SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Formalizing the Kate-Zaverucha-Goldberg Polynomial Commitment Scheme

Tobias Rothmann

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Formalizing the Kate-Zaverucha-Goldberg Polynomial Commitment Scheme

Eine Formalisierung des Kate-Zaverucha-Goldberg Polynomiellen Commitment Verfahrens

Author:	Tobias Rothmann
Supervisor:	Prof. Tobias Nipkow
Advisor:	Katharina Kreuzer
Submission Date:	April 15, 2024

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, April 15, 2024 Tobias Rothmann

Acknowledgments

Firstly I would like to thank my advisor Katharina Kreuzer, for initially accepting my proposal for this project in the practical course and going all the way to the end of this thesis with me. She introduced me to cryptographic proofs and specifically game-based proofs, which now form a core part of this work. I also want to thank her for her quick responses and the time spent with me thinking about the caveats of proofs I did not fully understand.

Additionally, I also want to thank Prof. Ian Goldberg, author of the KZG, for his swift responses and intuitive explanations.

Lastly, I want to thank Prof. Tobias Nipkow for allowing me to continue the project from a practical course in this thesis.

Abstract

We formalize security properties for the polynomial commitment scheme (PCS) by Kate, Zaverucha, and Goldberg (KZG) in the interactive theorem prover Isabelle/HOL. A PCS is a cryptographic primitive that is commonly used in modern cryptography, namely in *succinct non-interactive arguments of knowledge* (SNARKs). Building on our work in the practical course, where we formalized *correctness* and the security property *polynomial binding*, we complete the common security properties for polynomial commitment schemes. We formally prove *hiding*, *evaluation binding*, and, additionally, the property of *knowledge soundness*, which is commonly required for SNARK constructions. Furthermore, we extend our formalization to a batched version of the KZG for which we prove *evaluation binding* and *knowledge soundness* formally and outline an improved *hiding* proof. We provide our proofs in a game-based manner instead of the original reduction-style argument to strengthen trust in their security. This is the first formalization of any polynomial commitment scheme, to our knowledge.

Contents

Acknowledgments **iii**

Abstract **v**

1. Introduction **1**

2. Prelimiaries **5**
2.1. Mathematical Prelimiaries 5
2.2. Cryptographic Prelimiaries 5
2.3. Isabelle/HOL 10

3. KZG Definition **13**

4. Formalized Hardness Assumptions **17**

5. Formalized KZG Security **19**
5.1. Evaluation Binding 19
5.2. Hiding 21
5.3. Knowledge Soundness 27

6. Batch Version Definition **31**

7. Formalized Batch Version Security **35**
7.1. Evaluation Binding 35
7.2. Hiding 39
7.3. Knowledge Soundness 39

8. Conclusion **43**

A. Game-based Proofs **45**
A.1. Evaluation Binding Proof 45
A.2. Hiding Proof 45
A.3. Knowledge Soundness Proof 45

B. Additional Formalizations **47**
B.1. Extensions for SPMF 47
B.2. Extensions for CryptHOL 47

Bibliography **49**

1. Introduction

In the past 15 to 20 years the field of cryptography has shifted its focus from messages (e.g. hiding messages, encrypting messages etc.) to computation (e.g. mulit-party computation, computation over encrypted data and proving computation).

The first efficient *multi-party computation* (MPC) protocol was proposed in 2011 by Damgard et al.[Dam+11]. The solution to computation over encrypted data, which is *fully homomorphic encryption* (FHE), was proposed by Gentry in 2009 [Gen09] and followed by many FHE protocols [Che+17; Chi+18; BGV11; FV12]. Finally, Groth proposed the first efficient construction for succinctly proving computation in 2016: an efficient *succinct non-interactive argument of knowledge* (SNARK). This started a wave of research on SNARKs [Tha22; Gro16; Bün+18; Chi+19; Mal+19; GWC19; Bon+20; KST21; BC23].

