetilde{C}_2)$ for $P^{(g)}$, functions $hash : \mathcal{Q} \to \{0, 1, \ldots, 2(h+v)-1\}$, $g : \{0,1\}^k \to \{0,1\}$, 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 interaction 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 $\perp$

Proof of Lemma 6.3, [HS11]. 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. In the latter case, where g is a constant function, Lemma 6.3 is vacuously true. If g is the identity function, then the circuit $\widetilde{D}$ directly uses $\widetilde{C}$ to find a solution. From the assumptions of Lemma 6.3 it follows that $\widetilde{C}$ succeeds with probability at least $\delta + \varepsilon$. Hence, $\widetilde{D}$ trivially satisfies Lemma 6.3.

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

If it is possible to find an estimate with the high surplus, then we introduce a new monotone function $g' : \{0,1\}^{k-1} \to \{0,1\}$ such that $g'(b_2, \ldots, b_k) := g(b, b_2, \ldots, b_k)$ and a new circuit $\widetilde{C}' = (C_1', \widetilde{C}_2')$ with oracle access to $\widetilde{C}$. We apply Lemma 6.5 and conclude that almost surely it holds

$$S_{\pi^*,b} \geq \widetilde{S}_{\pi^*,b} - \frac{\varepsilon}{4k} \geq \Big(1 - \frac{1}{k}\Big)\varepsilon.$$

It follows that $\widetilde{C}'$ succeeds in solving the $(k-1)$–wise direct product of puzzles with probability at least

$$\Pr_{u \leftarrow \mu_\delta^{(k-1)}}[g'(u_1, \ldots, u_{k-1})] + \Big(1 - \frac{1}{k}\Big)\varepsilon.$$

In this case $\widetilde{C}'$ satisfies the conditions of Lemma 6.3 for the $(k-1)$-wise direct product of puzzles. The recursive call to $\widetilde{\mathrm{Gen}}$, which has oracle access to g' and $\widetilde{C}$, returns $\widetilde{D} = (D_1, \widetilde{D}_2)$ that with high probability satisfies

$$\begin{aligned}\Pr_{\substack{\pi,\rho \\ x:=\langle P^{(1)}(\pi), D_1^{\widetilde{C}}(\rho)\rangle_{trans} \\ (\Gamma_H,\Gamma_V):=\langle P^{(1)}(\pi), D_1^{\widetilde{C}}(\rho)\rangle_{P^{(1)}}}}\Big[\Gamma_V\big(\widetilde{D}_2^{P^{(1)},\widetilde{C},hash,g,\Gamma_H}(x,\rho)\big) = 1\Big] &\geq \delta + \Big(1 - \frac{1}{k}\Big)\frac{\varepsilon}{6(k-1)} \\ &= \delta + \frac{\varepsilon}{6k}. \end{aligned} \tag{6.12}$$

Let us bring our attention to the case where none of the estimates is greater than $(1 - \frac{3}{4k})\varepsilon$. If all surpluses $S_{\pi,0}$ and $S_{\pi,1}$ were lower than $(1 - \frac{1}{k})\varepsilon$, then it would mean that $\widetilde{C}$ does not succeed on the remaining $k-1$ puzzles with much higher probability than an algorithm that solves each puzzle independently with success probability δ. However, from the assumptions of Lemma 6.3 we know that on all k puzzles the success probability of $\widetilde{C}$ is higher. Hence, we suspect that the first puzzle is correctly solved unusually often. It remains to show that the fact that $\widetilde{\mathrm{Gen}}$ fails to find a surplus estimate that is large implies that with high probability there are only few surpluses greater than $(1 - \frac{1}{k})\varepsilon$ and their influence can be neglected. Additionally, we have to show that the circuit $\widetilde{D}$ outputs an answer almost surely.

The proof in this case requires lengthly probability manipulations. In Appendix A.2 we give a shorter proof under the simplifying assumptions that the circuit $\widetilde{D}$ always outputs an answer and all surpluses are low.

