On the Formal Verification of Polynomial Commitment Schemes: the KZG and beyond

Tobias Rothmann *[1,2]

[1]Technical University Munich (TUM)
[2]Arcium

January 14, 2025

Abstract

We formalize the notion of polynomial commitment schemes (PCSs) in the interactive theorem prover Isabelle/HOL and formally verify two KZG versions' security proofs. We formalize an abstract definition of polynomial commitment schemes and abstract games for correctness, binding, hiding, and knowledge soundness/extractability, unifying the notion of security for PCSs. We formally verify the security proofs for two PCSs; the standard DL-version KZG and a batched version of the KZG. Our proofs follow the sequence-of-games approach proposed by Shoup, with machine-checked transitions, as is the standard in the CryptHOL framework for formal verification of cryptography in Isabelle. To our knowledge, this is the first formalization of polynomial commitment schemes and the first formal verification of any concrete polynomial commitment scheme's security proofs. This is a work in progress; while the bulk of the work is already done (formalization of PCSs and according games, formal verification of the KZG and the batched KZG), there are still some remaining tasks and ongoing revisions we would like to implement; most notably we want to formalize the AGM properly.

1 Introduction

In the past 5 years general purpose zero-knowledge proofs (ZKPs) and succinct non-interactive arguments of knowledge (SNARKs) have gained significant adoption in the blockchain space. ZK-Rollups are implementing and using these new cryptographic primitives to secure billions of funds [Coia, Coib]. Polynomial Commitment Schemes (PCSs) play a vital role in constructing SNARKs, providing the cryptographic part to the complexity theory used to construct SNARKs. Though not all SNARKs are constructed in this way, many of them are [MBKM19, CHM+19, GWC19, BDFG20, KST21, AST23, ACFY24]. Furthermore, PCSs have recently been applied in various other cryptographic protocols, like Witness Key Encapsulation Mechanisms (WKEMs) [FHAS24], data availability sampling [HASW23], and threshold signatures [DCX+23].

This work is focused on formalizing the notion of Polynomial Commitment Schemes in the interactive theorem prover Isabelle/HOL. A PCS is a two-party cryptographic protocol between a committer and a verifier. It allows the committer to commit to an arbitrary polynomial in a way that is both binding (i.e. the commitment for a polynomial is unambiguous) and hiding from the verifier (i.e. the verifier cannot guess the polynomial from the commitment). Furthermore, a PCS allows to reveal points of the polynomial, such that only the points are revealed, but not the polynomial itself. Although PCS are frequently used to construct cryptographic protocols, such as SNARKs, we are not aware of a formalization of polynomial commitment schemes. With this work, we set out to change this and give the first formalization of an abstract polynomial commitment schemes in an interactive theorem prover, specifically focusing on the security proofs, thus, paving the way for formalized security of SNARKs and ZKPs.

We instantiate two versions of the PCS proposed by Kate Zaverucha Goldberg (KZG) [KZG10], the standard DL-version and a batched version.

*tobias.rothmann@tum.de

Related Work

Most PCS constructions fall into one of two groups, namely discrete-log-based (DL-based) or hash-based/code-based. We instantiate our formalization with DL-based PCSs and leave hash-based PCS to future work.

Hash-based PCS constructions, like FRI[BBHR18], Binius[DP23], Basefold[ZCF23], and WHIR[ACFY24], are plausibly post-quantum secure and used in transparent SNARKs such as scalable transparent non-interactive arguments of knowledge (STARKs) [BSBHR18]. Note, a SNARK is considered transparent if no trusted setup is needed (i.e. a STARK) [BSBHR18]. Furthermore, note, that the setup required by the KZG is trusted as it generates the secret key, which is a trapdoor information for the scheme. Further DL-based constructions, like e.g. Dory [Lee20], Hyrax[WTas$^+$17], and Bulletproofs[BBB$^+$18] improve time/space-complexities on some operations for a trade-off in others, compared to the KZG, or achive a transparent setup in turn for larger time/space complexity. Note, that there are many more constructions for PCSs, e.g. DARK[BFS20] and SLAP[AFLN23]. We point the reader to [BMM$^+$21] for an incomplete overview of polynomial commitment schemes.

The most notable advantages of the KZG, we are aware of, are: Firstly, the KZG is explicitly mentioned in many SNARKs [GWC19, MBKM19, CHM$^+$19, BDFG20, KST21] and, secondly, the KZG is the most efficient discrete-log based PCS w.r.t group operations [BMM$^+$21]. Furthermore, the KZG is the first construction of a polynomial commitment scheme and widley used in practice [BFL$^+$22, ZKS, Scr]. On the other hand, no clear contender for the most efficient hash-based PCS has emerged yet – the research is still ongoing [DP23, ZCF23, ACFY24].

We formalize our proofs in CryptHOL[Loc17], which is a framework to formalize game-based proofs in Isabelle/HOL. Besides CryptHOL there are many frameworks and tools specifically for formal verification of cryptography, to mention a few: 'A Framework for Game-Based Security Proofs' [Now07] or CertiCrypt [BGZB09] in Coq[BSTZ], EasyCrypt[Str], CryptoVerify[Cad] and cryptoline [cry]. Furthermore, the interactive theorem prover Lean[dMKA$^+$15] has recently been used for formal verification of cryptographic protocols due to its extensive math library *mathlib* [BM23]. These tools and Isabelle, specifically CryptHOL, have been used to formally verify many cryptographic primitives and protocols, of which we will only highlight the ones that are closely related to our formalization.

Firsov and Unruh formalize security properties for zero-knowledge protocols from sigma-protocols in EasyCrypt [FU22]. Bailey and Miller formalize specific SNARK constructions that do not rely on commitment schemes or other similarly complex cryptographic primitives in Lean [BM23].

Basin et al. formalize the Random Orcale Model (ROM) in CryptHOL [BLS17].

Butler et al. use CryptHOL to formalize a general framework for commitment schemes from Σ-protocols in [BLAG19], formally verifying among others the homomorphic Pedersen Commitment Scheme [BLAG19]. Though their framework provides a good orientation for our framework, it does not sufficiently capture the complexity of polynomial commitment schemes.

Bosshard, Bootle and Sprenger use CryptHOL to formally verify the sum-check protocol [BBS24], which is another cryptographic primitive that is used in SNARKs with fast provers, such as Lasso and Jolt [STW23, AST23]. Their formalization, similarly to our formalization, forms a step towards formally verified complex SNARK proving systems.

With our work, we want to lay the foundation to formalize IOP based SNARKs that rely on PCSs. This could mean formalizing the BCS transform [BSCS16] or formally verifying prominent KZG-based SNARKS, examples that explicitly mention the KZG include Plonk [GWC19], Halo[BDFG20], Marlin [CHM$^+$19], and Nova [KST21].

Contributions

We formalize an abstract polynomial commitment scheme and corresponding security games in the interactive theorem prover Isabelle/HOL. Furthermore, we instantiate two PCS constructions, the standard DL-based KZG and a batched KZG version.

We define five abstract games against an adversary, covering correctness, the standard security properties for polynomial commitment schemes and desirable properties for SNARK construction:

- **correctness:** Honest verifier and honest committer interaction. Formally, a Probability Mass-funtion (PMF) over the probability of acceptance by the verifier.

- **polynomial binding:** The adversary outputs one commitment to two distinct polynomials ϕ and ϕ' with according witnesses. The result is a PMF capturing the event that the verifier accepts both polynomials for the commitment.
- **evaluation binding:** The adversary outputs two distinct evaluation pairs $(\phi(i), i)$ and $(\phi(i)', i)$ to a commitment C (i and C being chosen and fixed by the adversary) with according witnesses. The result is a PMF capturing the event that the verifier accepts the two distinct evaluations for the polynomial commitment.
- **hiding:** For an arbitrary but fixed polynomial ϕ of degree d, given d evaluations of ϕ, the adversary outputs a polynomial. The result is a PMF over the equality of the adversary's output polynomial and ϕ.
- **knowledge soundness:** Sometimes referred to as extractability [Tha22, CY24]. Intuitively this property states, that an adversary must know a polynomial ϕ and compute the commitment C for ϕ honestly, to reveal points. This game takes an extractor algorithm E as input. The adversary outputs a commitment. The extractor, given access to the commitment then outputs an polynomial p and a trapdoor value td for the commitment. Then the adversary outputs the evaluation pair $(i, \phi(i))$ and according witness. The result is a PMF capturing the event that the verifier accepts the adversary's outputs, but the adversary's evaluation does not match the extractors evaluation (i.e. Eval applied to i and the extractor's polynomial p and trapdoor td).

  This property obviously contradicts the hiding property in the standard model, that is why this property is analyzed either in the Algebraic Group Model (AGM) or Random Orcale Model (ROM). These intuitively capture essential steps any adversary would have to make, forming the basis for the analysis.

We instantiate two concrete PCS constructions:

- **KZG:** The non-hiding DL-version of the KZG described in [KZG10]. We show correctness, polynomial binding, evaluation binding, knowledge soundness, and a weaker version of hiding that is only hiding for uniform random polynomials.
- **batched KZG** The batched version is an extension of the KZG for two more functions, which allow to evaluate a degree d polynomial at up to d points with one witness and one pairing check. We show correctness, polynomial binding, evaluation binding, knowledge soundness and discuss the claim made in [KZG10] that this extension is weakly hiding in the same sense as the DL-version.

The original paper proofs are outlined in a rough descriptive proof style, which is known to be error-prone [BR04]. To enhance the security expression of the proofs we transform them into sequence-of-games proofs, which are particularly rigorous [Sho04, BR04], following Shoup [Sho04]. We use CryptHOL[Loc17], a framework specifically for sequences-of-games proofs in Isabelle, for our formalization. Additionally, we extend the Isabelle theories $SPMF$ and $CryptHOL$ with some lemmas and definitions helpful for our formalization.

A preliminary version of the formalization and proofs can be found on github[1]. The final version will follow shortly, once finished and cleaned-up.

2 Preliminaries

In this section, we introduce the notation used throughout the paper and capture the most important preliminaries in definitions. We start with the mathematical notation and concepts used in this paper.

2.1 Mathematical Preliminaries

We let p and q denote prime numbers if not explicitly stated otherwise. Groups are written in a multiplicate manner with the $\cdot$ operator and the abbreviation ab for $a \cdot b$. We let g and h denote group elements if not explicitly stated otherwise. Furthermore, we use the notation $\mathbb{F}_p$ for a finite field

[1]https://github.com/tobias-rothmann/Polynomial-Commitment-Schemes

of prime order p (note that the integers modulo p are isomorph to any finite field of prime order p [Lan02]) with the conventional operators '+' and '·' for addition and multiplication. We let a,b, and c denote finite field elements if not explicitly stated otherwise.

Definition 2.1 (cyclic group)**.** Let $\mathbb{G}$ be a group of prime order p. We call a group cyclic iff: $\exists g \in \mathbb{G}.\ \forall e \in \mathbb{G}.\ \exists n \in \mathbb{N}.\ e = g^n$, which is equivalent to $\mathbb{G} = \{1, g, g^2, ..., g^{p-1}\}$ [Lan02]. If such a g exists, we call it a generator.

From now on we write $\mathbf{g}$ for a randomly chosen but fixed generator of a respective cyclic group.

Definition 2.2 (pairings)**.** Let $\mathbb{G}$ and $\mathbb{H}$ be two groups of prime order p. A pairing is a function: $\mathbb{G} \times \mathbb{G} \to \mathbb{H}$, with the following two properties:

- **Bilinearity:** $\forall g, h \in \mathbb{G}.\ \forall a, b \in \mathbb{F}_p.\ e(g^a, h^b) = e(g, h)^{ab}$
- **Non-degeneracy:** $\neg(\forall g, h \in \mathbb{G}.\ e(g, h) = 1)$

[KZG10]

From now on let e denote a pairing function if not explicitly stated otherwise.

Now that we have introduced the mathematical preliminaries we will tend to the cryptographic preliminaries.

2.2 Cryptographic Preliminaries

In this section, we will introduce the security notions that we use in this paper and the concepts behind them.

We start with the definition of a negligible and a poly-bounded function from which we will define our adversarial model, against which we will prove security in this paper.

Definition 2.3. Let $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}$ be a function. We call f negligible iff:

$$\forall c \in \mathbb{R}_{>0}.\ \exists n_0.\ \forall n \geq n_0.\ |f(n)| < 1/n^c$$

[BS23]

Boneh and Shoup state "Intuitively, a negligible function $f : \mathbb{F}_{\geq 0} \to \mathbb{R}$ is one that not only tends to zero as $n \to \infty$, but does so faster than the inverse of any polynomial." [BS23]

From now on let ϵ denote a negligible function if not explicitly stated otherwise.

Definition 2.4. Let $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}$ be a function. We call f poly-bounded iff:

$$\exists c, d \in \mathbb{R}_{>0}.\ \forall n \in \mathbb{N}_0.\ |f(n)| \leq n^c + d$$

[BS23]

Note, we will define (probabilistic) algorithms for some security parameter κ and bound their performance (i.e. running time) using the notion of negligibility and poly-boundedness with respect to κ.

We capture the security of our cryptographic system in games against an (efficient or bounded-query) adversary. Typically, the adversary has to break a security property in those games (e.g. decrypt a cyphertext). However, before we formally define games, we define what an adversary is.

Definition 2.5 (Efficient Adversary)**.** An adversary is a probabilistic algorithm, that takes a security parameter κ as its first argument and returns some probabilistic result. We call an adversary efficient if its running time is poly-bounded in κ except for negligible probability (with respect to κ) [BS23].

Additionally, we will make use of a different computational model, namley the Algebraic Group Model (AGM) [FKL17].

Definition 2.6 (algebraic algorithm)**.** Let $\mathbb{F}_p$ be a finite field of prime order p and $\mathbb{G}$ a cyclic group of prime order p. An algebraic algorithm is a (probabilistic) algorithm whose input includes at least one group element, the generator of $\mathbb{G}$. Furthermore, an algebraic algorithm is characterized by outputting linear combinations of the group elements it received with the according linear combination vector i.e. given $g_0, g_1, \ldots, g_n \in \mathbb{G}$, for every group output $z \in \mathbb{G}$, an algebraic algorithm must ouput $\vec{z} = (z_0, \ldots, z_n)$, such that $z = \prod_{i=0}^{n} g_i^{z_i}$ [FKL17].

Definition 2.7 (Algebraic Group Model (AGM))**.** The Algebraic Group Model is a computational model in which all adversaries are modelled as algebraic algorithms [FKL17]. The efficiency definition for adversaries in the AGM is analogue to 2.5

Now that we have defined the necessary adversary classes and computational model for our analysis, we define games. Whenever we don't specify the class of the adversary (i.e. we write adversary instead of AGM or efficient adversary) the adversary could be of any class.

Definition 2.8 (Games)**.** Games are probabilistic algorithms with access to an adversary [BS23]. They output a boolean value that represents either that the game has been won by the adversary or that it has been lost [BS23]. Formally we write games as a sequence of functions and adversary calls [BS23].

Notation wise, we write ‘ $\xleftarrow{\$}$ ‘ followed by a set for uniform sampling from that set, ‘ $\leftarrow$ ‘ followed by a probability mass function (e.g. an adversary result) to sample from that function space, and ‘ $=$ ‘ for an assignment of a deterministic value. Moreover, we write ‘ : ‘ followed by a condition to assure that the condition has to hold at this point. To give an example, think of the following game as "sampling a uniformly random a from $\mathbb{F}_p$, get the probabilistic result from $\mathcal{A}$ as b, computing c as F applied to a and b, and assert that P holds for c":

$$\begin{pmatrix} a \xleftarrow{\$} \mathbb{F}_p, \\ b \leftarrow \mathcal{A}, \\ c = F(a,b) \\ \ : \ P(c) \end{pmatrix}$$

Note, that the notation is applicable for any probabilistic algorithms, not just games. However, if not explicitly stated otherwise we will let this notation denote games.

Next, we define proofs in the sequence-of-games framework proposed by shoup [Sho04], the method which we will use to formally prove security.

Definition 2.9 (Sequences-of-Games (SoG) Proofs)**.** Sequences-of-games style proofs are a sequence of game-hops that bound the probability of one game to another [BR04, Sho04].

The two types of game hops we will use in our proofs are:

- **game hop as a bridging step:**

  A bridging step alters the function definitions, such that the game probability does not change [Sho04].

- **game hop based on a failure event:**

  In a game hop based on a failure event, two games are equal except if a specific failure event occurs [Sho04]. The failure event should have a negligible probability for the game-based proof to hold.

Typically we will define a game for a certain security definition applied to our cryptographic protocol and reduce that game using game hops to a hardness assumption game, thus showing that breaking the security definition for our cryptographic protocol is at least as hard as breaking the hardness assumption [BS23]. Hence we need to define hardness and accordingly hardness assumptions:

Definition 2.10 (hardness)**.** Given a computational problem P, we say P is hard if and only if no adversary exists, that solves P with non-negligible probability [MvOV96].

Definition 2.11 (hardness assumptions)**.** Hardness assumptions are computational problems that are generally believed to be hard [BS23, MvOV96].

Within cryptography, there exist several hardness assumptions, we will cover the ones used in this paper and formally define according games.