This work is focused on a cryptographic primitive that is frequently used in these new cryptographic protocols, specifically SNARKs[GWC19; Mal+19; Chi+19; Bon+20] and MPC [Bha+23; Bau+23; RB89; KZG10], namely the polynomial commitment scheme (PCS) constructed by Kate, Zaverucha and Goldberg (KZG)[KZG10].

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 unique) 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 any polynomial commitment scheme. With this work, we set out to change this and give the first formalization of a polynomial commitment scheme, specifically focusing on the security proofs, thus paving the way for formalized security in modern cryptography, specifically SNARKs.

However, we want to motivate this work not purely with academic reasons, but want to highlight why it is absolutely necessary by examples of applications. One such example is the Ethereum blockchain, which recently (on the 13th of March, 2024) upgraded its protocol to support EIP-4844, which uses the KZG to store commitments to data blobs [But+22]. As the blockchain with the largest market capitalization, after Bitcoin, of >440 billion USD [Coi], as of writing this, a security flaw in the KZG would probably result in losses of billions. Besides Ethereum, the KZG is heavily used in other blockchains (e.g. Aztec[Ltdb] and Mina[Fou]), blockchain applications and infrastructure, such as SNARK-based rollups (e.g. Scroll[Ltda], zkSync[Labb], and Taiko[Labc]), data-availability networks (e.g. Celestia[Laba]) and co-processors (e.g.

Axiom[Tec]). Therefore, any flaw in the security of the KZG would have a detrimental practical impact.

With this work, we aim to not only lay the foundation for formalized security of modern cryptographic protocols (like SNARKs) but also to strengthen the trust in the security of the KZG and thus in systems and applications that are built on it.

Related Work

There are multiple polynomial commitment schemes besides the KZG. The most notable advantage of the KZG over all of them is, firstly, the KZG is explicitly mentioned in many SNARKs [GWC19; Mal+19; Chi+19; Bon+20; KST21] and, secondly, the KZG is the most efficient one wrt. to group operations [Tha22; BS23; KZG10]. Furthermore, the KZG is the first construction of a polynomial commitment scheme. Nevertheless, we discuss the most relevant PCSs. We touch on 3 groups: pairing-based, hash-based or inner-product-argument-based (IPA-based). To not go beyond the scope we focus on the most prevalent construction for each group. We provide figure 1 as an overview.

FRI [Ben+18a] is a hash-based PCS, that is essentially a Merkle-tree vector commitment to $t+1$ points for a degree t polynomial. Thus committing to a polynomial is not constant in size of the polynomial, but linear, contrasting to the KZG. FRI is used for transparent SNARKs, called scalable transparent non-interactive arguments of knowledge (STARKs) [Ben+18b]. A SNARKs is considered transparent if no trusted setup is needed (i.e. a STARK) [Ben+18b]. Note, that the setup required by the KZG is trusted as it generates the secret key, which is a trapdoor-information for the scheme.

Bulletproofs [Bün+18] is the most relevant IPA commitment scheme. Bulletproofs uses Pedersen commitments[Ped92] together with an IPA to construct a PCS. We do not go into details of the construction, specifically IPAs as this would go beyond the scope. Bulletproofs is transparent, however not succinct, as the verification of a point is linear in the size of the polynomial, as opposed to the KZG for which the verification time is constant [Bün+18; KZG10]. Furthermore, the witness size for Bulletproofs is logarithmic in the size of the polynomial, whereas it is constant in the KZG [Bün+18; KZG10]. Hence the KZG is the more efficient PCS.

Dory[Bün+21] is, as well as the KZG, a pairing-based PCS. Dory improves the time-complexity of opening points of a polynomial from linear in the degree of the polynomial for the KZG to logarithmic for Dory [Bün+21]. However, this improvement comes at the cost of the witness size, which is constant for the KZG, but logarithmic for Dory [Bün+21].

Note, that there are many more constructions for PCSs, e.g. DARK[BFS20], Virgo [Zha+19] and Hyrax[Wah+17]. We point the reader to [Bün+21] for an overview including these and more polynomial commitment schemes.