We fix the notation as in the code listing of $\widetilde{D}_2$. Let us consider a single iteration of the outer loop of $\widetilde{D}_2$ where values $\pi_2, \ldots, \pi_k$ are fixed. We write π_1 to denote the randomness used by the problem poser to generate the input puzzle. Furthermore, we 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. We write $c := (c_1, c_2, \ldots, c_k)$, and for $b \in \{0, 1\}$ we define a set

$$\mathcal{G}_b \;:=\; \big\{(b_1, b_2, \ldots, b_k) : g(b, b_2, \ldots, b_k) = 1\big\}.$$

Using this notation we express

$$\begin{aligned}\Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_b] &= \Pr_{u \leftarrow \mu_\delta^k}[g(b, u_2, \ldots, u_k) = 1] \\ \Pr_{\pi^{(k)},\rho}[c \in \mathcal{G}_b] &= \Pr_{\pi^{(k)},\rho}[g(b, c_2, \ldots, c_k) = 1]. \end{aligned} \tag{6.13}$$

Let us fix the randomness π_1 to π^*. We use (6.11) and (6.13) to obtain

$$\begin{aligned}&\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1]-\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_0]\\&=\Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_1\mid\pi_1=\pi^*]-\Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_0\mid\pi_1=\pi^*]-(S_{\pi^*,1}-S_{\pi^*,0})\end{aligned}\tag{6.14}$$

By monotonicity of g it holds $\mathcal{G}_0\subseteq\mathcal{G}_1$, and we write (6.14) as

$$\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]=\Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_1\setminus\mathcal{G}_0\mid\pi_1=\pi^*]-(S_{\pi^*,1}-S_{\pi^*,0}).\tag{6.15}$$

Let us multiply both sides of (6.15) by

$$\Pr_{\substack{r\\x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\(\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x^*,r))=1]\ /\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0],$$

which yields

$$\begin{aligned}&\Pr_{\substack{r\\x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\(\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x^*,r))=1]\\&=\Pr_{\substack{r\\x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\(\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x^*,r))=1]\Pr_{\pi^{(k)},\rho}[c\in\mathcal{G}_1\setminus\mathcal{G}_0\mid\pi_1=\pi^*]\frac{1}{\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]}\\&\quad-\Pr_{\substack{r\\x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\(\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x^*,r))=1](S_{\pi^*,1}-S_{\pi^*,0})\frac{1}{\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]}.\end{aligned}\tag{6.16}$$

Let us study the first summand of (6.16). First, we have

$$\begin{aligned}&\Pr_{\substack{r\\x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\(\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x^*,r))=1]\\&=\Pr_{\substack{r\\x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\(\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x^*,r))=1|\widetilde{D}_2(x^*,r)\neq\perp]\Pr_{\substack{r\\x^*=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}}}[\widetilde{D}_2(x^*,r)\neq\perp]\\&\overset{(*)}{=}\Pr_{\pi^{(k)},\rho}[c_1=1\mid c\in\mathcal{G}_1\setminus\mathcal{G}_0,\pi_1=\pi^*]\Pr_{\substack{r\\x^*=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}}}[\widetilde{D}_2(x^*,r)\neq\perp]\end{aligned}\tag{6.17}$$

where in $(*)$ we use the observation that $\widetilde{D}_2(x^*,r)\neq\perp$ happens if and only if $\widetilde{D}_2(x^*,r)$ 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 $\Gamma_V(\widetilde{D}_2(x^*, r)) = 1$ happens if and only if $c_1 = 1$. Inserting (6.17) to the numerator of the first summand of (6.16) yields

$$\begin{aligned}
&\Pr_{\substack{r \\ x^* := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{trans} \\ (\Gamma_V, \Gamma_H) := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{P^{(1)}}}} [\Gamma_V(\widetilde{D}_2(x^*, r)) = 1] \Pr_{\pi^{(k)}, \rho} [c \in \mathcal{G}_1 \setminus \mathcal{G}_0 \mid \pi_1 = \pi^*] \\
&\qquad = \Pr_{\substack{r \\ x^* = \langle P^{(1)}(\pi^*), D_1(r)\rangle_{trans}}} [\widetilde{D}_2(x^*, r) \neq \perp] \Pr_{\pi^{(k)}, \rho} [c_1 = 1 \mid c \in \mathcal{G}_1 \setminus \mathcal{G}_0, \pi_1 = \pi^*] \Pr_{\pi^{(k)}, \rho} [c \in \mathcal{G}_1 \setminus \mathcal{G}_0 \mid \pi_1 = \pi^*].
\end{aligned} \tag{6.18}$$