From now on let ‘$\xleftarrow{\$}$‘ denote uniform sampling from the respective set.

Definition 2.12 (discrete logarithm (DL))**.** Let $\mathbb{G}$ be a cyclic group with generator $\mathbf{g}$. Let $a \xleftarrow{\$} \mathbb{F}_p$. Then for every efficient adversary $\mathcal{A}$: $\Pr[a = \mathcal{A}(\mathbf{g}^a)] = \epsilon$ holds [KZG10].

Formally we define the DL game:

$$\left(\begin{array}{c} a \xleftarrow{\$} \mathbb{F}_p \\ a' \leftarrow \mathcal{A}(\mathbf{g}^a) \\ : a = a' \end{array}\right)$$

Definition 2.13 (t-Strong Diffie-Hellmann (t-SDH))**.** Let $\mathbb{G}$ be a cyclic group with generator $\mathbf{g}$. Let $t \in \mathbb{N}$ be fixed. Let $a \xleftarrow{\$} \mathbb{F}_p$. For every adversary $\mathcal{A}$:

$$\Pr\left[(c, \mathbf{g}^{\frac{1}{a+c}}) = \mathcal{A}([\mathbf{g}, \mathbf{g}^a, \mathbf{g}^{a^2}, \ldots, \mathbf{g}^{a^{t-1}}])\right] = \epsilon$$

holds for all $c \in \mathbb{F}_p \backslash \{a\}$ [KZG10].

Formally we define the t-SDH game:

$$\left(\begin{array}{c} a \xleftarrow{\$} \mathbb{F}_p \\ (c, g') \leftarrow \mathcal{A}([\mathbf{g}, \mathbf{g}^a, \mathbf{g}^{a^2}, \ldots, \mathbf{g}^{a^{t-1}}]) \\ : \mathbf{g}^{\frac{1}{a+c}} = g' \end{array}\right)$$

The following definition is analogous to the previous one, except that the result is passed through a pairing function. Intuitively, this makes this assumption stronger as two groups, and thus group values, and the pairing function can now be used by an adversary.

Definition 2.14 (t-Bilinear Strong Diffie-Hellmann (t-BSDH))**.** Let $\mathbb{G}$ and $\mathbb{H}$ be cyclic groups with generators $\mathbf{g}$ and $\mathbf{h}$. Let $t \in \mathbb{N}$ be fixed and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{H}$ be a pairing function. Let $a \xleftarrow{\$} \mathbb{F}_p$. For every adversary $\mathcal{A}$:

$$\Pr\left[(c, e(\mathbf{g}, \mathbf{g})^{\frac{1}{a+c}}) = \mathcal{A}([\mathbf{g}, \mathbf{g}^a, \mathbf{g}^{a^2}, \ldots, \mathbf{g}^{a^{t-1}}])\right] = \epsilon$$

holds for all $c \in \mathbb{F}_p \backslash \{a\}$ [KZG10].

Formally we define the t-BSDH game:

$$\left(\begin{array}{c} a \xleftarrow{\$} \mathbb{F}_p \\ (c, g') \leftarrow \mathcal{A}([\mathbf{g}, \mathbf{g}^a, \mathbf{g}^{a^2}, \ldots, \mathbf{g}^{a^{t-1}}]) \\ : e(\mathbf{g}, \mathbf{g})^{\frac{1}{a+c}} = g' \end{array}\right)$$

Now that we have introduced the necessary preliminaries, notions, and definitions for proving security for cryptographic protocols, we tend to the type of protocols we formalize in this work.

Definition 2.15 (Commitment Schemes (CS))**.** A Commitment Scheme is a cryptographic protocol between two parties, we call the committer and the verifier, and consists of four functions:

- **KeyGen** takes a security parameter κ and generates a key ck for the committer and a key vk for the verifier.
- **Commit** takes the committer key ck, a message m and computes a commitment C for m and according trapdoor information td to open the commitment.
- **Verify** takes the verifier key vk, a commitment C, an trapdoor information td, a message m and decides whether C is a valid commitment to m using vk and td.

[Tha22]

The protocol assumes that KeyGen was used to distribute the keys ck and vk accordingly to the committer and verifier, such that no party can learn the keys of the other party if they are to remain secret.

Once the keys are correctly distributed, the protocol follows three steps:

1. The committer uses their keys ck to commit to a arbitrary message m, invoking Commit(ck, m) from which they obtain a commitment C to m and an trapdoor information td for the commitment. The committer stores td and sends C to the verifier.

2. At a later point in time, the committer might decide to open the commitment C they sent to the verifier. To open the commitment the committer sends the trapdoor information td and the message m to the verifier.

3. The verifier invokes Verify on the values it received from the committer (C, m and td) to decide whether m is the message the commitment C was computed for.

Once the keys are set up, the protocol may run arbitrary times and in parallel (i.e. the committer can commit to arbitrary many messages and reveal them independently).

In our formalization, we work with a specific type of commitment scheme, namely Polynomial Commitment Schemes (PCS):

Definition 2.16 (Polynomial Commitment Scheme (PCS))**.** A Polynomial Commitment Scheme is a Commitment Scheme as defined in 2.15, with its message space restricted to polynomials (i.e. messages are polynomials).

Furthermore, a PCS supports two more functions to allow point-wise (i.e. partially) opening a commitment to a polynomial:

- **Eval** takes the committer key ck, the trapdoor information td to the commitment C, the commited polynomial ϕ, a value i and computes the evaluation of ϕ at i, $\phi(i)$, and a witness w for the point $(i, \phi(i))$.

- **VerifyEval:** takes the verifier keys vk, a commitment C, a point $(i, \phi(i))$, a witness w and checks that the point is consistent with the commitment (i.e. the point is part of the polynomial that C is a commitment to) using w and vk.

We call the standard commitment schemes function *Verify* in the PCS *VerifyPoly* to avoid ambiguity with *VerifyEval*.

Note that we omit the verifier keys in the following four definitions for readability, but generally assume the adversaries to have access to the verifier keys. The same holds for the security parameter κ that is given to *KeyGen* and the adversary. These parameters are implicit.

We formally define four security properties for a PCS:

Definition 2.17 (Polynomial Binding)**.** We say a PCS is polynomial binding if and only if the probability of any adversary finding a commitment value C, trapdoor information td and td' and distinct polynomials ϕ and ϕ', such that the verifier accepts both polynomials is negligible:

$$\Pr\big[\text{VerifyPoly}(C, td, \phi) \wedge \text{VerifyPoly}(C, td', \phi')\big] = \epsilon$$

[KZG10]

Definition 2.18 (Evaluation Binding)**.** We say a PCS is evaluation binding if and only if the probability of any adversary finding a commitment value C, two witnesses w and w' and two distinct evaluations $\phi(i)$ and $\phi(i)'$ for any i, such that the verifier accepts both evaluations is negligible:

$$\Pr\big[\text{VerifyEval}(C, (i, \phi(i)), w) \wedge \text{VerifyEval}(C, (i, \phi(i)'), w')\big] = \epsilon$$

[KZG10]

Definition 2.19 (Knowledge Soundness)**.** We say a PCS is knowledge sound if and only if there exists an extractor algorithm E such that for an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ the probability of winning the following game is negligible:

$$\left(\begin{aligned} (ck, vk) &\leftarrow \text{KeyGen} \\ (C, st) &\leftarrow \mathcal{A}_1(ck) \\ \phi' &\leftarrow E(C) \\ (i, \phi(i), \omega) &\leftarrow \mathcal{A}_2(st) \\ &: \phi(i) \neq \phi'(i) \wedge \\ &\quad \text{VerifyEval}(vk, C, (i, \phi(i)), w) \end{aligned}\right)$$

[GWC19]

Definition 2.20 (hiding)**.** We say a PCS is hiding if and only if the probability of any adversary finding an unknown point of the polynomial ϕ from the commitment $C := \text{Commit}(ck, \phi)$ and $\deg(\phi)$ points with witness $(i, \phi(i), \text{Eval}(ck, \phi, i))$ is negligible. [KZG10]

Note that finding an unknown point of ϕ, given $deg(\phi)$ points is analog to finding the polynomial ϕ itself, since any finite degree polynomial ϕ is uniquely defined by $deg(\phi) + 1$ points [Lan02].

2.3 Isabelle/HOL

Now that we have covered the theoretical preliminaries, we introduce Isabelle/HOL[PN], the interactive theorem prover we use to formalize PCSs and formally verify our proofs in. Note that we will use the abbreviation *Isabelle* to refer to *Isabelle/HOL*.

Isabelle is an interactive theorem prover for higher-order logic (HOL) that supports a minimal functional programming language [Wen23], as well as a large set of formalizations of mathematical areas such as algebra and analysis. The most essential formalizations are shipped with Isabelle in the *HOL-library*, besides that there exists a large database of formalizations, the *Archive of Formal Proofs (AFP)*[Huc]. We will use the finite field definition as well as the factorization algorithm for polynomials over finite fields from the Berlekamp-Zassenhaus[DJTY16] AFP entry. Furthermore, we use the Lagrange Polynomial Interpolation algorithm from the AFP entry *Polynomial Interpolation*[TY16] and the *Elimination of Repeated Factors* algorithm from the AFP entry *Elimination_Of_Repeated_Factors*[KE23]. Apart from that, we use another AFP entry that is central to our proofs, the CryptHOL framework[Loc17].

CryptHOL

CryptHOL is a framework for game-based proofs in Isabelle [BLS17]. It captures games in a subprobability mass function (spmf) monad. That is essentially a *Option* type wrapped in a probability mass function (pmf). The unassigned probability mass, represented as the pmf of *None* is propagated through the monad. Elementary events from spmf's can be bound to variables through the monadic notation. The notation for games in CryptHOL is drawn from Haskell's do-notation.

Example 2.1.

```
do {
    x ← sample_uniform S;
    return_spmf x
}
```

The *do* $\{\ldots\}$ block is syntactic sugar, intuitively, it captures the monadic block (i.e. a game). *sample_uniform* is the spmf, that wraps the uniformly over S distributed pmf. We use $\leftarrow$ to bind an elementary event of the pmf to x, intuitively this means x is not a concrete element of S, but represents any element of S with respect to the probability distribution of the pmf (which in the example is uniform). Note, that if S is empty, the binding operation fails and the result is the unassigned probability mass. We use *return_spmf* to return the pmf, that x represents, wrapped in a spmf, which is *sample_uniform* of S in this example. Hence, the result in this example is, in fact, equivalent to the spmf '*sample_uniform* S'.

More complex games can be created by composing more functions, including functions that fail, i.e. return the unassigned probability mass, on purpose to perform checks. One specific example of such a function is the *assert_spmf* function, which is commonly used to check that messages from the adversary are well-formed. We illustrate the correct usage in example 2.2.

CryptHOL furthermore provides syntactic sugar to handle failure (i.e. an unassigned probability mass), namely the *TRY ELSE* block. *TRY A ELSE B*, captures semantically 'try to return A if A is a failure, return B'. We will commonly use this pattern to abstract whether an adversary has won a game. Specifically, we define A, such that the result is a wrapped spmf of *True* if and only if the adversary has won and otherwise let the game fail, i.e. result in the unassigned probability mass. We define then B to be the wrapped spmf of *False*, so the game *TRY A ELSE B* captures cleanly whether the adversary has won the game.

Example 2.2.

```
    TRY do {
        x::nat ← sample_uniform S;
        x'::nat ← 𝒜 (g^x);
        _::unit ← assert_spmf(x=x');
        return_spmf True
    } ELSE (return_spmf False)
```

This example game captures the discrete log problem.

As in the first game, an elementary event of S, which is a natural number, is bound to x. The adversary is applied to the group element $\mathbf{g}^x$ and returns an spmf over the natural numbers, which we bind to x'. Next, we use *assert_spmf* to check whether x and x' are equal i.e. the adversary guessed the right number. We bind the result only symbolically to *unit*, such that if the check does not hold, the failure can be propagated as the unassigned probability mass. If the assert check holds, the result is the wrapped spmf of *True*, otherwise a failure has been propagated and the *ELSE* branch is triggered. If the *ELSE* branch is triggered the result is the wrapped spmf of *False*.

3 Polynomial Commitment Schemes Formalized

In this section we formalize the notion of polynomial commitment schemes in Isabelle. There are multiple versions of PCS definitions in the literature, some omit to verify the complete polynomial [XZS22], some do not explicitly consstruct the eval and verify eval functions [BDFG20, BMM+21] but allow for an interactive evaluation protocol, and most differ slightly from each other in terminology of the functions [KZG10, BMM+21, XZS22, BDFG20, KT23].

For simplicity reasons, we follow [KZG10], but rename the "createWitness" function to the more common "Eval" [BDFG20, XZS22, BMM+21]. Additionally, we leave open to support the interactive eval protocol approach at a later point in time when interactive (oracle) proofs are formalized in Isabelle.

For now, we formalize the version outlined in 2.16 following [KZG10]. We formalize PCSs in Isabelle as a locale that can be instantiated by concrete implementations of 6 functions: *key_gen, commit, verify_poly, eval* and *verify_eval* for the PCS and a additional function called *valid_msg* that asserts that the adversary's messages are well-formed:

```
locale abstract_polynomial_commitment_scheme =
  fixes key_gen :: "'secuirty('ck × 'vk) spmf"
        - outputs the keys received by the two parties
  and commit :: "'ck ⇒ 'r poly ⇒ ('commit × 'trapdoor) spmf"
        - outputs the commitment as well as the trapdoor, which might be used
        to derive witnesses or open the commit
  and verify_poly :: "'vk ⇒ 'r poly ⇒ 'commit ⇒ 'trapdoor ⇒ bool"
        - checks whether the polynomial corresponds to the commitment
  and eval :: "'ck ⇒ 'trapdoor ⇒ 'r poly ⇒ 'argument
  ⇒ ('evaluation × 'witness)"
        - outputs a the evaluation to a given argument, a witness
        to prove validity of the evaluation value
  and verify_eval :: "'vk ⇒ 'commit ⇒ 'argument ⇒ ('evaluation × 'witness)
  ⇒ bool"
        - checks whether the point is on the polynomial corresponding to the
        commitment
  and valid_msg :: "'r poly ⇒ bool" - checks whether a message is valid
begin
```

A notable design decision one might find unintuitive at first, is that we do not take the same type that the polynomial is over (*'r*) for the value we pass to *eval* (i.e. we pass for a polynomial over e.g. a finite filed $\mathbb{F}$ not a value from $\mathbb{F}$), but instead pass an unrealted type *'argument*. The reasoning behind that decision is to include batched versions of PCSs, in those cases using *'r* would restrict instantiation to evaluate only one e.g. field value at a time, while *'argument* allows for more flexibility as it can be anything from a single finite field element to a k-tuple or list of values.