We formalize our proofs for the KZG 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 [BGZ09] in Coq[Coq], EasyCrypt[Str], CryptoVerify[Cad] and cryptoline [cry]. Furthermore, the interactive theorem prover Lean[Mou+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.

Butler et al. use CryptHOL to formalize a general framework for commitment schemes from Σ-protocols in [But+19]. Though their framework provides a good orientation for the proofs, it does not sufficiently capture the complexity of a polynomial commitment scheme.

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

With our work, we want to lay the foundation to formalize modern, more complex, zk-SNARKs (that rely on polynomial commitment schemes), prominent examples that explicitly mention the KZG include Plonk [GWC19], Halo[Bon+20], Sonic [Mal+19], Marlin [Chi+19], and Nova [KST21].

Bosshard, Bootle and Sprenger formally verify the sum-check protocol [BBS24], which is another cryptographic primitive that is used in zk-SNARKs with fast provers, such as Lasso and Jolt [STW23; AST23]. Though sum-checks are not directly related to SNARKs that use polynomial commitment schemes, their formalization, similarly to our formalization, forms a step towards formally verified complex SNARK proving systems and thus is worth acknowledging.

Contributions

We build on our work in the practical course, where we showed correctness and *polynomial binding* for the KZG, and complete the formalization of the common security properties for commitment schemes. We show *hiding* and *binding*, where binding is split up into the two properties *polynomial-binding* (showed in the practical course) and *evaluation-binding*. Additionally, we also formalize the security property *knowledge-soundness*, which is required for some SNARK constructions [Mal+19; GWC19]. Hence, we formalize these three security properties for the KZG:

- **evaluation binding:** No efficient adversary can compute a commitment, witnesses and two values, $\phi(x)$ and $\phi(x)'$, that are accepted by the verifier for an arbitrary but equal point i, except for negligible probability.
- **hiding:** No efficient adversary can compute a degree t polynomial from a commitment to that polynomial and t additional evaluations of the polynomial, except for negligible probability.

- **knowledge soundness:** Intuitively this property states, that an efficient adversary must know a polynomial ϕ and compute the commitment C for ϕ honestly, to reveal points, except for negligible probability.

Additionally, we formalize the **batched version** of the KZG[KZG10]. The batched version is an extension of the KZG for two more functions, which allow to evaluate a degree t polynomial at up to t points with one witness and one pairing check. We formalize the correctness of the batch version as well as the security properties of *evaluation binding* and *knowledge soundness*. Furthermore, we outline a game-based proof for *hiding*, which we could not formalize due to time constraints. Note, that the property of polynomial binding is not changed by adding the two functions for the batch version and thus is already shown for the batched version.

The original paper proofs are outlined in a reduction proof style, which is known to be error-prone [BR04]. To enhance the security expression of the proofs we transform them into game-based proofs, which are particularly rigorous [Sho04; BR04], before we formalize them. We use CryptHOL[Loc17], a framework specifically for game-based proofs in Isabelle, for our formalization. Additionally, we extend the theories *SPMF* and *CryptHOL* with some lemmas and definitions helpful for our formalization.

2. Prelimiaries

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 Prelimiaries

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 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 '$\cdot$' for addition and multiplication. We let a,b, and c denote finite field elements if not explicitly stated otherwise.

Definition 2.1.1 (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 **g** for a randomly chosen but fixed generator of a respective cyclic group.

Definition 2.1.2 (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:

- **Bilineraity:** $\forall g, h \in \mathbb{G}.\ \forall a, b \in \mathbb{F}_p.\ e(g^a, h^b) = e(g, h)^{ab}$
- **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 Prelimiaries

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 negligable and a poly-bounded function from which we will define our adversarial model, against which we will prove security in this paper.

Definition 2.2.1. Let $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}$ be a function. We call f negligable 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 2.2.2. 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 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) 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 2.2.3 (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].