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

$$\Pr_{\pi^{(k)}, \rho} [c_1 = 1 \mid c \in \mathcal{G}_1 \setminus \mathcal{G}_0, \pi_1 = \pi^*] \Pr_{\pi^{(k)}, \rho} [c \in \mathcal{G}_1 \setminus \mathcal{G}_0 \mid \pi_1 = \pi^*] \leq \frac{\varepsilon}{6k}. \tag{6.19}$$

In case where $\Pr_{\pi^{(k)}, \rho}[c \in \mathcal{G}_1 \setminus \mathcal{G}_0 \mid \pi_1 = \pi^*] > \frac{\varepsilon}{6k}$ the circuit $\widetilde{D}_2$ 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

$$\Pr_{\substack{r \\ x^* := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{trans}}} [\widetilde{D}_2(x^*, r) = \perp] \leq \left(1 - \frac{\varepsilon}{6k}\right)^{\frac{6k}{\varepsilon} \log(\frac{6k}{\varepsilon})} \leq \frac{\varepsilon}{6k}. \tag{6.20}$$

We conclude that in both aforementioned cases using (6.18), (6.19) and (6.20) the following holds

$$\begin{aligned}
&\Pr_{\substack{r \\ x^* := \langle P^{(1)}(\pi^*), D_1(r)\rangle_{trans}}} [\widetilde{D}_2(x^*, r) \neq \perp] \Pr_{\pi^{(k)}, \rho} [c_1 = 1 \mid c \in \mathcal{G}_1 \setminus \mathcal{G}_0, \pi_1 = \pi^*] \Pr_{\pi^{(k)}, \rho} [c \in \mathcal{G}_1 \setminus \mathcal{G}_0 \mid \pi_1 = \pi^*] \\
&\qquad \geq \Pr_{\pi^{(k)}, \rho} [c_1 = 1 \mid c \in \mathcal{G}_1 \setminus \mathcal{G}_0, \pi_1 = \pi^*] \Pr_{\pi^{(k)}, \rho} [c \in \mathcal{G}_1 \setminus \mathcal{G}_0 \mid \pi_1 = \pi^*] - \frac{\varepsilon}{6k} \\
&\qquad = \Pr_{\pi^{(k)}, \rho} [c_1 = 1 \wedge c \in \mathcal{G}_1 \setminus \mathcal{G}_0 \mid \pi_1 = \pi^*] - \frac{\varepsilon}{6k} \\
&\qquad = \Pr_{\pi^{(k)}, \rho} [g(c) = 1 \mid \pi_1 = \pi^*] - \Pr_{\pi^{(k)}, \rho} [c \in \mathcal{G}_0 \mid \pi_1 = \pi^*] - \frac{\varepsilon}{6k} \\
&\qquad \overset{(6.11)}{=} \Pr_{\pi^{(k)}, \rho} [g(c) = 1 \mid \pi_1 = \pi^*] - \Pr_{u \leftarrow \mu_\delta^{(k)}} [u \in \mathcal{G}_0] - S_{\pi^*, 0} - \frac{\varepsilon}{6k}.
\end{aligned} \tag{6.21}$$

We insert (6.21) into (6.16) and calculate the expected value over π^* which

yields

$$\Pr_{\substack{\pi,r\\ x:=\langle P^{(1)}(\pi),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x,r))=1] \geq \mathbb{E}_{\pi^*}\left[\frac{\Pr_{\pi^{(k)},\rho}[g(c)=1|\pi_1=\pi^*]-\Pr_{u\leftarrow\mu_\delta^{(k)}}[u\in\mathcal{G}_0]-\frac{\varepsilon}{6k}}{\Pr_{u\leftarrow\mu_\delta^{(k)}}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]}\right]$$
$$-\mathbb{E}_{\pi^*}\left[\left(\Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x^*,r))=1](S_{\pi^*,1}-S_{\pi^*,0})+S_{\pi^*,0}\right)\frac{1}{\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]}\right]. \tag{6.22}$$