Commitment Scheme sublocale

Some properties for PCSs are equivalent to properties of standard commitment schemes. Concretely CS correctness and polynomial binding correspond directly to CS properties correctness and binding. Since Butler et al. [BLAG19] formalized commitment schemes already in Isabelle, we built upon their formalization and instantiate a sublocale for standard commitment schemes with the functions *key_gen, commit, verify_poly*, and *valid_msg*:

```
sublocale cs: abstract_commitment key_gen commit verify_poly valid_msg .
```

This allows us to use their definitions of Correctness and Binding directly in our PCS locale:

```
definition correct_cs_game :: "'r poly ⇒ bool spmf"
  where "correct_cs_game ≡ cs.correct_game"

definition correct_cs
  where "correct_cs ≡ cs.correct"
```

3.1 Correctness

In tbhis subsection we define the correctness property concretely. We differentiate between two correctness notions:

- CS correctness

  This is the correctness property of normal commitment schemes, i.e. the probability of winning the following game must be one for an arbitrary polynomial ϕ:

$$\begin{pmatrix} (ck, vk) \leftarrow \text{KeyGen} \\ (C, td) \leftarrow \text{Commit}(ck, \phi) \\ : \text{VerifyPoly}(vk, C, \phi) \end{pmatrix}$$

  Formally in Isabelle, we refer to Butler et al. [BLAG19] and they define CS correctness as follows:

```
definition correct where "correct ≡
  (∀m. valid_msg m ⟹ spmf (correct_game m) True = 1)"
```

- Eval correctness

  This is the correctness property capturing the polynomial evaluation, i.e. the probability of winning the following game must be one for an arbitrary polynomial ϕ and value i:

$$\begin{pmatrix} (ck, vk) \leftarrow \text{KeyGen} \\ (C, td) \leftarrow \text{Commit}(ck, \phi) \\ (\phi(i), w) \leftarrow \text{Eval}(ck, td, \phi, i) \\ : \text{VerifyEval}(vk, C, i, \phi(i), w) \end{pmatrix}$$

  Formally in Isabelle we define Eval correctness as follows:

```
definition correct_eval where "correct_eval ≡
  (∀p i. valid_msg p ⟹ spmf (correct_eval_game p i) True = 1)"
```

Now we tend to the security properties. We define abstract security games and give adversary advantages for each game. These definitions can be used by instantiations to prove security properties for their concrete PCS construction. We define only the games and the adversary advantges, not concrete properties, to not restrict instantiations into one computational model or analysis model (e.g. perfect, statistical, computational hiding, analysis in the standard model, AGM, ROM). Hence in each instantiation the predefined advantages must be bound. This could look something like this (example theorem evaluation binding from the KZG formal verification):

```
theorem evaluation_binding: "eval_bind_advantage 𝒜 ≤
  t_SDH_G_p.advantage (eval_bind_reduction 𝒜)"
```

3.2 Polynomial Binding

We define the polynomial binding game following 2.17:

$$\left(\begin{array}{r l} PK \leftarrow & \text{KeyGen}, \\ (C, \phi, td, \phi', td') \leftarrow & \mathcal{A}(ck), \\ b = & \text{VerifyPoly}(vk, C, \phi, td), \\ b' = & \text{VerifyPoly}(vk, C, \phi', td'), \\ : & \phi \neq \phi' \wedge b \wedge b' \end{array}\right)$$

This corresponds to the standard CS binding game. Hence we use the existing definitions formalized by Butler et al. [BLAG19] for the CS binding game and according adversary advantage:

```
definition poly_bind_game
  where "poly_bind_game ≡ cs.bind_game"

definition poly_bind_advantage
  where "poly_bind_advantage ≡ cs.bind_advantage"
```

3.3 Evaluation Binding

The evaluation biding game is consistent throughout the literature [KZG10, ZCF23, DP23, BDFG20, BMM+21] and follows the CS binding game approach, using *VerifyEval* instead of *VerifyPoly*. We define the concrete game as follows:

$$\left(\begin{array}{r l} PK \leftarrow & \text{KeyGen}, \\ (C, i, \phi(x), \omega, \phi(x)', \omega') \leftarrow & \mathcal{A}(ck), \\ b = & \text{VerifyEval}(vk, C, i, \phi(x), \omega), \\ b' = & \text{VerifyEval}(vk, C, i, \phi(x)', \omega'), \\ : & \phi \neq \phi' \wedge b \wedge b' \end{array}\right)$$

This game expressed in Isabelle remains virtually the same:

```
definition eval_bind_game ::
  "('ck, 'commit, 'argument, 'evaluation, 'witness) eval_bind_adversary
  ⇒ bool spmf"
where "eval_bind_game 𝒜 = TRY do {
  (ck, vk) ← key_gen;
  (c, i, v, w, v', w') ← 𝒜 ck;
  let b = verify_eval vk c i (v,w);
  let b' = verify_eval vk c i (v',w');
  return_spmf (b ∧ b' ∧ v ≠ v')} ELSE return_spmf False"
```

3.4 Hiding

There are two notions of hiding used in the literature:

- **IND-CPA CS Hiding** This form of hiding is to our knowledge explicitly used in PCSs that have an interactive evaluation protocol [BDFG20, KT23]. It is equivalent to the IND-CPA hiding property of standard CSs. Since we do not explicitly support interactive evaluation protocols yet, we do not consider this property for now. Note however, since it is equivalent to CS IND-CPA hiding, which has already been formalized by Butler et al. [BLAG19], this proptery could be formalized easliy following the polynomial binding approach 3.2.

- **Eval-Hiding** This property was introduced in [KZG10], we put their definition in a formal game for an arbitrary polynomial ϕ and list of arguments I, which reveals $deg(\phi)$ distinct points of ϕ:

$$\begin{pmatrix} (ck, vk) \leftarrow \text{KeyGen}, \\ (C, td) = \text{Commit}(ck, \phi), \\ W = \text{Eval}(ck, td, \phi)`I \\ \phi' \leftarrow \mathcal{A}(vk, C, W), \\ : \ \phi = \phi' \end{pmatrix}$$

Note that the following equivalence holds, since a degree d polynomial is defined by $d+1$ distinct points:

$$\begin{pmatrix} (ck, vk) \leftarrow \text{KeyGen}, \\ (C, td) = \text{Commit}(ck, \phi), \\ W = \text{Eval}(ck, td, \phi)`I \\ \phi' \leftarrow \mathcal{A}(vk, C, W), \\ : \ \phi = \phi' \end{pmatrix} \equiv \begin{pmatrix} (ck, vk) \leftarrow \text{KeyGen}, \\ (C, td) = \text{Commit}(ck, \phi), \\ W = \text{Eval}(ck, td, \phi)`I \\ (i, y) \leftarrow \mathcal{A}(vk, C, W), \\ : \ \phi(i) = y \end{pmatrix}$$

Assessing hiding via this game is in some sense a possibly weaker and a stronger assumption than IND-CPA CS hiding. While it does consider the information provided by the witnesses from *Eval*, in that point being stronger than the IND-CPA hiding, it is not an indistinguishability game, in that point being a bit weaker. The difficulty in desgining a IND-CPA game for PCSs is that the information revealed by *Eval* is only partially hiding - the evaluation is revealed, while the rest of the polynomial is to remain secret. Hence to create an IND-CPA game, the adversary has to be restricted in the choice of polynomials; all points revealed via *Eval* must be equal for both distinct polynomials chosen by the adversary. Such a PCS IND-CPA hiding game could look as follows:

$$\begin{pmatrix} (ck, vk) \leftarrow \text{KeyGen}, \\ (\phi, \phi', I) \leftarrow \mathcal{A}_1 \\ b \xleftarrow{\$} \{0, 1\} \\ p = \text{if } b \text{ then } \phi \text{ else } \phi' \\ (C, td) = \text{Commit}(ck, p), \\ W = \text{Eval}(ck, td, p)`I \\ b' \leftarrow \mathcal{A}(vk, C, W), \\ : \ b = b' \wedge \phi \neq \phi' \wedge \forall i \in I \ .\phi(i) = \phi'(i) \end{pmatrix}$$

While this can be an proposal for future work, it is not evidence-based or battle-tested and generally not used by researchers creating PCSs and hence researchers formally verifying these PCSs. That is why we do not formalize this version of hiding, but the hiding notion proposed by [KZG10]:

```
definition eval_hiding_game :: "'r poly ⇒ 'argument list
   ⇒ ('r, 'vk, 'commit, 'argument, 'evaluation, 'witness)
      eval_hiding_adversary ⇒ bool spmf"
where "eval_hiding_game p I 𝒜 = TRY do {
   (ck, vk) ← key_gen;
   (c,d) ← commit ck p;
   let W = map (λi. eval ck d p i) I;
   p' ← 𝒜 vk c I W;
   return_spmf (p = p')} ELSE return_spmf False"
```

3.5 Knowledge Soundness

The definition of knowledge soundess is consistent throughout the literature [GWC19, ZCF23, KT23, XZS22], altough sometimes referred to as *extractabillity* [Tha22, CY24]. We define the following game for knowledge soundness against an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with an extractor algrithm $\mathcal{E}$:

$$\begin{pmatrix} (ck, vk) \leftarrow \text{KeyGen}, \\ (C, \sigma) \leftarrow \mathcal{A}_1(ck), \\ (\phi, td) \leftarrow \mathcal{E}(C), \\ (i, \phi(i)', w) \leftarrow \mathcal{A}_2(\sigma) \\ : \ \text{VerifyEval}(vk, C, i, (\phi(i)', w)) \wedge \phi(i) \neq \phi(i)' \end{pmatrix}$$

We translate the game in Isabelle and adjust it to our definition of PCSs:

```
definition knowledge_soundness_game :: "('ck, 'commit, 'state, 'argument,
      'evaluation, 'witness)
      knowledge_soundness_adversary ⇒ ('r, 'commit, 'trapdoor) extractor
      ⇒ bool spmf"
where "knowledge_soundness_game 𝒜 E = TRY do {
            let (𝒜1,𝒜2) = 𝒜;
            (ck,vk) ← key_gen;
            (c,σ) ← 𝒜1 ck;
            (p,d) ← E c;
            (i, w) ← 𝒜2 σ;
            let (p_i,_) = w;
            let (p_i',_) = eval ck d p i;
            return_spmf (verify_eval vk c i w ∧ p_i ≠ p_i')
            } ELSE return_spmf False"
```

The Isabelle version differes in the evaluation of ϕ from the abstract knowledge soundness game defined before. Since we defined the PCS definition explicitly to cover batch versions through the *'argument* and *'evaluation* types, we cannot simply evaluate the polynomial ϕ, which would give a value of type *'r* and compare it to a *'evaluation* value. Instead, we assert that the result of *Eval* applied to ϕ, the polynomial extracted by the extractor, has to be distinct from the *'evaluation* value given by the adversary $\mathcal{A}_2$ to be verified in *VerifyEval*. This is semantically equivalent to $\phi(i) \neq \phi(i)'$ if *'evaluation* is *'r*. In the case of an batched version this represents the event that at least one evaluation provided by the adversary differs from the extracted polynomial.

4 Instantiations

In this section we instantiate the two concrete PCS constructions, the standard KZG and the batched KZG.

4.1 Formalized Hardness Assumptions

Before we start with the instantiation and security proofs of concrete PCSs, we formalize the hardness assumptions according to 2.11. We define them as games, such that we can target them in sequences-of-games style proofs against an adversary.

Discrete Logarithm

We formalize the DL game according to 2.12 as follows:

```
TRY do {
    a  ← sample_uniform p;
    a' ← 𝒜 (g^a);
    return_spmf (a = a')
} ELSE return_spmf False
```

We sample a $\in \mathbb{Z}_p$ uniformly random using *sample_uniform* and supply the adversary $\mathcal{A}$ with $\mathbf{g}^a$. The adversary returns a guess for a, namely a'. The adversary wins the game if and only if a equals a'.

t-Strong Diffie-Hellmann

We formalize the t-SDH game according to 2.13 as follows:

```
TRY do {
    $\alpha$ ← sample_uniform p;
    let instc = map $(\lambda i.\mathbf{g}\hat{}(\alpha\hat{}i))$ [0..<t+1];
    (c,g) ← $\mathcal{A}$ instc;
    return_spmf (g = $\mathbf{g}^{\frac{1}{\alpha+c}}$)
} ELSE return_spmf False
```

We sample $\alpha \in \mathbb{Z}_p$ uniformly random using *sample_uniform* and compute the t-SDH instance $(\mathbf{g}, \mathbf{g}^{\alpha}, \mathbf{g}^{\alpha^2}, \ldots, \mathbf{g}^{\alpha^t})$ as instc. The adversary $\mathcal{A}$ receives the t-SDH instance and returns a field element c $\in \mathbb{Z}_p$ and a group element g $\in \mathbb{G}_p$. The adversary wins if and only if the group element g is equal to $\mathbf{g}^{\frac{1}{\alpha+c}}$.

Note, that the t-SDH assumption was formalized in the practical course and thus is not part of this thesis. Nevertheless, we outlined it for completeness.

t-Strong Bilinear Diffie-Hellmann

We formalize the t-BSDH game according to 2.14 as follows:

```
TRY do {
    $\alpha$ ← sample_uniform p;
    let instc = map $(\lambda i.\mathbf{g}\hat{}(\alpha\hat{}i))$ [0..<t+1];
    (c,g) ← $\mathcal{A}$ instc;
    return_spmf (g = (e g g)^$(1/(\alpha+c))$)
} ELSE return_spmf False
```

The game is equivalent to the t-SDH game except for the equality check and the adversary's return types. While the t-SDH adversary returns a group element from $\mathbb{G}_p$, the t-BSDH adversary returns a group element of the parings e target group $\mathbb{G}_T$. Accordingly, the equality check is performed on the target group, specifically on the t-SDH value passed through the pairing e: e $\mathbf{g}$ $\mathbf{g}^{\frac{1}{\alpha+c}} = (\text{e } \mathbf{g} \text{ } \mathbf{g})^{\frac{1}{\alpha+c}}$.

Lastly, note that every game in this chapter can trivially be transformed to do the equality check in an assert statement and return True after that, without changing the according game's spmf. This will be useful for the security proofs in the next chapter. Proofs for each game are to be found in the respective Isabelle theories.

4.2 KZG

In this section, we give the concrete instantiation, as well as an intuition of the KZG.

Intuition

In this section, we want to give an intuitive idea of how the KZG constructs a PCS. Specifically, how a commitment is constructed and how the createWitness function works. We neglect the exact setup process for the intuition but note that we obtain the ability to evaluate a polynomial on a fixed unknown random field element α.

The commitment C is the evaluation at the unknown random field element, intuitively this is sound because of the Schwarzt-Zippel lemma, which states that the probability of two different polynomials evaluating to the same value at a random point is negligible [Tha22].

To reveal a point $(i, \phi(i))$, we need to create a witness. For that, consider the following: for any point i, any (non-trivial) polynomial $\phi(x)$ can be expanded to $\phi(x) = \psi(x) * (x - i) + \phi(i)$, for $\psi(x) = \frac{\phi(x)-\phi(i)}{(x-i)}$. Note that this equation is equivalent to $\phi(\alpha) = \psi(\alpha) * (\alpha - i) + \phi(i)$ (i.e. the equation evaluated at the unknown random point), except for negligible probability, due to the Schwartz-Zippel lemma [Tha22]. We use $\psi(\alpha)$ as the witness ω, additionally note that $\phi(\alpha)$ is the commitment C. This choice of the witness is sound, as the equation to be checked by the verifier, $C = \omega * (\alpha - i) + \phi(i) \iff \phi(\alpha) = \psi(\alpha) * (\alpha - i) + \phi(i)$, holds if and only if the claimed $\phi(i)$ is exactly the evaluation of $\phi(x)$ at point i, as $\phi(x) = \psi(x) * (x - i) + \phi(i) \iff \phi(x) - \phi(i) = \psi(x) * (x - i)$ and the left-hand-side must be zero for $x = i$.

In the real protocol, the values for the unknown random point, the commitment, and the witness are provided in a group (e.g. α is given as $\mathbf{g}^{\alpha}$) to hide them and a pairing is used to check the equation for the witness.

Definition

The KZG is a polynomial commitment scheme as defined in 2.16. In this section, we outline the PCS-constructing functions based on the original paper [KZG10]:

- **KeyGen**$(\kappa, t) \rightarrow \mathbb{G}_p, \mathbb{G}_T, e, PK$

  takes a security parameter κ and generates instances for the algebraic primitives the KZG needs, which are:

  - two cyclic groups, $\mathbb{G}_p$ and $\mathbb{G}_T$, of prime order p, where $p \geq 2^{2^{\kappa}}$.
  - a pairing function $e : \mathbb{G}_p \times \mathbb{G}_p \rightarrow \mathbb{G}_T$, as in 2.2, such that the t-SDH assumption holds.

  [KZG10] Additionally, KeyGen generates a toxic-waste secret key α, from which it generates the public key $PK = (\mathbf{g}, \mathbf{g}^{\alpha}, \mathbf{g}^{\alpha^2}, \ldots, \mathbf{g}^{\alpha^t}) \in \mathbb{G}_p^{t+1}$, for $t < 2^{\kappa}$. KeyGen outputs the algebraic primitives and PK to both the prover and verifier (as pk and ck) [KZG10]. For good practice, the toxic waste secret key α should be deleted right after PK was generated, as α contains trap-door information.

- **Commit(PK,**$\phi(x)$**)** $\rightarrow C$

  takes the public key PK and a polynomial $\phi(x) \in \mathbb{Z}_p[X]$ of maximum degree t, such that $\phi(x) = \sum_0^{deg(\phi)} \phi_j x^j$ [KZG10]. Commit returns the commitment $C = \mathbf{g}^{\phi(\alpha)}$ for ϕ as $C = \prod_0^{deg(\phi)} (\mathbf{g}^j)^{\phi_j}$ [KZG10]. The trapdoor information for the KZG is simply the polynomial $\phi(x)$, which is why we omit to return it as well.

- **VerifyPoly**$(PK, C, \phi(x)) \rightarrow bool$

  takes the public key PK, a commitment C and a polynomial $\phi(x) \in \mathbb{Z}_p[X]$ of maximum degree t, such that $\phi(x) = \sum_0^{deg(\phi)} \phi_j x^j$ [KZG10]. *VerifyPoly* returns 1 if $C = \mathbf{g}^{\phi(\alpha)}$, otherwise 0 [KZG10].

- **CreateWitness**$(PK, \phi(x), i) \rightarrow i, \phi(i), \omega_i$

  takes the public key PK, a polynomial $\phi(x) \in \mathbb{Z}_p[X]$ of maximum degree t, such that $\phi(x) = \sum_0^{deg(\phi)} \phi_j x^j$, and a value $i \in \mathbb{Z}_p$ [KZG10]. CreateWitness computes $\psi_i(x) = \sum_0^{deg(\psi)} \psi_j x^j$ as $\psi_i(x) = \frac{\phi(x) - \phi(i)}{(x-i)}$ and returns the tuple $(i, \phi(i), \mathbf{g}^{\psi(\alpha)})$, where $\mathbf{g}^{\psi(\alpha)}$ is computed, similar to the commit, as $\mathbf{g}^{\psi(\alpha)} = \prod_0^{deg(\psi)} (\mathbf{g}^j)^{\psi_j}$ [KZG10].