Besides this definition, we will use a stronger definition of adversaries, namely that of the adversary in the Algebraic Group Model (AGM) [FKL17].

Definition 2.2.4 (AGM Adversary)**.** Let $\mathbb{F}_p$ be a finite field of prime order p and $\mathbb{G}$ a cyclic group of prime order p. An adversary in the AGM is an adversary as in 2.2.3, that furthermore outputs a vector $\vec{v}_e \in \mathbb{F}_p^t$ for every element e from $\mathbb{G}$ in its output, such that $e = \prod_1^t g_i^{v_{e_i}}$, where $g \in \mathbb{G}^t$ is the vector of all elements of $\mathbb{G}$ that the adversary has seen so far [FKL17].

The efficiency definition is analogue to 2.2.3

Now that we have defined adversary models, we define games.

Definition 2.2.5 (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].

Notatioinwise, 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":

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

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 game-based proofs, the method which we will use to formally prove security.

Definition 2.2.6 (game-based proofs)**.** Game-based 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 2.2.7 (hardness)**.** Given a computational problem P, we say P is hard if and only if no efficient adversary exists, that solves P with non-negligible probability [MOV96].

Definition 2.2.8 (hardness assumptions)**.** Hardness assumptions are computational problems that are generally believed to be hard [BS23; MOV96].

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

Formally we define the DL game:

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

Definition 2.2.10 (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:

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

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 2.2.11 (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:

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

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

- **KeyGen** generates a key ck for the commiter and a key vk for the verifier.
- **Commit** takes the commiter key ck, a message m and computes a commitment C for m and a opening value ov.
- **Verify** takes the verifier key vk, a commitment C, an opening value ov, a message m and decides whether C is a valid commitment to m using vk and ov.

[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 commiter uses their keys ck to commit to a arbitrary message m, invoking $\text{Commit}(ck, m)$ from which they obtain a commitment C to m and an opening value ov for the commitment. The commiter stores ov 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 opening value ov and the message m to the verifier.
3. The verifier invokes Verify on the values it received from the committer (C, m and ov) 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 2.2.13 (Polynomial Commitment Scheme (PCS))**.** A Polynomial Commitment Scheme is a Commitment Scheme as defined in 2.2.12, 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. partly) opening a commitment to a polynomial:

- **CreateWitness** takes the commiter key ck, a polynomial ϕ, a value i and computes 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.

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.

We formally define four security properties for a PCS:

Definition 2.2.14 (Polynomial Binding)**.** We say a PCS is polynomial binding if and only if the probability of any efficient adversary finding a commitment value C, opening values ov and ov' and polynomials ϕ and ϕ', such that:

$$\Pr\left[\text{Verify}(C, ov, \phi) \wedge \text{Verify}(C, ov', \phi')\right] = \epsilon$$

[KZG10]

Definition 2.2.15 (Evaluation Binding)**.** We say a PCS is evaluation binding if and only if the probability of any efficient adversary finding a commitment value C, two witnesses w and w' and two evaluations $\phi(i)$ and $\phi(i)'$ for any i, such that:

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

[KZG10]

Definition 2.2.16 (Knowledge Soundness (AGM))**.** We say a PCS is knowledge sound in the AGM if and only if there exists an efficient extractor algorithm E such that for every efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in the AGM the probability of winning the following game is negligible:

$$\left(\begin{array}{rl} (ck, vk) & \leftarrow \text{KeyGen} \\ (C, \vec{v}_C, \sigma) & \leftarrow \mathcal{A}_1(ck) \\ p & \leftarrow E(C, \vec{v}) \\ (i, \phi(i), \omega, \vec{v}_\omega) & \leftarrow \mathcal{A}_2(\sigma) \\ & : \phi(i) \neq p(i) \wedge \\ & \quad \text{VerifyEval}(vk, C, (i, \phi(i)), w) \end{array}\right)$$

[GWC19]