We now show that if $\widetilde{\text{Gen}}$ does not recurse, then the majority of estimates is low almost surely. Let us assume that

$$\Pr_{\pi}\left[\left(S_{\pi,0}\leq(1-\frac{1}{2k})\varepsilon\right)\wedge\left(S_{\pi,1}\leq(1-\frac{1}{2k})\varepsilon\right)\right]<1-\frac{\varepsilon}{6k}, \tag{6.23}$$

then $\widetilde{\text{Gen}}$ recurses almost surely, because the probability that $\widetilde{\text{Gen}}$ 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 6.5. Therefore, under the assumption that $\widetilde{\text{Gen}}$ does not recurse with high probability it holds

$$\Pr_{\pi,\rho}\left[\left(S_{\pi,0}\leq(1-\frac{1}{2k})\varepsilon\right)\wedge\left(S_{\pi,1}\leq(1-\frac{1}{2k})\varepsilon\right)\right]\geq 1-\frac{\varepsilon}{6k}. \tag{6.24}$$

Let us define a set

$$\mathcal{W}=\left\{\pi:\left(S_{\pi,0}\leq(1-\frac{1}{2k})\varepsilon\right)\wedge\left(S_{\pi,1}\leq(1-\frac{1}{2k})\varepsilon\right)\right\}. \tag{6.25}$$

Additionally, let $\overline{\mathcal{W}}$ denote the complement of $\mathcal{W}$. We bound the numerator of the second summand in (6.22)

$$\mathbb{E}_{\pi^*}\left[S_{\pi^*,0}+\Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x^*,r))=1](S_{\pi^*,1}-S_{\pi^*,0})\right]$$
$$=\mathbb{E}_{\pi^*}\left[S_{\pi^*,0}+\Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x^*,r)=1](S_{\pi^*,1}-S_{\pi^*,0})\;\middle|\;\pi^*\in\overline{\mathcal{W}}\right]\Pr_{\pi^*}[\pi^*\in\overline{\mathcal{W}}]$$
$$+\mathbb{E}_{\pi^*}\left[S_{\pi^*,0}+\Pr_{\substack{r\\ x^*:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans}\\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x^*,r))=1](S_{\pi^*,1}-S_{\pi^*,0})\;\middle|\;\pi^*\in\mathcal{W}\right]\Pr_{\pi^*}[\pi^*\in\mathcal{W}]$$

$$\leq \frac{\varepsilon}{6k} + \mathbb{E}_{\pi^*}\Big[S_{\pi^*,0} + \Pr_{\substack{r \\ x:=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{trans} \\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi^*),D_1(r)\rangle_{P^{(1)}}}}\big[\Gamma_V(\widetilde{D}_2^{\widetilde{C}}(x^*,r)) = 1\big]\big((1-\frac{1}{2k})\varepsilon - S_{\pi^*,0}\big) \mid \pi^* \in \mathcal{W}\Big]$$

$$\leq \frac{\varepsilon}{6k} + (1-\frac{1}{2k})\varepsilon = (1-\frac{1}{3k})\varepsilon. \tag{6.26}$$

Finally, we insert (6.26) into (6.22) which yields

$$\Pr_{\substack{\pi,\rho \\ x:=\langle P^{(1)}(\pi),D_1(\rho)\rangle_{\text{trans}} \\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi),D_1(\rho)\rangle_{P^{(1)}}}}\big[\Gamma_V(\widetilde{D}_2(x,\rho)) = 1\big] \geq \mathop{\mathbb{E}}_{\pi^*}\left[\frac{\Pr_{\pi^{(k)},\rho}[g(c)=1 \mid \pi_1=\pi^*] - \Pr_{u\leftarrow\mu_\delta^k}[u\in G_0] - (1-\frac{1}{6k})\varepsilon}{\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]}\right].$$