- **VerifyEval**$(PK, C, i.\phi(i), \omega_i) \rightarrow bool$

  takes the public key PK, a commitment C, a claimed point $(i, \phi(i))$ and a witness ω_i for that point [KZG10]. VerifyEval checks the equation $\phi(\alpha) = \psi(\alpha)(\alpha - i) + \phi(i)$ using the pairing e as: $e(C, g) = e(\omega_i, \mathbf{g}^{\alpha}/\mathbf{g}^i) e(\mathbf{g}, \mathbf{g})^{\phi(i)}$ and returns the result [KZG10].

Formalization and Instatiation

In the formalization, we extract the algebraic primitives (i.e. the groups and the pairing) into a locale that can be instantiated. The major advantage of this is its compatibility with formalized instances of these algebraic primitives. Hence, the KZG's security could be instantiated for concrete constructions of pairings, such as the Barreto-Nährig[BN06] or Baretto-Lynn-Scott[BLS03] pairing friendly Elliptic Curves (ECs), should they be formalized.

We define *KeyGen* as a wrapper around a function called *Setup*, which mirrors the Setup function from [KZG10] without generating the algebraic primitives, essentially *Setup* is exposing the secret key α as well as the public key PK and *KeyGen* exposes only PK. This change is mentioned here, because we will often decompose *KeyGen* into *Setup*, to gain access to the secret key α in the proofs.

The remaining functions are defined trivially according to 4.2, we omit the details as they are not part of this thesis and are irrelevant to the proofs.

We use the functions to instantiate the KZG in our formalized PCS framework (see 3) as follows:

```
sublocale PCS: abstract_polynomial_commitment_scheme KeyGen Commit VerifyPoly
   CreateWitness VerifyEval valid_msg .
```

Note, *CreateWitness* is instantiated as the *eval* function.

4.3 Batched KZG

In this section, we otline the batched version of the KZG, as defined in [KZG10], and intsantiate it in our framework. The batched version allows to verifiably open a commitment to a polynomial for up to t points using only one witness [KZG10]. Note, that this batch version is different from the one mentioned in the Sonic[MBKM19] and Plonk[GWC19] SNARK, where multiple commitments can be batch opened for one point. The [KZG10] batched version is an extension of the KZG as defined in chapter 4.2 for the following two functions [KZG10]:

1. **CreateWitnessBatch**$(PK, \phi(x), B) \rightarrow B, r(x), \omega_B$

   takes the public key PK, a polynomial $\phi(x) \in \mathbb{Z}_p[X]$ of maximum degree t, such that $\phi(x) = \sum_0^{deg(\phi)} \phi_j x^j$, and a set $B \subset \mathbb{Z}_p$ of maximum t values [KZG10]. *CreateWitnessBatch* computes $\psi_B(x) = \sum_0^{deg(\psi)} \psi_j x^j$ and $r(x) = \sum_0^{deg(r)} r_j x^j$ as $\psi_B(x) = \frac{\phi(x)-r(x)}{\prod_{i\in B}(x-i)}$ and $r(x) = \phi(x) \bmod \prod_{i\in B}(x-i)$ and returns the tuple $(B, r(x), \mathbf{g}^{\psi(\alpha)})$, where $\mathbf{g}^{\psi(\alpha)}$ is computed as $\mathbf{g}^{\psi(\alpha)} = \prod_0^{deg(\psi)}(\mathbf{g}^j)^{\psi_j}$, similarly to the commitment [KZG10]. Furthermore, note $\phi(x) = \psi_B(x) * (\prod_{i\in B}(x-i)) + r(x)$ and thus $\forall i \in B.\ r(i) = \phi(i)$.

2. **VerifyEvalBatch**$(PK, C, B, r(x), \omega_B) \rightarrow bool$

   takes the public key PK, a commitment C, a set of positions B, a polynomial $r(x)$ which evaluates to the claimed evaluations of $\phi(x)$ at the positions in B, and a witness ω_B for the claimed points [KZG10]. *VerifyEvalBatch* checks the equation $\phi(x) = \psi_B(x) * (\prod_{i\in B}(x-i)) + r(x)$ using the pairing e as: $e(C, \mathbf{g}) = e(\mathbf{g}^{\prod_{i\in B}(\alpha-i)}, \omega_B)e(\mathbf{g}, \mathbf{g}^{r(\alpha)})$ and returns the result [KZG10].

Intuitively, *CreateWitnessBatch* uses the same technique as *CreateWitness*, but divides ϕ for multiple points, i.e. $\prod_{i\in B}(x-i)$ instead of $(x-i)$. Therefore, *CreateWitnessBatch* respectively returns a polynomial with all evaluations for the points in the set B instead of a value for a single point i, i.e. $r(x)$ for B, where $\forall i \in B.\ r(i) = \phi(i)$, instead of $\phi(i)$ for i.

Similarly, *VerifyEvalBatch* checks an equation that matches the equation checked by *CreateWitness*, except for the evaluation point $\phi(i)$, which is replaced by the evaluation polynomial $r(x)$, and the division for multiple positions $\prod_{i\in B}(x-i)$ instead of one $(x-i)$, i.e. $\phi(x) = \psi_B(x)*(\prod_{i\in B}(x-i))+r(x)$ instead of $\phi(x) = \psi_i(x) * (x-i) + \phi(i)$.

Formalization and Instatiation

We formalize the batch version as a locale extension of the KZG definition locale *KZG_Def*:

```
locale KZG_BatchOpening_def = KZG_Def
```

We formalize *CreateWitnessBatch* as follows:

```
definition CreateWitnessBatch :: "'a pk  ⇒ 'e polynomial ⇒ 'e eval_position set
    ⇒ ('e eval_position set × 'e polynomial × 'a eval_witness)"
where
"CreateWitnessBatch PK ϕ B =(
  let r = r B ϕ;
      ψ = ψ_B B ψ;
      w_i = g_pow_PK_Prod PK ψ
  in
  (B, r, w_i)
)"
```

where r is the function that computes the remainder of ϕ divided by $\prod_{i\in B}(x-i)$, for a polynomial ϕ and a set of positions B, formally we define r in Isabelle as follows:

```
fun r :: "'e eval_position set ⇒ 'e polynomial ⇒ 'e polynomial" where
    "r B ϕ = do {
    let prod_B = prod (λi. [:-i,1:]) B;
    ϕ mod prod_B}"
```

, ψ_B is the function that computes $\frac{\phi(x)-r(x)}{\prod_{i\in B}(x-i)}$, formally we define ψ_B in Isabelle as follows:

```
fun ψB :: "'e eval_position set ⇒ 'e polynomial ⇒ 'e polynomial" where
"ψB B ϕ = do {
    let prod_B = prod (λi. [:-i,1:]) B;
    (ϕ - (r B ϕ)) div prod_B}"
```

, and *g_pow_PK_Prod* computes the $\mathbf{g}^{\phi(\alpha)}$ from the public key $PK = (\mathbf{g}, \mathbf{g}^{\alpha}, \mathbf{g}^{\alpha^2}, \ldots, \mathbf{g}^{\alpha^t})$ and a polynomial ϕ.

We formalize *VerifyEvalBatch* as follows:

```
definition VerifyEvalBatch :: "'a pk ⇒ 'a commit ⇒ 'e eval_position set
    ⇒ 'e polynomial ⇒ 'a eval_witness ⇒ bool"
where
    "VerifyEvalBatch PK C B r_x w_B = (
    let g_pow_prod_B = g_pow_PK_Prod PK (prod (λi. [:-i,1:]) B);
        g_pow_r = g_pow_PK_Prod PK r_x in
    (e g_pow_prod_B w_B) ⊕ (e g g_pow_r) = e C g))"
```

where e is a pairing function as defined in chapter 4.2 (KZG Def), $\oplus$ is the group operation in $\mathbb{G}_T$ and *g_pow_PK_Prod* is the group generator $\mathbf{g}$, exponentiated with the evaluation of a polnomial on α. Hence, *g_pow_prod_B* and *g_pow_r* are the polynomials $\prod_{i\in B}(x-i)$ and $r(x)$ evaluated at α and the equation outlined in the definition of VerifyEvalBatch can be checked on the, to the committer and verifier unknown, secret key α.

We instantiate the batched KZG using the *KeyGen, Commit* and *VerifyPoly* functions from the standard DL-version KZG, only changing the *eval* and *verify_eval* i.e. *CreateWitness* and *VerifyEval* functions to *CreateWitnessBatch* and *VerifyEvalBatch*:

```
sublocale PCS: abstract_polynomial_commitment_scheme KeyGen Commit VerifyPoly
   CreateWitnessBatch VerifyEvalBatch valid_msg .
```

5 Proofs

5.1 KZG Proofs

In this section, we outline the security properties as well as the idea behind the formalization of each security property. Specifically, we summarise the reductionist proof from the paper, show how we transformed it into a sequence-of-games proof and outline the latter. We will not give concrete proofs in Isabelle syntax as this would go beyond the scope.

Each section covers one security property. Firstly, we describe the paper proof, followed by the sequence-of-games proof. Lastly, we outline the formalization, specifically the security game, the reduction algorithm and if needed additional formalization details.

5.1.1 Polynomial Binding

In this subsection we outline the polynomial binding proof given in [KZG10], transform it into a sequences-of-games format and outline our formal Verification of that proof. The following game captures the abstract *polynomial binding* game, which we're using to proof polynomial binding, instantiated with the KZG functions, played against an efficient adversary $\mathcal{A}$:

$$\left(\begin{array}{rl} PK \leftarrow & \text{KeyGen}, \\ (C,\phi,\phi') \leftarrow & \mathcal{A}(PK), \\ b = & \text{VerifyPoly}(PK,C,\phi), \\ b' = & \text{VerifyPoly}(PK,C,\phi'), \\ : & \phi \neq \phi' \wedge b \wedge b' \end{array}\right)$$

Original Paper Proof

The proof in the original paper [KZG10] is a reduction to t-SDH assumption. Given the two polynomials ϕ and ϕ' that have the same commitment C, all provided by the adversary, we know by unfolding the definition of *Commit* that $\phi(\alpha) = \phi'(\alpha)$, where α is the trapdoor information from *KeyGen* and the t-SDH instance. The reduction computes $\phi'' := \phi - \phi'$, which has the root α, since $\phi(\alpha)'' = \phi(\alpha) - \phi'(\alpha) = 0$. The reduction computes α then by factoring ϕ'', which is polynomial time [KZG10].

Sequence-of-games Proof

A look at the *polynomial binding* and the *t-SDH* games reveals that *KeyGen*'s generation of PK is equivalent to generating a t-SDH instance. Furthermore, the major difference between the two games are in the asserts. While the t-SDH game asserts that the t-SDH assumption is broken, the polynomial binding game asserts that both polynomials verify via *VerifyPoly*. Since we know from the orginal papaer proof that *VerifyPoly* for both polynomials fro the same commitment already implies that α can be extracted, what we can do is:

1. simply add the computation of α to the assertion of the polynomial binding game.
2. erase the *VerifyPoly* conditions, making th game weaker
3. wrap the polynomial binding adversary in a function that computes α from the adversary outputs and returns the tuple $(0, \mathbf{g}^{\frac{1}{\alpha}})$, which is breaking the t-SDh assumption

The wrapped function is the reduction algorithm and the remainign game is equivalent to the t-SDh game, which concledes the sequence-of-games proofs. The formal proof of this outline is given in the file *KZG_poly_bind.thy*.

Formalization

The goal of this proof is to show the following theorem, which states that the probability of any adversary breaking polynomial binding is less than or equal to winning the t-SDh (using the reduction adversary):

```
theorem polynomial_binding: "poly_bind_advantage 𝒜
    ≤ t_SDH.advantage (bind_reduction 𝒜)"
```

The instantiated polynomial binding game in Isabelle looks as follows:

```
TRY do {
    PK ← KeyGen;
    (C, ϕ, ϕ') ← 𝒜 PK;
    _::unit ← assert_spmf(ϕ_i≠ ϕ_i' ∧ valid_msg ϕ
        ∧ valid_msg ϕ);
    let b = VerifyPoly PK C ϕ;
    let b' = VerifyPoly PK C ϕ';
    return_spmf (b ∧ b')
} ELSE return_spmf False
```

The game captures the spmf over True and False, which represent the events whether the adversary has broken polynomial binding . The public key PK is generated using the formalized *KeyGen* function of the KZG. The adversary $\mathcal{A}$, given PK, outputs values to break polynomial binding, namely a commitment value C and two polynomials, ϕ and ϕ'. We use *assert_spmf* to assert that the two polynomials are distinct and of degree less than the output of *KeyGen* via *valid_msg*. The game is counted as lost for the adversary in case the asserts do not hold. We assign to *b* and *b'* the result of the formalized *VerifyPoly* algorithm of the KZG. Polynomial binding is broken if and only if *b* and *b'* hold i.e. two distinct polynomials verify for the same commitment.

We formally define the reduction adversary as follows:

```
    fun bind_reduction
    :: "('a pk, 'e polynomial, 'a commit, 'e polynomial)  bind_adversary ⇒ ('a,'e) t_SDH.adversary
  where
    "bind_reduction 𝒜 PK = do {
    (C, ϕ, _, ϕ', _) ← 𝒜 PK;
    let α = find_α (PK!1) (ϕ - ϕ');
    return_spmf (0::'e mod_ring, g^(1/α))}"
```

That is a higher-order function, that takes the polynomial binding adversary $\mathcal{A}$ and returns an adversary for the t-SDH game. That is, the function that calls the adversary $\mathcal{A}$ on some public key PK and returns $(0, \mathbf{g}^{\frac{1}{\alpha}})$. The function *find_*$\alpha$ is a subroutine that factors a given polynomial and checks for a root r whether $\mathbf{g}^r$ equals a given group point (in this case PK!0 which corresponds to $\mathbf{g}^\alpha$ i.e. it checks whether $r = \alpha$) and if so returns r. Factoring polynomials is not straight forward to formalize and did pose the biggest challenge in the formalization of this security property. While the Berlekamp-Zassenhaus formalization [DJTY16] does support factoring of square-free polynomials, there was no polynomial time algorithm in Isabelle to factor non-square-free polynomials. We want to thank Katharina Heidler (formerly Kreuzer) and Manuel Eberl for formalizing the *Elimnation of Repeated Factors Algorithm*(ERF) [KE23] following our discussions. The ERF allows to extract the square-free part of polynomials over finite fields, preserving the roots of the polynomial. Thus allowing to extract the roots of non-square free polynomials via concatination with the factorization from Berlekamp-Zassenhaus.

5.1.2 Evaluation Binding

In this subsection we outline the *evaluation binding* proof given in [KZG10], transform it into a sequences-of-games format and outline our formal verification of that proof. The following game captures the abstract *evaluation binding* game, which we're using to proof evaluation binding, instantiated with the KZG functions, played against an efficient adversary $\mathcal{A}$:

$$\left(\begin{array}{r} PK \leftarrow \text{KeyGen}, \\ (C, i, \phi_i, \omega_i, \phi'_i, \omega'_i) \leftarrow \mathcal{A}(PK), \\ b = \text{VerifyEval}(PK, C, i, \phi_i, \omega_i), \\ b' = \text{VerifyEval}(PK, C, i, \phi'_i, \omega'_i), \\ : \ \phi_i \neq \phi'_i \wedge b \wedge b' \end{array}\right)$$

Original Paper Proof

The paper [KZG10] proof argues that given an adversary $\mathcal{A}$, that can break the evaluation binding property, an algorithm $\mathcal{B}$ can be constructed, that can break the t-SDH assumption [KZG10]. The concrete construction for $\mathcal{B}$ is: given the t-SDH instance $tsdh\_inst = (\mathbf{g}, \mathbf{g}^\alpha, \mathbf{g}^{\alpha^2}, \ldots, \mathbf{g}^{\alpha^t})$, call $\mathcal{A}$ with $tsdh\_inst$ as PK for $(C, i, \phi(i), \omega_i, \phi(i)', \omega'_i)$ and return:

$$\left(\frac{\omega_i}{\omega'_i}\right)^{\frac{1}{\phi(i)' - \phi(i)}}$$

The reason to why $\mathcal{B}$ is a correct construction is the following:

Breaking evaluation binding means, that $\mathcal{A}$, given a valid public key $PK = (\mathbf{g}, \mathbf{g}^\alpha, \mathbf{g}^{\alpha^2}, \ldots, \mathbf{g}^{\alpha^t})$, can give a commitment C and two witness-tuples, $\langle i, \phi(i), \omega_i \rangle$ and $\langle i, \phi(i)', \omega'_i \rangle$, such that $VerifyEval(PK, C, \langle i, \phi(i), \omega_i \rangle)$

and $VerifyEval(PK, C, \langle i, \phi(i)', \omega_i' \rangle)$ return true [KZG10]. Since *VerifyEval* is a pairing check against $e(C, \mathbf{g})$ we can conclude that:

$$e(\omega_i, \mathbf{g}^{\alpha - i})e(\mathbf{g}, \mathbf{g})^{\phi(i)} = e(C, \mathbf{g}) = e(\omega_i', \mathbf{g}^{\alpha - i})e(\mathbf{g}, \mathbf{g})^{\phi(i)'}$$

which is the pairing term for:

$$\begin{aligned}
&\psi_i(\alpha) \cdot (\alpha - i) + \phi(i) = \psi_i(\alpha)' \cdot (\alpha - i) + \phi(i)' \\
\iff &\psi_i(\alpha) \cdot (\alpha - i) - \psi_i(\alpha)' \cdot (\alpha - i) = \phi(i)' - \phi(i) \\
\iff &(\alpha - i) \cdot (\psi_i(\alpha) - \psi_i(\alpha)') = \phi(i)' - \phi(i) \\
\iff &\frac{\psi_i(\alpha) - \psi_i(\alpha)'}{\phi(i)' - \phi(i)} = \frac{1}{\alpha - i}
\end{aligned}$$

where $\psi_i(\alpha) = log_{\mathbf{g}}\ \omega_i$ and $\psi_i(\alpha)' = log_{\mathbf{g}}\ \omega_i'$ and omitting the $e(C, \mathbf{g})$ [KZG10]. Hence:

$$\left(\frac{\omega_i}{\omega_i'}\right)^{\frac{1}{\phi(i)' - \phi(i)}} = \left(\frac{\mathbf{g}^{\psi_i(\alpha)}}{\mathbf{g}^{\psi_i(\alpha)}}\right)^{\frac{1}{\phi(i)' - \phi(i)}} = \mathbf{g}^{\frac{\psi_i(\alpha) - \psi_i(\alpha)'}{\phi(i)' - \phi(i)}} = \mathbf{g}^{\frac{1}{\alpha - i}}$$

Since $\mathbf{g}^{\frac{1}{\alpha - i}}$ breaks the t-SDH assumption for i, $\mathcal{B}$ is correct [KZG10].