Definition 2.2.17 (hiding)**.** We say a PCS is hiding if and only if the probability of any efficient adversary finding an unknown point of the polynomial ϕ from the commitment $C := \text{Commit}(ck, \phi)$ and $\deg(\phi) - 1$ points with witness $(i, \phi(i), \text{CreateWitness}(ck, \phi, i))$ is negligible. [KZG10]

2.3. Isabelle/HOL

Now that we have covered the theoretical preliminaries, we introduce Isabelle/HOL[Isa], the interactive theorem prover we use to formalize our proofs. Note that we will use the abbreviation *Isabelle* to refer to *Isabelle/HOL* from now on.

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)*[AFP]. We will use the finite field definition as well as the factorization algorithm for polynomials over finite fields from the Berlekamp-Zassenhaus[Div+16] AFP entry. Furthermore, we use the Lagrange Polynomial Interpolation algorithm from the AFP entry *Polynomial Interpolation*[TY16]. 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 sub-probability 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 2.3.1.

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

The *do {...}* 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 ← 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 2.3.2.

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 2.3.2.

```
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. KZG Definition

In this chapter, we give the concrete construction, as well as an intuition of the KZG. Furthermore, we outline how the KZG has been formalized, note, however, that the definitions were formalized in a practical course and are not part of this thesis.

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 point α.

The commitment C is the evaluation at the unknown random point, 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 create a witness for a point to reveal, 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. We use $\psi(\alpha)$ as the witness ω, furthermore 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 2.2.13. In this section, we outline the PCS-constructing functions based on the original paper [KZG10]:

- **KeyGen(κ, 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 2.1.2, 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 opening value for the KZG is simply the polynomial $\phi(x)$, which is why we omit to return it as well.

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

  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]. Verify returns 1 if $C = \mathbf{g}^{\phi(\alpha)}$, otherwise 0.

- **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.

Formalization

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 3, we omit the details as they are not part of this thesis and are irrelevant to the proofs.

4. Formalized Hardness Assumptions

Before we continue with the formalization of the KZG's security in the next chapter, we formalize the hardness assumptions defined following 2.2.8 instantiated for the KZG in this chapter. We define them as games, such that we can target them in game-based proofs against an adversary.

Discrete Logarithm

We formalize the DL game according to 2.2.9 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 2.2.10 as follows:

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

t-Strong Bilinear Diffie Hellmann

We formalize the t-BSDH game according to 2.2.11 as follows:

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

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.

5. Formalized KZG Security

In this chapter, 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 game-based proof and outline the latter. We will not give concrete profs in Isabelle syntax as this would go beyond the scope due to a lot of boilerplate Isabelle-specific abbreviations and functions.

Each section covers one security property. Firstly, we describe the paper proof, followed by the *formalization* subsection, which outlines the target of our formalization, namely the security game, the reduction algorithm and if needed additional intermediary games. Lastly, we outline the game-based proof in the *games-based transformation* subsection.

Note, we skip the property of *polynomial-binding* as it was shown as part of a practical course and thus is not part of this thesis.

5.1. Evaluation Binding

We formalize the property *evaluation binding* according to 2.2.13. Formally we define *evaluation binding* as the following game against an efficient adversary $\mathcal{A}$:

$$\left(\begin{array}{r} PK \leftarrow \text{key\_gen}, \\ (C, i, \phi_i, \omega_i, \phi_i', \omega_i') \leftarrow \mathcal{A}\ PK, \\ \_ :: \text{unit} \leftarrow \text{assert\_spmf}(\phi_i \neq \phi_i' \\ \wedge\ \text{valid\_msg}\ \phi_i\ \omega_i \wedge \text{valid\_msg}\ \phi_i'\ \omega_i'), \\ b = \text{VerifyEval}\ PK\ C\ i\ \phi_i\ \omega_i, \\ b' = \text{VerifyEval}\ PK\ C\ i\ \phi_i'\ \omega_i', \\ :\ b \wedge b' \end{array}\right)$$

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:

$$(\frac{\omega_i}{\omega_i'})^{\frac{1}{\phi(i)' - \phi(i)}}$$

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.

Game-based Proof

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

A look at the *evaluation binding* and *t-SDH* games reveals that *key_gen*'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 Appendix A for the fully outlined game-based proof or the Isabelle theory *KZG_eval_bind* for the formal proof.

Formalization

We formally define the evaluation binding game in CryptHOL as follows:

```
TRY do {
    PK ← key_gen;
    (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 *key_gen* function of the KZG. The Adversary $\mathcal{A}$, given PK, outputs values to break evaluation binding, namely a commitment value C and two witnesses, w_i and w_i', and 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, where correct means 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 $(\frac{\omega_i}{\omega_i'})^{\frac{1}{\phi(i)'-\phi(i)}}$.

5.2. Hiding

We formalize the *hiding* property as given in the original paper [KZG10] that slightly differs from the standard definition as given in 2.2.17. The *hiding* property of the KZG does not unconditionally fulfill the *hiding* property as defined in 2.2.17 but requires

additionally the polynomial to be uniformly randomly chosen to hold [KZG10]. Formally we define *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:

$$\left(\begin{array}{r} \phi \xleftarrow{\$} \{\phi.\ \text{degree}(\phi) \leq t\}, \\ PK \leftarrow \text{key\_gen}, \\ C = \text{Commit}(PK, \phi), \\ wtnss = \text{map}\ (\text{CreateWitness}(PK, \phi, i))\ I, \\ \phi' \leftarrow \mathcal{A}(PK, C, wtnss), \\ :\ \phi = \phi' \end{array}\right)$$

Original Paper Proof

The paper argues that given an Adversary $\mathcal{A}$, that can break the 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 construction 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:

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

Firstly, $\mathcal{B}$ generates the public key PK using the function *Setup*, which additionally exposes the secret key α, as opposed to *key_gen*, which is wrapped around *Setup* as a filter for the public key. Secondly, $\mathcal{B}$ samples t-random points pts, which are used to create the t-witnesses $wtnss$ 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 tuple $wtnss$ 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.

Game-based 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 hiding: "hiding_advantage 𝒜 ≤ t_SDH.advantage (reduction 𝒜)"
```

Firstly, we outline why the reduction algorithm $\mathcal{B}$ cannot trivially be transformed into a reduction adversary for a game-based proof. Note that while $\mathcal{B}$ samples the point list uniform random and thus creates witnesses at uniform random positions, the 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 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 2.2.6, 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 points 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 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, 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 A.2 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:

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

Now that we have discussed the changes to the reduction algorithm as outlined in 5.2, we tend to the proof. We outline the main ideas behind the proof and refer the reader to appendix A.2 for a detailed proof and the Isabelle theory *KZG_hiding* for the formalized proof.

We start with the 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:

$$\left(\phi \xleftarrow{\$} \{\phi.\ \text{degree}(\phi) \leq t\},\ \right) = \left(\begin{array}{r} evals \xleftarrow{\$} \mathbb{Z}_p^t, \\ grp\_evals = \text{map}\ (\lambda\ i.\ \mathbf{g}^i)\ evals, \\ \phi = \text{interpolate}(\text{zip}(I, grp\_evals)), \end{array}\right)$$

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

$$\begin{aligned} &\left(\begin{array}{r} evals \xleftarrow{\$} \mathbb{Z}_p^t, \\ grp\_evals = \text{map}\ (\lambda\ i.\ \mathbf{g}^i)\ evals, \end{array}\right) \\ = &\left(\begin{array}{r} a \xleftarrow{\$} \mathbb{Z}_p, \\ g^a = \mathbf{g}^a, \\ evals \leftarrow \text{sample\_uniform}\ \mathbb{Z}_p^{t-1}, \\ grp\_evals = g^a\#\ \text{map}\ (\lambda\ i.\ \mathbf{g}^i)\ evals, \end{array}\right) \end{aligned}$$

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 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). Since $\phi' = \phi$ implies $\phi'(i) = \phi(i)$, we can use over-estimation to conclude that the probability of solving the hiding game's check is greater or equal

to solving the DL-game's check. Semantically we are over-estimating the hiding game with the DL game:

$$\left(: \phi' = \phi \quad \right) \leq \quad \left(: \phi'(i) = \phi(i) \quad \right)$$

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 5.2 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 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 $(i\phi(i)) \in pts$. Note, pts is I zipped with some randomly chosen evaluations. We state the full emulation here:

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

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

We formally define the hiding game in CryptHOL as follows:

```
TRY do {
    φ ← sample_uniform_poly t;
    PK ← key_gen;
    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 *key_gen*. 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 a polynomial ϕ'. The adversary wins if and only if the guessed polynomial ϕ' equals the chosen polynomial ϕ.

Additionally, we formalize the reduction algorithm. The prior introduced reduction algorithm $\mathcal{B}$ cannot be trivially turned into a reduction adversary that works for a game-based proof (see the next section 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 will ease the proof outlined in the next section as the probability of 0 being in the arbitrary list I can be neglected.

Formally, we define the reduction adversary as follows:

```
fun reduction 𝒜 PK = do {
    let i = pick_not_from 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 $(i, a)\#(\text{zip } I \; eval\_values)$. 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 5.2 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.

5.3. Knowledge Soundness

We formalize the property *knowledge soundness* according to 2.2.13 in the AGM. 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:

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

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[Bon+20]). 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 novel proof that exploits the 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 binding property is already broken, because VerifyEval verifies for two different evaluations at the same point of a polynomial.

Game-based proof

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) in a game-based proof:

```
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)"
```

Based on the idea from 5.3, 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:

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

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 5.3). 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 appendix A.3.

Formalization

We formalize the knowledge soundness game in CryptHOL as follows:

```
TRY do {
    PK ← key_gen;
    (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 (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 *key_gen*. 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 $\text{VerifyEval}(PK, C, i, \phi\_i, w\_i)$ hold.

We formalize the reduction adversary to the binding game as defined in 5.3:

```
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 5.3 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. Batch Version Definition

In this chapter, we outline the batched version of the KZG, as defined in [KZG10], and show how we formalized it. 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[Mal+19] 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 3 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, similar to the commit, as $\mathbf{g}^{\psi(\alpha)} = \prod_0^{deg(\psi)} (\mathbf{g}^j)^{\psi_j}$ [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.

Intuitively, *CreateWitnessBatch* uses the same technique as *CreateWitness*, but divides ϕ for multiple points, i.e. $\prod_{i \in B}(x-i)$ instead of $(x-i)$, and 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

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 ψ_B :: "'e eval_position set ⇒ 'e polynomial ⇒ 'e polynomial" where
"ψ_B 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 3 (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 alpha and the equation outlined in the definition of VerifyEvalBatch can be checked on the, to the commiter and verifier unknown, secret key α.

7. Formalized Batch Version Security

In this chapter, 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, 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, each section covers one security property. Firstly, we describe the paper proof, followed by the *formalization* subsection, which outlines the target of our formalization, namely the security game, the reduction algorithm and if needed additional intermediary games. Lastly, we outline the game-based proof in the *games-based transformation* subsection.

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 (see chapter 5).

7.1. Evaluation Binding

We formalize the property *evaluation binding* slightly adapted from 2.2.13 (we introduce *VerifyEvalBatch*) as described in the original paper [KZG10]. Formally we define *evaluation binding* as the following game against an efficient adversary $\mathcal{A}$ according to [KZG10]:

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

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. 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 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 $r'(x) = \frac{r(x)-r(i)}{(x-i)}$. 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 $\langle B, r(x), \omega_B \rangle$, 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.

Game-based Proof

The transformation into a game-based proof is analog to 5.1:

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)"
```

A look at the *evaluation binding* and *t-BSDH* games reveals that *key_gen*'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) 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 Appendix A for the fully outlined game-based proof or the Isabelle theory *KZG_BatchOpening_bind* for the formal proof.