From the assumptions of Lemma 6.3 it follows that

$$\Pr_{\pi^{(k)},\rho}[g(c)=1] \geq \Pr_{u\leftarrow\mu_\delta^{(k)}}[g(u)=1] + \varepsilon. \tag{6.27}$$

We observe that

$$\begin{aligned}\Pr_{u\leftarrow\mu_\delta^k}[g(u)=1] &= \Pr[u\in\mathcal{G}_0 \vee (u\in\mathcal{G}_1\setminus\mathcal{G}_0 \wedge u_1=1)] \\ &= \Pr[u\in\mathcal{G}_0] + \delta\Pr[u\in\mathcal{G}_1\setminus\mathcal{G}_0]. \end{aligned}\tag{6.28}$$

Using (6.28) and (6.27) we obtain

$$\begin{aligned}\Pr_{\substack{\pi,\rho \\ x:=\langle P^{(1)}(\pi),D_1(\rho)\rangle_{\text{trans}} \\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi),D_1(\rho)\rangle_{P^{(1)}}}}[\Gamma_V(\widetilde{D}_2(x,\rho)) = 1] &\geq \frac{\Pr_{u\leftarrow\mu_\delta^k}[g(u)=1] + \varepsilon - \Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_0] - (1-\frac{1}{6k})\varepsilon}{\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]} \\ &\overset{(6.28)}{\geq} \frac{\varepsilon + \delta\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0] - (1-\frac{1}{6k})\varepsilon}{\Pr_{u\leftarrow\mu_\delta^k}[u\in\mathcal{G}_1\setminus\mathcal{G}_0]} \geq \delta + \frac{\varepsilon}{6k}.\end{aligned}\tag{6.29}$$

Clearly, the running time of $\widetilde{\text{Gen}}$ is bounded by some polynomial $p(k, \frac{1}{\varepsilon}, n)$ with oracle calls. Furthermore, the algorithm $\widetilde{\text{Gen}}$ outputs a circuit $\widetilde{D}$ such that it satisfies with probability at least $1 - p(k, \frac{1}{\varepsilon}, n) \cdot 2^n$ the statement of Lemma 6.3. This concludes the proof of Lemma 6.3. □

6.2 Discussion

6.2.1 Optimality of the result

TODO: When does p.r. fails to a.h.
TODO: Why fixing all puzzles is not possible
TODO: Why we believe that it is not possible to do any better
TODO: Mention the number of the verification queries
TODO: Try to explain somehow the number of hint queries

TOOD: What we tried to prove the lower bound
TOOD: Why it was not possible

- no easy way to handle inefficient solver without visible effect on the number of hint queries
- complicated breaker + puzzles has to depend on each other,
- has to ask many hint and verification queries in each phase on the more less the same domain
- it interacts with the solver that is computationally bounded, hence limitation on the number of hint and verification queries
- the solver may be dishonest answer the queries with different (not true response)

Appendix A

Appendix

A.1 Concentration Bounds

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

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

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

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

where $0 \leq \delta \leq 1$.

A.2 The Proof of Lemma 6.3 Under the Simplifying Assumptions

We give a proof of Lemma 6.3 in the case where non of the estimate is large. For the sake of simplicity we make the following assumptions

$$\Pr_{\substack{\pi,\rho \\ x:=\langle P^{(1)}(\pi),D_1(\rho)\rangle_{trans}}}[D_2(x,\rho) \neq \perp] = 1 \tag{A.4}$$

$$\forall \pi \in \{0,1\}^n : S_{\pi,b} \leq (1 - \frac{1}{k}). \tag{A.5}$$

Loosely speaking, in (A.4) we assume that for every π circuit D outputs an answer, and in (A.5) we make the assumption that all surpluses are low. In the complete proof of Lemma 6.3 this assumptions fail only slightly such that it is still possible to obtain the desired result. However, the calculations are fairly lengthy. The following simplified calculations are intended to give the Reader better intuition about the full proof. We have