Sequence-of-games Proof

A look at the *evaluation binding* and the *t-SDH* games reveals that *KeyGen*'s generation of PK is equivalent to generating a t-SDH instance. Furthermore, the games differ only in their checks in the respective *assert_spmf* calls (and the adversary's return types). Additionally, we know from the paper proof that the adversary's messages, if correct and wellformed, which is checked in the eval_bind game's asserts, already break the t-SDH assumptions on PK. Hence we give the following idea for the proof:

1. rearrange the eval_bind game to accumulate (i.e. conjuncture) the return-check and all other checks into an assert
2. derive that this conjuncture of statements already implies that the t-SDH is broken and add that fact to the conjuncture.
3. erase every check in the conjuncture by over-estimation, to be only left with the result that the t-SDH is broken.

The resulting game is the t-SDH game with the reduction adversary. See the Isabelle theory *KZG_eval_bind* for the full formal proof.

Formalization

The goal of this proof is to show the following theorem, which states that the probability of any adversary breaking evaluation binding is less than or equal to winning the t-SDH game (using the reduction adversary):

```
theorem evaluation_binding: "eval_bind_advantage 𝒜
    ≤ t_SDH.advantage (bind_reduction 𝒜)"
```

The instantiated evaluation binding game in CryptHOL looks as follows:

```
TRY do {
    PK ← KeyGen;
    (C, i, ϕ_i, w_i, ϕ_i', w_i') ← 𝒜 PK;
    _::unit ← assert_spmf(ϕ_i≠ ϕ_i' ∧ valid_msg ϕ_i w_i
        ∧ valid_msg ϕ_i' w_i');
    let b = VerifyEval PK C i ϕ_i w_i;
    let b' = VerifyEval PK C i ϕ_i' w_i';
    return_spmf (b ∧ b')
} ELSE return_spmf False
```

The game captures the spmf over True and False, which represent the events that the adversary has broken evaluation binding or not. The public key PK is generated using the formalized *KeyGen* function of the KZG. The adversary $\mathcal{A}$, given PK, outputs values to break evaluation binding, namely a commitment value C, two witnesses, w_i and w_i', and two evaluations, ϕ_i and ϕ_i', for a freely chosen point i. Note that we use *assert_spmf* to ensure that the adversary's messages are wellformed and correct. Correct means here ϕ_i and ϕ_i' are indeed two different values. Should the assert not hold, the game is counted as lost for the adversary. We assign to b and b' the result of the formalized *VerifyEval* algorithm of the KZG. Evaluation binding is broken if and only if b and b' hold i.e. both witnesses and evaluations verify at the same point for the same commitment.

We formally define the reduction adversary as follows:

```
fun bind_reduction 𝒜 PK = do {
    (C, i, ϕ_i, w_i, ϕ_i', w_i') ← 𝒜 PK;
    return_spmf (-i, (w_i ÷ w_i')^(1/(ϕ_i' - ϕ_i)));
}
```

That is a higher-order function, that takes the evaluation binding adversary $\mathcal{A}$ and returns an adversary for the t-SDH game. That is, the function that calls the adversary $\mathcal{A}$ on some public key PK and returns $(\frac{\omega_i}{\omega_i'})^{\frac{1}{\phi(i)'-\phi(i)}}$.

5.1.3 Weak Hiding

We formalize the *hiding* property as given in the original paper [KZG10]. Note, that this is a weaker definition than the one formalized in 3.4 and outlined in 2.20, which is why we will call it *weak hiding* from now on.

The *weak hiding* property of the KZG does not unconditionally fulfill the *hiding* property as defined in 2.20 but only holds for uniformly random sampled polynomials [KZG10]. Formally we define *weak hiding* as the following game against an efficient adversary $\mathcal{A}$, where $I \in \mathbb{Z}_p^t$ is an arbitrary distinct list of length t:

$$\left(\begin{array}{r} \phi \xleftarrow{\$} \{\phi.\ \text{degree}(\phi) \le t\}, \\ PK \leftarrow \text{KeyGen}, \\ C = \text{Commit}(PK, \phi), \\ W = (\text{CreateWitness}(PK, \phi))\text{`}I, \\ \phi' \leftarrow \mathcal{A}(PK, C, W), \\ :\ \phi = \phi' \end{array}\right)$$

Original Paper Proof

The paper argues that given an adversary $\mathcal{A}$, that can break the weak hiding property, an algorithm $\mathcal{B}$ can be constructed, that can break the DL assumption [KZG10]. Intuitively, the construction given in the paper proof exploits the fact that the commitment is a group value. Using the trapdoor information, the secret key α, the reduction interpolates a polynomial over t random group points and the DL instance and obtains a commitment value for the interpolated polynomial. Since the adversary can retrieve the polynomial of a commitment, it can hence also retrieve the value of the DL-instance, which is just an evaluation of the polynomial at some point. Note, that the proof contains more boilerplate simulations, like creating a correct public key PK and creating witnesses for the t random group points. We outline the concrete reduction B from the paper, as we understand it, in a probabilistic algorithm, given $\mathbf{g}^a$ as the DL-instance:

$$\left(\begin{aligned}(\alpha, PK) &\leftarrow \text{Setup}, \\ pts &\xleftarrow{\$} \mathbb{Z}_p^{2^t} \\ grp\_pts &= (0, \mathbf{g}^a)\#(\lambda(x,y).\ (x, \mathbf{g}^y))\ `\ pts, \\ C &= \text{interpolate } grp\_pts, \\ W &= (\lambda(x,y).\ (x, y, ((C/\mathbf{g}^y)^{\frac{1}{\alpha - x}})))\ `\ pts, \\ \phi(x) &\leftarrow \mathcal{A}(PK, C, W), \\ &\phi(0)\end{aligned}\right)$$

Firstly, $\mathcal{B}$ generates the public key PK using the function *Setup*, which additionally exposes the secret key α, as opposed to *KeyGen*. Secondly, $\mathcal{B}$ samples t random points *pts*, which are used to create the t witnesses W the adversary requires. Thirdly, $\mathcal{B}$ interpolates the random points in group form (*grp_pts*) together with the DL-instance for the commitment C. Once all inputs for the adversary, namely the public key PK, the commitment C, and the t witness tuples W are generated, the adversary $\mathcal{A}$ is called on them to retrieve the polynomial $\phi(x)$. The result is the polynomial $\phi(x)$ evaluated at zero. Note that the evaluation of $\phi(x)$ at zero is the value a from the DL-instance $\mathbf{g}^a$ because they were paired for the interpolation values. Since returning a from $\mathbf{g}^a$ wins the DL game, this reduction breaks the DL assumption.

Sequence-of-games proof

Firstly, we outline why the reduction algorithm $\mathcal{B}$ cannot trivially be transformed into a reduction adversary for a sequence-of-games style proof.

Note that while $\mathcal{B}$ samples the point list uniform random and thus creates witnesses at uniform random positions, the weak hiding game evaluates the polynomial on a given arbitrary list I and thus creates witnesses at the positions from I. Hence, the witnesses in the weak hiding game are fundamentally different from the ones in the DL-reduction game (uniform random vs arbitrary). Moreover, the probability of the uniformly randomly sampled points matching the arbitrary I is negligible in $\mathbb{Z}_p$, thus, following the game-hops described in 2.9, we see no trivial conversion between the witnesses and thus games.

Our approach to solving this incompatibility is to introduce the arbitrary list I into the reduction algorithm. Instead of uniformly randomly sampling the complete point list, we take an arbitrary I for the positions and let the reduction sample only the evaluations for those positions uniformly random. Thus the reduction creates witnesses at exactly the positions that are in I. Hence, provided with the same I (and that the witness creation is correct) the reduction and the weak hiding game produce the same witness tuples.

However, introducing an arbitrary I for the positions creates a new problem. Note, that to interpolate a polynomial from a list of points, the list must be distinct. Choosing zero for the position on that the polynomial should evaluate to the DL-value was okay in the reduction proof as the probability of t uniformly random points containing zero is negligible in $\mathbb{Z}_p$. However, the probability of zero being contained in an arbitrary list is not negligible (in fact, note, that we cannot express any probability for this event, as we cannot know the probability distribution for the arbitrary list).

We could solve this problem by choosing a random position i, which again would make the probability of i being contained in an arbitrary list of length t negligible. However, we decided to solve this problem algorithmically, by picking an i that is deterministically distinct from I (see the *PickDistinct* algorithm in the *KZG_hiding* Isabelle theory for the concrete algorithm to pick i). This decision eases the proof as we do not have to explicitly expose the probability of i being contained in I, which would be necessary to make a negligibility argument in CryptHOL, but can use the fact that i is distinct from I directly in Isabelle.

We obtain the following new reduction algorithm:

$$\begin{pmatrix} i = \mathit{PickDistinct}(I), \\ (\alpha, PK) \leftarrow \text{Setup}, \\ pts \xleftarrow{\$} \mathbb{Z}_p^t \\ grp\_pts = (i, \mathbf{g}^a) \ \# \ (\lambda(x,y).\ (x, \mathbf{g}^y)) \text{ ` } (\text{zip}(I, pts)), \\ C = \text{interpolate}(grp\_pts), \\ wtnss = (\lambda(x,y).\ (x, y, ((C/\mathbf{g}^y)^{\frac{1}{\alpha - x}}))) \text{ ` } pts, \\ \phi(x) \leftarrow \mathcal{A}(PK, C, wtnss), \\ \phi(i) \end{pmatrix}$$

Now that we have discussed the changes to the reduction algorithm as outlined in 5.1.3, we tend to the proof. We outline the main ideas behind the proof and refer the reader to the Isabelle theory *KZG_hiding* for the full formalized proof.

We start with the weak hiding game and use a bridging step to restate uniformly random sampling of a polynomial as the interpolation of a zipped list of arbitrary I (in the formalized game $i\#I$) and uniformly random sampled evaluations:

$$\left(\phi \xleftarrow{\$} \{\phi.\ \text{degree}(\phi) \le t\}, \quad \right) = \begin{pmatrix} evals \xleftarrow{\$} \mathbb{Z}_p^t, \\ \phi = \text{interpolate}(\text{zip}(I, evals)), \end{pmatrix}$$

Furthermore, note we can expand this split to interpolation over group values:

$$\begin{pmatrix} \phi \xleftarrow{\$} \{\phi.\ \text{degree}(\phi) \le t\}, \\ \phi = \mathbf{g}^{\phi} \end{pmatrix} = \begin{pmatrix} evals \xleftarrow{\$} \mathbb{Z}_p^t, \\ grp\_evals = (\lambda\ i.\ \mathbf{g}^i) \text{ ` } evals, \\ \phi = \text{interpolate}(\text{zip}(I, grp\_evals)), \end{pmatrix}$$

This enables us to split up the DL-instance creation:

$$\begin{pmatrix} evals \xleftarrow{\$} \mathbb{Z}_p^t, \\ grp\_evals = (\lambda\ i.\ \mathbf{g}^i) \text{ ` } evals, \end{pmatrix}$$

$$= \begin{pmatrix} a \xleftarrow{\$} \mathbb{Z}_p, \\ g^a = \mathbf{g}^a, \\ evals \leftarrow \text{sample\_uniform } \mathbb{Z}_p^{t-1}, \\ grp\_evals = g^a \# \ (\lambda\ i.\ \mathbf{g}^i) \text{ ` } evals, \end{pmatrix}$$

The obtained game is equivalent to the DL game for the reduction-adversary except for the computation of the commitment and witnesses and the equality check in the end.

While the weak hiding game checks whether $\phi = \phi'$, where ϕ' is the polynomial outputted by the adversary and ϕ is the randomly sampled polynomial, the DL game for the reduction-adversary only asserts $\phi'(i) = a$, which is equivalent to $\phi'(i) = \phi(i)$ since ϕ is interpolated for (i, a). Moreover, since $\phi' = \phi$ implies $\phi'(i) = \phi(i)$, we can use over-estimation to conclude that the probability of solving the weak hiding game's check is greater or equal to solving the DL-game's check. Semantically we are over-estimating the weak hiding game with the DL game:

$$(\ : \phi' = \phi \quad) \le \quad (\ : \phi'(i) = \phi(i) \quad)$$

The only remaining difference between the game we have obtained now and the DL game is the computation of the commitment and the witness. We describe in the formalization 5.1.3 why the computation of the commitment is equivalent and thus omit the details here. The only difference left in the games is the computation of the witnesses.

The weak hiding game computes the witnesses using the standard KZG function *CreateWitness* for every value in I:

$$\text{CreateWitness}(Pk, \phi, i) = \begin{pmatrix} \psi(x) = \dfrac{\phi(x) - \phi(i)}{x - i}, \\ (i, \phi(i), \mathbf{g}^{\psi(\alpha)}) \end{pmatrix}$$

The reduction adversary emulates the witness computed by *CreateWitness* with the term $(C/\mathbf{g}^{\phi(i)})^{\frac{1}{\alpha - i}}$ and returns the emulated value together with each point $(i\phi(i)) \in \text{zip}(I, pts)$. Note, pts is a list of randomly chosen evaluations (for the values in I). We state the full emulation here:

$$\Big((\lambda(i,\phi(i)).\ (i,\phi(i),((C/\mathbf{g}^{\phi(i)})^{\frac{1}{\alpha-i}}))) \text{ ` } (\text{zip}(I,pts)),\Big)$$

Moreover, note the following equality:

$$(\mathrm{C}/\mathbf{g}^{\phi(i)})^{\frac{1}{\alpha-i}} = (\mathbf{g}^{\phi(\alpha)} \div \mathbf{g}^{\phi(i)})^{\frac{1}{\alpha-i}} = (\mathbf{g}^{\phi(\alpha)-\phi(i)})^{\frac{1}{\alpha-i}} = \mathbf{g}^{\frac{\phi(\alpha)-\phi(i)}{\alpha-i}} = \mathbf{g}^{\psi(\alpha)}$$

However, this equation does not hold for $i = \alpha$ as $\frac{1}{i-\alpha}$ would be a division by zero. While the reduction's function is undefined due to the division by zero in the event that $i = \alpha$, *CreateWitness* is not undefined, as it uses polynomial division before evaluating the polynomial at α. However, the probability of $i = \alpha$ i.e. $i \in pts$ is negligible as the length of pts t is 2^κ and $p \geq 2^{2^\kappa}$,for $\mathbb{Z}_p$. Hence, we can use a game-hop based on a failure event to restate $CreateWitness$ with 'map $(\lambda(i,\phi(i)).\ (i,\phi(i),((C/\mathbf{g}^{\phi(i)})^{\frac{1}{\alpha-i}})))$ pts' and obtain the DL-game for the reduction adversary. Thus proving the theorem stated at the beginning of this subsection.

Formalization

The goal of the weak hiding proof is to show the following theorem, which states that the probability of any adversary breaking weak hiding is less than or equal to winning the DL game (using the reduction adversary):

```
theorem weak_hiding: "weak_hiding_advantage 𝒜 ≤ t_SDH.advantage (reduction 𝒜)"
```

We define the weak hiding game formally in Isabelle as a wrapper around the standard hiding game, that takes an adversary, a list of arguments, simply samples a polynomial uniformly random and passes the sampled polynomial togehter with the adversary and the argument list to the standard hiding game:

```
TRY do {
    ϕ ← sample_uniform_poly t;
    PCS.hiding_game ϕ I 𝒜
} ELSE return_spmf False
```

We unfold this game then into the following game, that resembles clearly the predefined abstract weak hiding game:

```
TRY do {
    ϕ ← sample_uniform_poly t;
    PK ← KeyGen;
    let C = Commit PK ϕ;
    let wtns_tpls = map (λ i. CreateWitness PK ϕ i) I;
    ϕ' ← 𝒜 PK C wtns_tpls;
    return_spmf (ϕ = ϕ')
} ELSE return_spmf False
```

The game captures the spmf over True and False, which represent the events that the adversary has broken hiding or not. Firstly, the polynomial $\phi \in \mathbb{Z}[X]^{\leq t}$ of degree t or less is uniformly sampled. Secondly, the public key PK is generated using *KeyGen*. The commitment C to ϕ is computed using *Commit* and PK. Then, t valid witness tuples *wtns_tpls* are computed, where I $\in \mathbb{Z}_p^t$ is a list of t arbitrary but distinct field elements. The adversary $\mathcal{A}$ receives the public key PK, the commitment C and the t witness tuples *wtns_tpls* and returns a guess for the polynomial ϕ, namely ϕ'. The adversary wins if and only if the guessed polynomial ϕ' equals the chosen polynomial ϕ.

Additionally, we formalize the reduction algorithm. The reduction algorithm $\mathcal{B}$ from the original paper proof in [KZG10] cannot be trivially turned into a reduction adversary that works for a game-based proof (see the prior subsection for details). Hence we need to define a slightly different reduction. We alter the reduction in two ways:

1. we sample random evaluations for a given arbitrary list I, instead of randomly sampling the complete point list. This changes the reduction to be more compatible with the hiding game, which evaluates a random polynomial on a given list I.

2. We pick an i that is guaranteed to be distinct from the list I instead of zero for the position where ϕ evaluates to the DL value a. This eases the proof outlined in the prior subsection as the probability of zero being contained in the arbitrary list I is negligible.

Formally, we define the reduction adversary as follows:

```
fun reduction 𝒜 PK = do {
    let i = PickDistinct I;
    eval_values ← sample_uniform_list t;
    let eval_group = g # map (λi. g^i) eval_values;
    (α, PK) ← Setup;
    let C = interpolate_on (zip (i#I) eval_group) α;
    let wtns_tpls = map (λ(x,y). (x,y, ((C ÷ (g^y))^1/(α − x)))) (zip (i#I)
        eval_values);
    ϕ' ← 𝒜 PK C wtns_tpls;
    return_spmf (poly ϕ' i);
}
```

That is a higher-order function, that takes the hiding adversary $\mathcal{A}$ and an arbitrary field element list I and returns an adversary for the DL game. The algorithm starts by picking a field element i that is distinct from the list I. Then it samples a list of t uniform random field elements *eval_values* and computes the group values *eval_group* of those. The secret key α and the public key PK are generated using the *Setup* function. Once these primitives are generated, the algorithm can compute the commitment C, which is $\mathbf{g}^{\phi(\alpha)}$, where ϕ is the Lagrange interpolation of the points list $(i,a)\#(\text{zip } I \text{ } eval\_values)$. We omit further details on the interpolation as that would go beyond the scope, for the exact computation, we refer the reader to the function *interpolate_on* of the KZG_hiding theory in the formalization. Using the commitment, valid witnesses can be created for the points in the list 'zip I *eval_values*' except for the event that $\alpha \in I$, which has negligible probability (see 5.1.3 for details). The hiding game adversary $\mathcal{A}$ is supplied with the commitment C to ϕ and the t witness tuples *wtns_tpls* and returns the guessed polynomial ϕ'. The result of the algorithm is then the polynomial ϕ' evaluated at i. Note if the adversary $\mathcal{A}$ can break the hiding property i.e. guess the polynomial to which C is the commitment and the t witness tuples belong, then $\phi' = \phi$ and thus ϕ' evaluated at i is a. Furthermore, returning a for the DL-instance $\mathbf{g}^a$ breaks the DL-assumptions, hence this reduction is correct.

Additionally, we outline how we formalized the game-hop based on a failure event, for the exact proof details see the *KZG_hiding* theory. Our formalization follows the approach from the CryptHOL tutorial[LS18], which uses the fundamental lemma to bind the probability of winning one game to the probability of winning a similar game except for a possible failure event and externalizes the probability of the failure event [LS18]. We define $\alpha \in I$ as the failure event and show that we can exchange the witness creation in the games except for the probability that $\alpha \in I$ i.e. the games are equivalent except for the failure event. Using the fundamental lemma we conclude:

```
theorem fundamental_lemma_game1_game2:
    "spmf (game2 I 𝒜) True + (max_deg+1)/p ≥ spmf (game1 I 𝒜) True"
```

where game1 is the DL-game with an inlined reduction adversary and game2 is the game that is equivalent to game1 except that it uses *CreateWitness* to create the witnesses. Furthermore, note that $max\_deg = t$ and thus $(max\_deg + 1)/p$ is negligible.

5.1.4 Knowledge Soundness

We formalize the property *knowledge soundness* following 3.5 in the AGM. Formally we define *knowledge soundness* in the AGM as the following game with an efficient extractor algorithm E against an efficient AGM adversary $\mathcal{A} = (\mathcal{A}', \mathcal{A}'')$:

$$\left(\begin{array}{r}
PK \leftarrow \text{KeyGen}, \\
(C, calc\_vec, \sigma) \leftarrow \mathcal{A}'(PK), \\
\_ :: \text{unit} \leftarrow \text{assert\_spmf}\left(\text{len}(PK) = \text{len}(calc\_vec) \wedge C = \prod_{1}^{len(calc\_vec)} PK_i^{calc\_vec_i}\right), \\
\phi = E(C, calc\_vec), \\
(i, \phi_i, \omega_i) \leftarrow \mathcal{A}''(\sigma, PK, C, calc\_vec), \\
: \ \phi(i) \neq \phi_i \wedge \text{VerifyEval}(PK, C, i, \phi_i, \omega_i)
\end{array}\right)$$

We omit the AGM vector for w_i as we do not need it for our proof, for completeness one can think of it as an implicit output that is never used.

Original Paper Proof

There are multiple proofs for knowledge soundness in different SNARK schemes (see e.g. Plonk[GWC19] and Halo[BDFG20]). We will not discuss them in detail but note one important property, all of them share: they use an additional security assumption to prove knowledge soundness, for example, the Q-DLOG assumption in the case of Plonk and Halo. We provide a proof that exploits the evaluation binding property of the KZG to prove knowledge soundness, without adding a security assumption:

Firstly, note that *calc_vec* provides exactly the coefficients one would need to obtain from a polynomial one wants to commit to. Thus E can return a polynomial ϕ that has exactly the coefficients from *calc_vec*. Since we know $C = \text{Commit}(PK, \phi)$ and the KZG is correct, we can conclude that for every value i, $\text{VerifyEval}(PK, C, i, \phi, \phi(i))$ must hold. Hence, if the adversary $\mathcal{A}''$ can provide a ϕ_i, such that $\phi_i \neq \phi(i)$ and $\text{VerifyEval}(PK, C, i, \phi, \phi_i)$, the evaluation binding property is already broken, because VerifyEval verifies for two different evaluations at the same point of a polynomial.

Sequence-of-games proof

We reduce the knowledge-soundess game to the evaluation binding game. Based on the idea from 5.1.4, we formally define the following reduction adversary to the evaluation binding game, given the knowledge soundness adversary $\mathcal{A} = (\mathcal{A}', \mathcal{A}'')$, the extractor E, and the input for the evaluation binding adversary; the public key PK:

$$\left(\begin{array}{r}
(C, calc\_vec, \sigma) \leftarrow \mathcal{A}'(PK), \\
\phi = E(C, calc\_vec), \\
(i, \phi_i, \omega_i) \leftarrow \mathcal{A}''(\sigma, PK, C, calc\_vec), \\
\phi_i' = \phi(i), \\
\omega_i' = \text{CreateWitness}(PK, \phi, i), \\
(C, i, \phi_i, \omega_i, \phi_i', \omega_i')
\end{array}\right)$$

The reduction creates a tuple $(C, i, \phi_i, \omega_i, \phi_i', \omega_i')$ to break the evaluation binding property. The commitment C is determined by the adversary $\mathcal{A}'$, from which messages the extractor E also computes the polynomial ϕ, to which C is a commitment (see 5.1.4). The position i, as well as the first evaluation ϕ_i and witness ω_i are provided by the adversary $\mathcal{A}''$. The second evaluation $\phi_i' = \phi(i)$ and witness $\omega_i' = \text{CreateWitness}(PK, \phi, i)$ are computed from ϕ. Note, if the knowledge soundness adversary is efficient and correct, then $\phi_i \neq \phi_i'$ and $\text{VerifyEval}(PK, C, i, \phi_i, \omega_i) \wedge \text{VerifyEval}(PK, C, i, \phi_i', \omega_i')$ holds, hence the reduction is a correct and efficient adversary for evaluation binding.

For the game-based proof note that the knowledge soundness game and the evaluation binding game with the reduction adversary are equivalent except for the asserts: while the knowledge soundness game asserts

$$\phi(i) \neq \phi_i \wedge \text{VerifyEval}(PK, C, i, \phi_i, \omega_i)$$

the evaluation binding game asserts

$$\phi_i \neq \phi_i' \wedge \text{VerifyEval}(PK, C, i, \phi_i, \omega_i) \wedge \text{VerifyEval}(PK, C, i, \phi_i, \omega_i)$$

where $\phi(i) = \phi_i$. Furthermore, note that the two statements are equivalent since VerifyEval for $\phi(i)$ is trivially true. Hence, the game-based proof is effectively equational reasoning over the asserts. The complete game-based proof is to be found in the Isabelle theory $KZG\_knowledge\_soundness$.

Formalization

The goal of this proof is to show the following theorem, which states that the probability of any adversary breaking knowledge soundness is less than or equal to breaking the binding property (using a reduction adversary):

```
theorem knowledge_soundness_eval_bind: "knowledge_soundness_advantage 𝒜
    ≤ eval_bind_advantage (reduction 𝒜)"
```

Moreover, since we already formalized the theorem *evaluation_binding* (i.e. that the probability of breaking evaluation binding is less than or equal to breaking the t-SDH assumption), we get the following theorem through transitivity, given the *knowledge_soundness_eval_bind* theorem holds:

```
theorem knowledge_soundness: "knowledge_soundness_advantage 𝒜
    ≤ t_SDH.advantage (bind_reduction 𝒜)"
```

Since the AGM is not yet formalized we cannot simply instantiate the knowledge soundness game formalized in 3.5 but have to manually adjust the types to resemble the AGM. We formalize the knowledge soundness game in CryptHOL as follows:

```
TRY do {
    PK ← KeyGen;
    (C, calc_vec) ← 𝒜' PK;
    _ :: unit ← assert_spmf (length PK = length calc_vec
        ∧ C = fold (λ i acc. acc · PK!i ^ (calc_vec!i)) [0..<length PK] 1);
    let ϕ = E C calc_vec;
    (i,ϕ_i, w_i) ← 𝒜 PK C calc_vec;
    _::unit ← assert_spmf(valid_msg ϕᵢ, w_i);
    return_spmf (VerifyEval PK C i ϕ_i w_i ∧ ϕ_i ≠ poly ϕ i)
} ELSE return_spmf False
```

The game captures the spmf over True and False, which represent the events that the adversary has broken knowledge soundness or not. Firstly, the public key PK is generated using $KeyGen$. Secondly, the adversary $\mathcal{A}'$ provides a commitment C in the algebraic group model i.e. with the vector *calc_vec*, which constructs C from PK. We use an assert to ensure in Isabelle that the message of the $\mathcal{A}'$ is correct according to the AGM. Thirdly, the extractor E computes a polynomial ϕ given access to the messages of $\mathcal{A}'$. The second part of the adversary, $\mathcal{A}''$, computes a position i, an evaluation ϕ_i for that position, and a respective witness w_i. We use an assert to check that the messages of $\mathcal{A}''$ are valid, that is to ensure in Isabelle that w_i is a group element. The adversary wins the game if $\phi\_i \neq \phi(i)$ and VerifyEval(PK, C, i, ϕ_i, w_i) hold.

We formalize the reduction adversary to the binding game as defined in 5.1.4:

```
fun reduction (𝒜',𝒜'') PK = do {
    (C, calc_vec) ← 𝒜'PK;
    _ :: unit ← assert_spmf (length PK = length calc_vec
        ∧ C = fold (λ i acc. acc · PK!i ^ (calc_vec!i)) [0..<length PK] 1);
    let ϕ = E C calc_vec;
    (i,ϕ_i, w_i) ← 𝒜 PK C calc_vec;
    _::unit ← assert_spmf(valid_msg ϕᵢ, w_i);
    return_spmf (C, i, ϕ_i, w_i, ϕ_i',w_i')
}
```

This function mirrors exactly the outline in 5.1.4 except for the validity checks of the adversary's messages in the form of the asserts (see the game definition above for details), thus we skip a detailed description.

5.2 Batched KZG Proofs

In this section, we outline the security properties for the batched version of the KZG as well as the idea behind the formalization of each security property. Again, as in chapter 5.1, we will not give concrete proofs in Isabelle syntax as this would go beyond the scope due to a lot of boilerplate Isabelle-specific abbreviations and functions.

Similarly to chapter 5.1, each section covers one security property. Firstly, we describe the paper proof, followed by the game-based proof. Lastly, we outline the formalization, specifically the security game, the reduction algorithm and if needed additional formalization details.

Note, the property of *polynomial-binding* does not involve any partially opening functions (i.e. neither CreateWitness nor CreateWitnessBatch) and hence does not change for the batched version. The property thus holds automatically for the batched version.

5.2.1 Evaluation Binding

In this subsection we outline the *evaluation binding* proof given in [KZG10], transform it into a sequences-of-games format and outline our formal verification of that proof. The following game captures the abstract *evaluation binding* game, which we're using to proof evaluation binding, instantiated with the batched KZG functions, played against an efficient adversary $\mathcal{A}$:

$$\left(\begin{array}{rl} PK \leftarrow & \text{KeyGen}, \\ (C, i, \phi_i, \omega_i, B, r(x), \omega_B) \leftarrow & \mathcal{A}(PK), \\ \_ :: \text{unit} \leftarrow & \text{assert\_spmf}(i \in B \wedge \phi_i \neq r(x)), \\ b = & \text{VerifyEval}(PK, C, i, \phi_i, \omega_i), \\ b' = & \text{VerifyEvalBatch}(PK, C, B, r(x), \omega_B), \\ : & b \wedge b' \end{array}\right)$$

Intuitively this game expresses that no adversary can find an $r(x)$ (with a witness) and an evaluation (with a witness) that diverges from the evaluation of $r(x)$ at any $i \in B$.

Original Paper Proof

The paper proof is similar to the evaluation binding proof outlined in 5.1.2. Essentially, the values provided by the adversary, if they are correct i.e. accepted by VerifyEval and respectively VerifyEvalBatch, can be rearranged to obtain the result (the secret key) for the t-BSDH instance, which is the public key. Nevertheless, we outline the paper proof concretely for completeness:

The paper proof argues that given an adversary $\mathcal{A}$, that can break the evaluation binding property, an algorithm $\mathcal{B}$ can be constructed, that can break the t-BSDH assumption [KZG10]. The concrete construction for $\mathcal{B}$ is: given the t-BSDH instance $tsdh\_inst = (\mathbf{g}, \mathbf{g}^{\alpha}, \mathbf{g}^{\alpha^2}, \ldots, \mathbf{g}^{\alpha^t})$, call $\mathcal{A}$ with $tsdh\_inst$ as PK for $(C, B, r(x), \omega_B, i \in B, \phi(i), \omega_i)$ and return:

$$\left(e\left(\mathbf{g}^{p'(\alpha)}, \omega_B\right) \oplus e\left(\frac{\mathbf{g}^{r'(\alpha)}}{\omega_i}, \mathbf{g}\right)\right)^{\frac{1}{\phi(i)-r(i)}}$$

where $p'(x)$ is the product polynomial $\prod_{j \in B \setminus \{i\}}(x-j) = \frac{\prod_{j \in B}(x-j)}{(x-i)}$ and $r'(x) = \frac{r(x)-r(i)}{(x-i)}$ [KZG10]. The reason to why $\mathcal{B}$ is a correct construction is the following:

Breaking evaluation binding means, that $\mathcal{A}$, given a valid public key PK, can give a Commitment C and two witness-tuples, $\langle i, \phi(i), \omega_i \rangle$ and $\langle B, r(x), \omega_B \rangle$, where $i \in B$, such that $VerifyEval(PK, C, \langle i, \phi(i), \omega_i \rangle)$ and $VerifyEvalBatch(PK, C, \langle B, r(x), \omega_B \rangle)$ return true [KZG10]. Since *VerifyEvalBatch*, as well as *VerifyEval*, is a pairing check against $e(C, \mathbf{g})$ we can conclude that:

$$e(\omega_i, \mathbf{g}^{\alpha - i}) e(\mathbf{g}, \mathbf{g})^{\phi(i)} = e(C, \mathbf{g}) = e(\mathbf{g}^{\prod_{i \in B}(\alpha - i)}, \omega_B) e(\mathbf{g}, \mathbf{g}^{r(\alpha)})$$

which is the pairing term for:

$$\begin{aligned}
&\psi_i(\alpha)\cdot(\alpha - i) + \phi(i) = \prod_{j\in B}(\alpha - j)\cdot\psi_B(\alpha) + r(\alpha) \\
\iff &\phi(i) - r(\alpha) = \prod_{j\in B}(\alpha - j)\cdot\psi_B(\alpha) - \psi_i(\alpha)\cdot(\alpha - i) \\
\iff &\phi(i) - r(\alpha) = \frac{\prod_{j\in B}(\alpha - j)}{(\alpha - i)}\cdot(\alpha - i)\cdot\psi_B(\alpha) - \psi_i(\alpha)\cdot(\alpha - i) \\
\iff &\phi(i) - r(\alpha) = p'(\alpha)\cdot(\alpha - i)\cdot\psi_B(\alpha) - \psi_i(\alpha)\cdot(\alpha - i) \\
\iff &\phi(i) - r(\alpha) = (p'(\alpha)\cdot\psi_B(\alpha) - \psi_i(\alpha))\cdot(\alpha - i) \\
\iff &\phi(i) - (r'(\alpha)\cdot(\alpha - i) + r(i)) = (p'(\alpha)\cdot\psi_B(\alpha) - \psi_i(\alpha))\cdot(\alpha - i) \\
\iff &\phi(i) - r(i) = (p'(\alpha)\cdot\psi_B(\alpha) - \psi_i(\alpha) + r'(\alpha))\cdot(\alpha - i) \\
\iff &\frac{1}{(\alpha - i)} = \frac{p'(\alpha)\cdot\psi_B(\alpha) - \psi_i(\alpha) + r'(\alpha)}{\phi(i) - r(i)}
\end{aligned}$$

where $\psi_i(\alpha) = log_{\mathbf{g}}\ \omega_i$ and $\psi_B(\alpha) = log_{\mathbf{g}}\ \omega_B$ and omitting the $e(C, \mathbf{g})$ [KZG10].