$$\begin{aligned}
\Pr_{\substack{\pi,\rho \\ x:=\langle P^{(1)}(\pi),D_1(\rho)\rangle_{trans} \\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi),D_1(\rho)\rangle_{P^{(1)}}}}[\Gamma_V(D_2(x,\rho)) = 1] &\overset{(A.4)}{=} \Pr_{\substack{\pi,\rho \\ x:=\langle P^{(1)}(\pi),D_1(\rho)\rangle_{trans} \\ (\Gamma_V,\Gamma_H):=\langle P^{(1)}(\pi),D_1(\rho)\rangle_{P^{(1)}}}}[\Gamma_V(D_2(x^*,\rho)) = 1 \mid D_2(x,\rho) \neq \perp] \\
&\overset{(*)}{=} \Pr_{\pi^{(k)}}[c_1 = 1 \mid c \in \mathcal{G}_1 \setminus \mathcal{G}_0] \\
&= \frac{\Pr_{\pi^{(k)}}[c_1 = 1 \wedge c \in \mathcal{G}_1 \setminus \mathcal{G}_0]}{\Pr_{\pi^{(k)}}[c \in \mathcal{G}_1 \setminus \mathcal{G}_0]} \\
&\overset{(6.15)}{=} \mathop{\mathbb{E}}_{\pi^*}\left[\frac{\Pr_{\pi^{(k-1)}}[c_1^* = 1 \wedge c \in \mathcal{G}_1 \setminus \mathcal{G}_0]\big(\Pr_{\pi^{(k-1)}}[c \in \mathcal{G}_0 \setminus \mathcal{G}_1] - (S_{\pi^*,1} - S_{\pi^*,0})\big)}{\Pr_{\pi^{(k-1)}}[c \in \mathcal{G}_0 \setminus \mathcal{G}_1]\Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_1 \setminus \mathcal{G}_0]}\right] \\
&\geq \mathop{\mathbb{E}}_{\pi^*}\left[\frac{\Pr_{\pi^{(k-1)}}[c_1^* = 1 \wedge c \in \mathcal{G}_1 \setminus \mathcal{G}_0]\big(\Pr_{\pi^{(k-1)}}[c \in \mathcal{G}_0 \setminus \mathcal{G}_1] - (1 - \frac{1}{k})\varepsilon\big)}{\Pr_{\pi^{(k-1)}}[c \in \mathcal{G}_0 \setminus \mathcal{G}_1]\Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_1 \setminus \mathcal{G}_0]}\right] \\
&\geq \frac{\Pr_{\pi^{(k)}}[c_1 = 1 \wedge c \in \mathcal{G}_1 \setminus \mathcal{G}_0] - (1 - \frac{1}{k})\varepsilon}{\Pr_{u \leftarrow \mu_\delta^k}[u \in \mathcal{G}_1 \setminus \mathcal{G}_0]} \\
&\overset{(6.28)}{\geq} \frac{\delta \Pr[u \in \mathcal{G}_1 \setminus \mathcal{G}_0] + \varepsilon - (1 - \frac{1}{k})\varepsilon}{\Pr[u \in \mathcal{G}_1 \setminus \mathcal{G}_0]} \\
&\geq \delta + \frac{\varepsilon}{k},
\end{aligned}$$

where in $(*)$ we use the facts that $D_2(x,\rho) \neq \perp$ if and only if D_2 finds $c \in \mathcal{G}_1 \setminus \mathcal{G}_0$ and conditioned on $D_2(x,\rho) \neq \perp$ we have that $\Gamma_V(D_2(x,r)) = 1$ if and only if $c_1 = 1$.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

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

[CHS05] Ran Canetti, Shai Halevi, and Michael Steiner. Hardness amplification of weakly verifiable puzzles. In *Theory of Cryptography*, pages 17–33. Springer, 2005.

[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 *Theory of cryptography*, pages 128–145. Springer, 2009.

[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.

[HS11] Thomas Holenstein and Grant Schoenebeck. General hardness amplification of predicates and puzzles. In *Theory of Cryptography*, pages 19–36. Springer, 2011.

[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.

[Yao82] Andrew C Yao. Theory and application of trapdoor functions. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 80–91. IEEE, 1982.