Hence:

$$\begin{aligned}
&\left(e\left(\mathbf{g}^{p'(\alpha)}, \omega_B\right) \oplus e\left(\frac{\mathbf{g}^{r'(\alpha)}}{\omega_i}, \mathbf{g}\right)\right)^{\frac{1}{\phi(i)-r(i)}} \\
&= \left(e\left(\mathbf{g}^{p'(\alpha)}, \mathbf{g}^{\psi_B(\alpha)}\right) \oplus e\left(\frac{\mathbf{g}^{r'(\alpha)}}{\mathbf{g}^{\psi_i(\alpha)}}, \mathbf{g}\right)\right)^{\frac{1}{\phi(i)-r(i)}} \\
&= \left(e(\mathbf{g}, \mathbf{g})^{p'(\alpha)\cdot\psi_B(\alpha)} \oplus e(\mathbf{g}, \mathbf{g})^{r'(\alpha)-\psi_i(\alpha)}\right)^{\frac{1}{\phi(i)-r(i)}} \\
&= \left(e(\mathbf{g}, \mathbf{g})^{p'(\alpha)\cdot\psi_B(\alpha)+r'(\alpha)-\psi_i(\alpha)}\right)^{\frac{1}{\phi(i)-r(i)}} \\
&= e(\mathbf{g}, \mathbf{g})^{\frac{p'(\alpha)\cdot\psi_B(\alpha)+r'(\alpha)-\psi_i(\alpha)}{\phi(i)-r(i)}} \\
&= e(\mathbf{g}, \mathbf{g})^{\frac{1}{\alpha-i}}
\end{aligned}$$

Since $e(\mathbf{g}, \mathbf{g})^{\frac{1}{\alpha-i}}$ breaks the t-BSDH assumption for i, $\mathcal{B}$ is correct []KZG.

Sequence-of-games Proof

The transformation into a sequence-of-games proof is analog to 5.1.2.

A look at the *evaluation binding* and the *t-BSDH* games reveals that *KeyGen*'s generation of PK is equivalent to generating a t-BSDH instance. Furthermore, the games differ only in their checks in the respective *assert_spmf* calls (and the adversary's return types). Additionally, we know from the paper proof that the adversary's messages, if correct and wellformed, which is checked in the eval_bind game's asserts, already break the t-BSDH assumptions on PK. Hence we give the following idea (which is analog to 5.1.2) for the proof:

1. rearrange the eval_bind game to accumulate (i.e. conjuncture) the return-check and all other checks into an assert
2. derive that this conjuncture of statements already implies that the t-BSDH is broken and add that fact to the conjuncture.
3. erase every check in the conjuncture by over-estimation, to be only left with the result that the t-BSDH is broken.

The resulting game is the t-BSDH game with the reduction adversary. See the Isabelle theory *KZG_BatchOpening_bind* for the full formal proof.

Formalization

The goal of this proof is to show the following theorem, which states that the probability of any adversary breaking evaluation binding is less than or equal to winning the DL game (using the reduction adversary) in a game-based proof:

```
theorem batchOpening_binding: "bind_advantage 𝒜
    ≤ t_BSDH.advantage (bind_reduction 𝒜)"
```

The instantiated evaluation binding game in CryptHOL looks as follows:

```
TRY do {
    PK ← KeyGen;
    (C, i, ϕ_i, w_i, B, w_B, r_x) ← 𝒜 PK;
    _::unit ← assert_spmf(i ∈ B ∧ ϕ_i ≠ poly r_x i
        ∧ valid_msg ϕ_i w_i ∧ valid_batch_msg r_x w_B B);
    let b = VerifyEval PK C i ϕ_i w_i;
    let b' = VerifyEvalBatch PK C B r_x w_B;
    return_spmf (b ∧ b')
} ELSE return_spmf False
```

The game captures the spmf over True and False, which represent the events that the adversary has broken evaluation binding or not. The public key *PK* is generated using the formalized *KeyGen* function of the KZG. The adversary $\mathcal{A}$, given PK, outputs values to break the evaluation binding game, namely a commitment value C, a set of positions B, that are to be opened, a polynomial r_x which evaluates to the claimed evaluations of ϕ (the polynomial, C is the commitment to) at the positions in B, a witness w_B that validates that the points $(i, r\_x(i))$ for all $i \in B$ are valid for C, a point $i \in B$, a claimed value ϕ_i that should be different from $r\_x(i)$ and a witness, w_i for the point (i, ϕ_i). Note that we use *assert_spmf* to ensure that the adversary's messages are wellformed and correct, where correct means that $i \in B$ and ϕ_i and the evaluation of r_x at i are indeed two different values. Should the assert not hold, the game is counted as lost for the adversary. We assign to b and b' the result of the formalized *VerifyEvalBatch* and respectively *VerifyEval* algorithm of the KZG. Evaluation binding is broken if and only if b and b' hold i.e. both witnesses and verifying checks pass and thus the same commitment can efficiently be resolved to two different values at some point.

We formally define the reduction adversary as follows:

```
fun bind_reduction 𝒜 PK = do {
    (C, i, ϕ_i, w_i, B, w_B, r_x) ← 𝒜 PK;
    let p' = g_pow_PK_Prod PK (prod (λi. [:-i,1:]) B div [:-i,1:]);
    let r' = g_pow_PK_Prod PK ((r_x - [:poly r_x i:]) div [:-i,1:]);
    return_spmf (-i, (e p' w_B ⊕ e (r' ÷ w_i) g)^(1/(ϕ_i - poly r_x i)))
}
```

That is a higher-order function, that takes the evaluation binding adversary $\mathcal{A}$ and returns an adversary for the t-BSDH game. That is, the function that calls the adversary $\mathcal{A}$ on some public key PK and returns $\left(-i, \left(e\left(\mathbf{g}^{p'(\alpha)}, \omega_B\right) \oplus e\left(\frac{\mathbf{g}^{r'(\alpha)}}{\omega_i}, \mathbf{g}\right)\right)^{\frac{1}{\phi(i)-r(i)}}\right)$, the solution to the t-BSDH game for $-i$.

The term *[:poly r_x i:]* is Isabelle's notation for a constant polynomial with the constant value of the polynomial r_x evaluated at i. The term $[: -i, 1 :]$ is Isabelle's notation for the polynomial $(x - i)$ and *prod (λ. [:-i,1:]) B* is the product term $\prod_{i \in B}(x - i)$. The *g_pow_PK_Prod* function returns the group generator $\mathbf{g}$, exponentiated with the evaluation of a polynomial on the secret key α, hence, $p' = \mathbf{g}^{\frac{\prod_{i \in B}(x-i)}{(x-i)}}$ and $r' = \mathbf{g}^{\frac{r_x - r_x(i)}{(x-i)}}$.

5.2.2 Weak Hiding

The authors of [KZG10] claim that this property automatically holds for the batched-KZG version since it does not expose more information than the standard DL-version fo the KZG (for which we already formally verifeid that it holds). In this subsection we discuss our thoughts on extending the weak hiding property shown for the KZG in 5.1.3 to the batched version. We present two possible approaches to proving this property and outline why we think neither of them will work.

Note, that the definition of weak hiding does not unconditionally fulfill the *hiding* property as defined in 2.20 but requires additionally the polynomial to be uniformly randomly chosen to hold.

Formally we define the *weak hiding* game against an efficient adversary $\mathcal{A}$, where $B \subset \mathbb{Z}_p$ is an arbitrary set of size t, as follows:

$$\left(\begin{array}{r l} \phi \xleftarrow{\$} & \{\phi.\ \text{degree}(\phi) \leq t\}, \\ PK \leftarrow & \text{KeyGen}, \\ C = & \text{Commit}(PK, \phi), \\ wtns\_tpl = & \text{CreateWitnessBatch}(PK, \phi, B), \\ \phi' \leftarrow & \mathcal{A}(PK, C, wtns\_tpl), \\ : & \phi = \phi' \end{array}\right)$$

Original Paper Proof

The original paper [KZG10] does not provide a proof for this property. The authors argue that the function *CreateWitnessBatch* does not reveal any more information than *CreateWitness* and thus the property is already shown by the hiding proof for the standard KZG [KZG10]. However, this argument is not formally satisfying, which is why we discuss some approaches to proving this property in the following section.

Sequence-of-games Proof

The goal of the hiding proof is to show the following theorem, which states that the probability of any adversary breaking hiding is less than or equal to winning the DL game (using the reduction adversary) in a game-based proof:

```
theorem batch_hiding: "batch_hiding_advantage 𝒜
    ≤ t_SDH.advantage (reduction 𝒜)"
```

We focus on two approaches to proving the hiding property for the batch version:

1. reducing the batched hiding game to the normal hiding game
2. adapting the normal hiding proof to the batched version

Firstly, we argue why we do not think that the hiding game for the batched version can be reduced to the hiding game of the normal KZG. We outline specifically the difficulties in transitioning from *CreateWitnessBatch* to *CreateWitness*. Secondly, we argue why the hiding proof for the standard KZG cannot be trivially adapted to the batched version.

Reducing Batched Hiding to Normal Hiding

For this approach, we aim to show the following theorem:

```
theorem batch_hiding_reduction: "batch_hiding_advantage 𝒜
    ≤ hiding_advantage (hiding_reduction 𝒜)"
```

Essentially, this boils down to creating a reduction adversary *hiding_reduction* that wins the batch hiding game using the adversary for the normal hiding game. Hence emulating the normal hiding game in the reduction adversary.

Note, that the majority of the inputs for the adversaries are equal, they take firstly the public key PK and secondly a commitment C. The only value they differ in is the witness tuple. Specifically, the normal hiding game computes the $|I|$ many witnesses for the adversary as 'map (CreateWitness(PK, ϕ, i)) I', while the batched version computes one witness tuple as 'CreateWitnessBatch(PK, ϕ, B)' for its respective adversary.

Hence, CreateWitness(PK, ϕ, i) would need to be emulated using only the reduction adversary's inputs to emulate the normal hiding game in the reduction adversary for the batched version. One would need to emulate CreateWitness(PK, ϕ, i) for all $i \in I$ using only PK, C, and the result from CreateWitnessBatch(PK, ϕ, B).

As a little reminder, the result from *CreateWitness* is a tuple $(i, \phi(i), \omega_i)$, where ω_i is the wintess for the point $(i, \phi(i))$. The result from *CreateWitnessBatch* is a tuple $(B, r(x), \omega_B)$, where B is a set of values, $r(i) = \phi(i)$ for all $i \in B$, and ω_B is a witness for all points $(i, r(i))$ for $i \in B$. Furthermore, note $\omega_i = \mathbf{g}^{\psi_i(\alpha)}$ where $\psi_i(x) = \frac{\phi(x)-\phi(i)}{(x-i)}$ and $\omega_B = \mathbf{g}^{\psi_B(\alpha)}$ where $\psi_B(x) = \frac{\phi(x)-r(x)}{\prod_{i\in B}(x-i)}$ and $r(x) = \phi(x) \bmod \prod_{i\in B}(x-i)$.

Converting $(B, r(x))$ in $|B|$ many points $(i, \phi(i))$ for $i \in B$ is trivial. The challenge is to compute $\mathbf{g}^{\psi_i(\alpha)}$ from $PK, C, B, r(x)$, and $\mathbf{g}^{\psi_B(\alpha)}$.

To naively compute $\mathbf{g}^{\psi_i(\alpha)} = \mathbf{g}^{\frac{\phi(\alpha)-\phi(i)}{(\alpha-i)}}$ one would need to know the value of $(\alpha - i)$ and the group values for $\phi(\alpha)$ and $\phi(i)$. Note, that we can compose group values for addition and subtraction in the exponents, but no multiplication or division. While we do know the group values of $\phi(\alpha)$ and $\phi(i)$ from $C = \mathbf{g}^{\phi(\alpha)}$ and $r(i) = \phi(i)$, we do not know the value of $(\alpha - i)$. The scheme reveals only group values dependent on α (the commitment C, the witness ω_B and the public key PK), but no field values, thus α is secure by the DL assumption. Hence, constructing a field value dependent on α, like $\alpha - i$, from group values is also impossibly hard by DL assumption.

One could ideate to somehow carve ψ_i out of ψ_B, as $\psi_B = \mathbf{g}^{\frac{\phi(\alpha)-r(\alpha)}{\prod_{i\in B}(\alpha-i)}}$ already includes the divions by $\alpha - i$. However, this would require knowledge of $\prod_{i\in B\setminus\{i\}}(\alpha - i)$, which again is a field element that dependents on α. Hence this approach is also impossible by DL assumption.

Thus we do not think that the batched hiding game can be reduced to the normal hiding game.

Adapting the Normal Hiding Proof

For this approach, we aim to show the following theorem, which is the one firstly outlined in this section:

```
theorem batch_hiding: "batch_hiding_advantage 𝒜
    ≤ t_SDH.advantage (reduction 𝒜)"
```

Note, that the batched hiding game is equal to the normal hiding game except for the computation of the witnesses. While the normal hiding game computes them as 'map (CreateWitness(PK, ϕ, i)) I', the batched version computes them as
'CreateWitnessBatch(PK, ϕ, B)'.

The normal reduction proof, outlined in 5.1.3, emulates *CreateWitness* without knowing the full polynomial. Only t points of the polynomial and one additional group point (i.e. $(i, \mathbf{g}^{\phi(i)})$) are known. Still t correct points $(i, \phi(i))$ can be outputted as part of the result of *CreateWitness*, even though the full polynomial is unknown.

However, *CreateWitnessBatch* does not output t points for the evaluations, but the remainder $r(x) = \phi(x) \bmod \prod_{i\in B}(x-i)$. Since the $t+1$th group point is hidden by DL assumption, the full polynomial $\phi(x)$ cannot be known and thus the remainder is not trivially computable. Hence, the result of *CreateWitnessBatch* cannot be trivially emulated. Thus the hiding proof from 5.1.3 is not trivially adaptable to the batched hiding game. Furthermore, note that all information about $\phi(x)$, like evaluations, except for the t known points, are provided in a grouped manner i.e. as $\mathbf{g}^{\phi(\alpha)}$. Thus intuitively we do not think it is possible to derive a field element-based value, like the remainder $r(x)$, that depends on more information about ϕ than the t known points due to the DL assumption.

For now, it remains an open question whether this property can be proven or not.

5.2.3 Knowledge Soundness

We extend the knowledge soundness property as defined in 2.19 and proved for the KZG in 5.1.4 to the batched KZG version. Formally we define *knowledge soundness* as the following game against an efficient AGM adversary $\mathcal{A} = (\mathcal{A}', \mathcal{A}'')$ and an efficient extractor E:

$$\left(\begin{array}{r}PK \leftarrow \text{KeyGen}, \\ (C, \mathit{calc\_vec}, \sigma) \leftarrow \mathcal{A}'(PK), \\ \_ :: \text{unit} \leftarrow \text{assert\_spmf}\left(\text{len}(PK) = \text{len}(\mathit{calc\_vec}) \wedge C = \prod_{1}^{\mathit{len}(\mathit{calc\_vec})} PK_i^{\mathit{calc\_vec}_i}\right), \\ \phi = E(C, \mathit{calc\_vec}), \\ (i, B, r(x), \omega_B) \leftarrow \mathcal{A}''(\sigma, PK, C, \mathit{calc\_vec}), \\ : \ \phi(i) \neq r(i) \wedge \text{VerifyEvalBatch}(PK, C, B, r(x), \omega_B)\end{array}\right)$$

We omit the AGM vector for w_B as we do not need it for our proof, for completeness one can think of it as an implicit output that is never used.

Original Paper Proof

The proof is analog to 5.1.4:

Firstly, note that *calc_vec* provides exactly the coefficients one would need to obtain from a polynomial one wants to commit to. Thus E can return a polynomial ϕ that has exactly the coefficients from *calc_vec*. Since we know $C = \text{Commit}(PK, \phi)$ and the KZG is correct, we can conclude that for every value i, $\text{VerifyEval}(PK, C, i, \phi, \phi(i))$ must hold. Hence, if the adversary $\mathcal{A}''$ can provide B and $r(x)$, such that $i \in B$, $r(i) \neq \phi(i)$ and $\text{VerifyEvalBatch}(PK, C, B, r(x), \omega_B)$ hold, the evaluation binding property is already broken because VerifyEval and VerifyEvalBatch verify for two different evaluations at the same point of a polynomial.

Sequence-of-games Proof

The transformation into a sequence-of-games proof is analog to 5.1.4: We reduce the knowledge soundness game to the evaluation binding game (batched version).

Based on the idea from 5.2.3, we formally define the following reduction adversary to the evaluation binding game, given the knowledge soundness adversary $\mathcal{A} = (\mathcal{A}', \mathcal{A}'')$, the extractor E, and the input for the evaluation binding adversary; the public key PK:

$$\left(\begin{array}{r}(C, \mathit{calc\_vec}, \sigma) \leftarrow \mathcal{A}'(PK), \\ \phi = E(C, \mathit{calc\_vec}), \\ (i, B, r(x), \omega_B) \leftarrow \mathcal{A}''(\sigma, PK, C, \mathit{calc\_vec}), \\ \phi_i = \phi(i), \\ \omega_i = \text{CreateWitness}(PK, \phi, i), \\ (C, i, \phi_i, \omega_i, B, r(x), \omega_B)\end{array}\right)$$

The reduction creates a tuple $(C, i, \phi_i, \omega_i, B, r(x), \omega_B)$ to break the evaluation binding property. The commitment C is determined by the adversary $\mathcal{A}'$, from which messages the extractor E also computes the polynomial ϕ, to which C is a commitment (see 5.1.4). The adversary $\mathcal{A}'$ provides a set of positions B, the claimed evaluations of $\phi(x)$ to the positions in B captured in the polynomial $r(x)$ (i.e. $\forall i \in B.\ r(i) \overset{!}{=} \phi(i)$), a witness ω_B for $r(x)$ and B (i.e. such that $VerifyEvalBatch(PK, C, B, r(x), \omega_B)$ holds), and a position $i \in B$ for that it claims that $\phi(i) \neq r(i)$. Then the real evaluation of ϕ on i, $\phi(i)$ is computed as ϕ_i and a witness ω_i for the point $(i, \phi(i))$ is computed using $CreateWitness$. Note, if the knowledge soundness adversary is correct, then $\phi_i = \phi(i) \neq r(x)$ and $\text{VerifyEval}(PK, C, i, \phi_i, \omega_i) \wedge \text{VerifyEvalBatch}(PK, C, B, r(x), \omega_B)$ holds. Hence the reduction is a correct and efficient adversary for evaluation binding.

For the sequence-of-games proof, note, that the knowledge soundness game and the evaluation binding game with the reduction adversary are equivalent except for the asserts: while the knowledge soundness game asserts

$$\phi(i) \neq r(i) \wedge \text{VerifyEvalBatch}(PK, C, B, r(x), \omega_B)$$

the evaluation binding game asserts

$$\phi_i \neq r(i) \wedge \text{VerifyEval}(PK, C, i, \phi_i, \omega_i) \wedge \text{VerifyEvalBatch}(PK, C, B, r(x), \omega_B)$$

where $\phi(i) = \phi_i$. Furthermore, note that the two statements are equivalent since VerifyEval for $\phi(i)$ is trivially true. Hence, the sequence-of-games proof is effectively equational reasoning over the asserts. The complete sequence-of-games proof is to be found in the Isabelle theory *KZG_BatchOpening_knowledge_soundness*.

Formalization

The goal of this proof is to show the following theorem, which states that the probability of any efficient AGM adversary breaking knowledge soundness is less than or equal to breaking the evaluation binding property (using a reduction adversary):

```
theorem knowledge_soundness_game_eq_bind_game_knowledge_soundness_reduction:
    "knowledge_soundness_game E 𝒜′ 𝒜″
     = bind_game (knowledge_soundness_reduction E 𝒜′ 𝒜″)"
```

Moreover, since we already formalized the theorem *evaluation_binding* (i.e. that the probability of breaking evaluation binding is less than or equal to breaking the t-BSDH assumption), we get the following theorem through transitivity, given the *knowledge_soundness_game_eq_bind_game$_k$nowledge_soundness_reduction* theorem holds:

```
theorem knowledge_soundness: "knowledge_soundness_advantage 𝒜
    ≤ t_BSDH.advantage (bind_reduction (knowledge_soundness_reduction 𝒜))"
```

Similarly to 5.1.3, we cannot instantiate the knowledge soudness game defined in 3.5 directly, since the AGM is not formaliuzed yet. Instead, following 5.1.3 we redefine the types manually to resemble the AGM - we formalize the knowledge soundness game in CryptHOL as follows:

```
TRY do {
    PK ← KeyGen;
    (C, calc_vec) ← 𝒜′ PK;
    _ :: unit ← assert_spmf (length PK = length calc_vec
        ∧ C = fold (λ i acc. acc · PK!i ^ (calc_vec!i)) [0..<length PK] 1);
    let ϕ = E C calc_vec;
    (i, B, r_x, w_B) ← 𝒜 PK C calc_vec;
    _::unit ← assert_spmf(i∈B ∧ valid_batch_msg r_x w_B B);
    return_spmf (VerifyEvalBatch PK C B r_x w_B ∧ poly r_x i ≠ poly ϕ i)
} ELSE return_spmf False
```

The game captures the spmf over True and False, which represent the events that the adversary has broken knowledge soundness or not. Firstly, the public key PK is generated using *KeyGen*. Secondly, the adversary $\mathcal{A}'$ provides a commitment C in the algebraic group model i.e. with the vector *calc_vec*, which constructs C from PK. We use an assert to ensure in Isabelle that the message of the $\mathcal{A}'$ is correct according to the AGM. Thirdly, the extractor E computes a polynomial ϕ given access to the messages of $\mathcal{A}'$. The second part of the adversary, $\mathcal{A}''$, computes a set of position B, an evaluation polynomial r_x that captures the claimed evaluations of ϕ on the positions B (i.e. $\forall i \in B.\ \phi(i) \overset{!}{=} r(i)$), a witness w_B for B and r_x, and a position $i \in B$ for which $\phi(i) \neq r(i)$ should hold. We use an assert to check that the messages of $\mathcal{A}''$ are valid, that is to ensure in Isabelle that w_B is a group element and $i \in B$. The adversary wins the game if and only if $r(x) \neq \phi(i)$ and $\text{VerifyEvalBatch}(PK, C, B, r\_x, w\_B)$ hold.

We formalize the reduction adversary to the binding game as defined in 5.1.4:

```
fun reduction (𝒜′, 𝒜″) PK = do {
    (C, calc_vec) ← 𝒜′PK;
    _ :: unit ← assert_spmf (length PK = length calc_vec
        ∧ C = fold (λ i acc. acc · PK!i ^ (calc_vec!i)) [0..<length PK] 1);
    let ϕ = E C calc_vec;
    (i,ϕ_i, w_i) ← 𝒜 PK C calc_vec;
    _::unit ← assert_spmf(valid_msg ϕ_i, w_i);
    return_spmf (C, i, ϕ_i, w_i, ϕ_i’,w_i’)
}
```

This function mirrors exactly the outline in 5.2.3 except for the validity checks of the adversary's messages in the form of the asserts (see the game definition above for details), thus we skip a detailed description.

6 Conclusion

We presented the first formalization of Polynomial Commitment Schemes in an interactive theorem prover, provided concrete games for a unified security analysis of formally verfied PCSs. We instantiated and formally verfied two versions of the KZG PCS, the standard DL-version and the batched version, both outlined in [KZG10]. This work forms the first step towards formally verified SNARKs and ZKPs, still leaving a lot of promising future work open. As a first step, we want to finish this work by properly formalizing the AGM in Isabelle, following the ROM formalization by Basin et al. [BLS17].

6.1 Future Work

The logical next step would be to formalize Interactive Oracle Proofs (IOPs) together with the Fiat-Shamir (FS) transform [FS87] (IOPs + FS). IOPs + FS in combination with the existing formalization of the sum-check protocol [BBS24] and the formal verfication of the KZG from our work would allow for the formal verification of a range of KZG based SNARKs like Nova [KST21], Halo[BDFG20] and Plonk [GWC19] (Plonk and Halo even without the sum-check protocol). Furthermore IOPs + FS could be used to formalize hash-based PCS constructions, which typically use IOPs of Proximity (IOPPs) to proof distance to a code word (commonly some kind of Reed-Solomon code). Formalizing IOPs + FS and Reed-Solomon codes would enable formal verification of hash-based/code-based PCSs, like FRI [BBHR18], Binius[DP23], and Basefold[ZCF23], and in consequence formal verification of plausibly post-quantum secure SNARKs like STARKs [BSBHR18]. Since the mentioned code-based PCSs are multilinear if not mutlivariate this would also impose the formalization of multi-variate/multilinear polynomial commitment schemes (MPCS/MLPCS) as an extension of our formalization. Concrete instatiations of MPCS/MLPCS would be e.g. Zeromorph[KT23], Binius[DP23], and BaseFold[ZCF23].

Acknowledgements

We want to thank Katharina Heidler (formerly Kreuzer) for discussions and her formalization of the Elimination of Repeated Factors algorithm together with Manuel Eberl, whom we would also like to thank. Additonally, we want to thank Jens Ernstberger who gave the initial pointer that motivated this work.

References

[ACFY24] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: Reed–solomon proximity testing with super-fast verification. Cryptology ePrint Archive, Paper 2024/1586, 2024.

[AFLN23] Martin R. Albrecht, Giacomo Fenzi, Oleksandra Lapiha, and Ngoc Khanh Nguyen. SLAP: Succinct lattice-based polynomial commitments from standard assumptions. Cryptology ePrint Archive, Paper 2023/1469, 2023.

[AST23] Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. Cryptology ePrint Archive, Paper 2023/1217, 2023. `https://eprint.iacr.org/2023/1217`.

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[BBS24] Azucena Garvía Bosshard, Jonathan Bootle, and Christoph Sprenger. Formal verification of the sumcheck protocol, 2024.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Recursive zk-snarks from any additive polynomial commitment scheme. Cryptology ePrint Archive, Paper 2020/1536, 2020. https://eprint.iacr.org/2020/1536.

[BFL$^+$22] Vitalik Buterin, Dankrad Feist, Diederik Loerakker, George Kadianakis, Matt Garnett, Mofi Taiwo, and Ansgar Dietrichs. Eip-4844: Shard blob transactions. Ethereum Improvement Proposals, no. 4844, 2022. https://eips.ethereum.org/EIPS/eip-4844.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 677–706, Cham, 2020. Springer International Publishing.

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. *SIGPLAN Not.*, 44(1):90–101, jan 2009.

[BLAG19] David Butler, Andreas Lochbihler, David Aspinall, and Adria Gascon. Formalising σ-protocols and commitment schemes using crypthol. Cryptology ePrint Archive, Paper 2019/1185, 2019. https://eprint.iacr.org/2019/1185.

[BLS03] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi, editors, *Security in Communication Networks*, pages 257–267, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[BLS17] David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. Crypthol: Game-based proofs in higher-order logic. Cryptology ePrint Archive, Paper 2017/753, 2017. https://eprint.iacr.org/2017/753.

[BM23] Bolton Bailey and Andrew Miller. Formalizing soundness proofs of snarks. Cryptology ePrint Archive, Paper 2023/656, 2023. https://eprint.iacr.org/2023/656.

[BMM$^+$21] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 65–97, Cham, 2021. Springer International Publishing.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography*, pages 319–331, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BR04] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Paper 2004/331, 2004. https://eprint.iacr.org/2004/331.

[BS23] Dan Boneh and Victor Shoup. A graduate course in applied cryptography, 2023. http://toc.cryptobook.us/book.pdf.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018. https://eprint.iacr.org/2018/046.

[BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam Smith, editors, *Theory of Cryptography*, pages 31–60, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BSTZ] Yves Bertot, Matthieu Sozeau, Nicolas Tabareau, and Théo Zimmermann. Coq.

[Cad] David Cadé. Cryptoverif.

[CHM$^+$19] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. Cryptology ePrint Archive, Paper 2019/1047, 2019. https://eprint.iacr.org/2019/1047.

[Coia] CoinMarketCap. Starknet.

[Coib] CoinMarketCap. Zksync.

[cry] cryptoline. cryptoline.

[CY24] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024.

[DCX$^+$23] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 356–370, New York, NY, USA, 2023. Association for Computing Machinery.

[DJTY16] Jose Divasón, Sebastiaan J. C. Joosten, René Thiemann, and Akihisa Yamada. The factorization algorithm of berlekamp and zassenhaus. *Archive of Formal Proofs*, October 2016. https://isa-afp.org/entries/Berlekamp_Zassenhaus.html, Formal proof development.

[dMKA$^+$15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, pages 378–388, Cham, 2015. Springer International Publishing.

[DP23] Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. Cryptology ePrint Archive, Paper 2023/1784, 2023.

[FHAS24] Nils Fleischhacker, Mathias Hall-Andersen, and Mark Simkin. Extractable witness encryption for KZG commitments and efficient laconic OT. Cryptology ePrint Archive, Paper 2024/264, 2024.

[FKL17] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. Cryptology ePrint Archive, Paper 2017/620, 2017. https://eprint.iacr.org/2017/620.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[FU22] Denis Firsov and Dominique Unruh. Zero-knowledge in easycrypt. Cryptology ePrint Archive, Paper 2022/926, 2022. https://eprint.iacr.org/2022/926.

[Gre93] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. https://eprint.iacr.org/2019/953.

[HASW23] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Foundations of data availability sampling. Cryptology ePrint Archive, Paper 2023/1079, 2023.

[Huc] Fabian Huch. Archive of formal proofs.

[KE23] Katharina Kreuzer and Manuel Eberl. Elimination of repeated factors algorithm. *Archive of Formal Proofs*, November 2023. https://isa-afp.org/entries/Elimination_Of_Repeated_Factors.html, Formal proof development.

[KST21] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. Cryptology ePrint Archive, Paper 2021/370, 2021. https://eprint.iacr.org/2021/370.

[KT23] Tohru Kohrita and Patrick Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. Cryptology ePrint Archive, Paper 2023/917, 2023.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[Lan02] Serge Lang. *Algebra*, volume 3. Springer New York, NY, 2002.

[Lee20] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. Cryptology ePrint Archive, Paper 2020/1274, 2020.

[Loc17] Andreas Lochbihler. Crypthol. *Archive of Formal Proofs*, May 2017. https://isa-afp.org/entries/CryptHOL.html, Formal proof development.

[LS18] Andreas Lochbihler and S. Reza Sefidgar. A tutorial introduction to CryptHOL. Cryptology ePrint Archive, Paper 2018/941, 2018.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updateable structured reference strings. Cryptology ePrint Archive, Paper 2019/099, 2019. https://eprint.iacr.org/2019/099.

[MvOV96] Alfred John Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[Now07] David Nowak. A framework for game-based security proofs. Cryptology ePrint Archive, Paper 2007/199, 2007. https://eprint.iacr.org/2007/199.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[PN] Lawrence C. Paulson and Tobias Nipkow. Isabelle.

[Scr] Scroll. Scroll docs. https://docs.scroll.io/en/learn/zero-knowledge/kzg-commitment-scheme/.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Paper 2004/332, 2004. https://eprint.iacr.org/2004/332.

[Str] Pierre-Yves Strub. Easycrypt.

[STW23] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with lasso. Cryptology ePrint Archive, Paper 2023/1216, 2023. https://eprint.iacr.org/2023/1216.

[Tha22] Justin Thaler. *Proofs, Arguments, and Zero-Knowledge.* Now Publishers Inc, 2022.

[TY16] René Thiemann and Akihisa Yamada. Polynomial interpolation. *Archive of Formal Proofs*, January 2016. https://isa-afp.org/entries/Polynomial_Interpolation.html, Formal proof development.

[Wen23] Markarius Wenzel. The isabelle/isar reference manual. https://isabelle.in.tum.de/dist/Isabelle2023/doc/isar-ref.pdf, 2023.

[WTas$^+$17] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. Cryptology ePrint Archive, Paper 2017/1132, 2017. https://eprint.iacr.org/2017/1132.

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part IV*, page 299–328, Berlin, Heidelberg, 2022. Springer-Verlag.

[ZCF23] Hadas Zeilberger, Binyi Chen, and Ben Fisch. BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. Cryptology ePrint Archive, Paper 2023/1705, 2023.

[ZKS] ZKSync. Zksync era documentation. https://matter-labs.github.io/zksync-era/core/latest/guides/advanced/13_zk_intuition.html#intuition-guide-to-zk-in-zkevm.

[ZXZS19] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. Cryptology ePrint Archive, Paper 2019/1482, 2019. https://eprint.iacr.org/2019/1482.