Formalization

We formally define the evaluation binding game in CryptHOL as follows:

```
TRY do {
    PK ← key_gen;
    (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 *key_gen* 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 from B and r_x are valid for C, a point $i \in B$, a claimed value ϕ_i that should be different from the value of r_x at 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 $i \in B$, 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 $\left(-i, \left(e\left(\mathbf{g}^{p'(\alpha)}, \omega_B\right) \oplus e\left(\frac{\mathbf{g}^{p'(\alpha)}}{\omega_i}, \mathbf{g}\right)\right)^{\frac{1}{\phi(i)-r(i)}}\right)$, 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 $r' = \mathbf{g}^{\frac{r_x - r_x(i)}{(x-i)}}$.

7.2. Hiding

Original Paper Proof

Game-based Proof

Formalization

7.3. Knowledge Soundness

We extend the knowledge soundness property as defined in 2.2.16 and proved for the KZG in 5.3 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:

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

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 5.3:

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.

Game-based Proof

The game-based transformation is analog to 5.3:

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) in a game-based proof:

```
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_gameknowledge_soundness_reduction* theorem holds:

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

Based on the idea from 7.3, 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:

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

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 5.3). The adversary $\mathcal{A}'$ provides a set of positions B, the claimed evaluations of ϕ to the positions captured in the polynomial $r(x)$ (i.e. $\forall i \in B.\ r(i) \stackrel{!}{=} \phi(i)$), a witness ω_B for $r(x)$ (i.e. such that $VerifyEvalBatch(PK, C, B, r(x), \omega_B)$ holds) and B, 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 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

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

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 game-based proof is effectively equational reasoning over the asserts. The complete game-based proof is to be found in Appendix B.

Formalization

We formalize the knowledge soundness game in CryptHOL as follows:

```
TRY do {
    PK ← key_gen;
    (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 *key_gen*. 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. $\forall i \in B.\ \phi(i) \stackrel{!}{=} r(i)$), 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 VerifyEvalBatch(PK, C, B, r_x, w_B) hold.

We formalize the reduction adversary to the binding game as defined in 5.3:

```
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 7.3 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.

8. Conclusion

We successfully formalized the polynomial-commitment-scheme-typical security properties (polynomial binding, evaluation binding, and hiding) for the KZG, as well as the additional property of knowledge soundness that is commonly required for SNARK constructions [Tha22; GWC19; Mal+19; Bon+20]. Furthermore, we formalized the proofs in a game-based format for additional rigourness and thus trust in the security of our proofs. This work is hence a strong guarantee for the security of the KZG, proving it a secure cryptographic primitive.

Future Work

There are three particular directions for future work: firstly, further development on the formalization of the KZG, secondly continuing from the KZG and formalizing more complex KZG- or polynomial-commitment-scheme-based cryptographic primitives, and thirdly, formalizing cryptographic primitives assumed by the KZG.

Further Development on the Formalization

Isabelle supports code generation for several languages, including Haskell, OCaml and SML [Haf09] and, recently added, Go [SH23]. Hence, it would be desirable to extract a formally verified implementation out of this formalization, thus erasing the potential risk of implementation errors, which is a common root for security bugs in cryptographic implementations (see the recent Nova incident [NBS23], the 00-Plonk incident [Ngu22] or the *Frozen Heart* incident [Milb]). Due to the time-constraint setting of this thesis, we were not able to execute the code generation, but will do this for sure once the thesis is submitted.

Furthermore, a batched version that opens multiple commitments, to respectively multiple polynomials, for one value as proposed in [GWC19] and [Mal+19] could be formalized. This would improve verifier time for SNARKs such as Plonk, Sonic, and Halo (that make use of this feature), should they be formalized and extracted via code generation, to obtain a formally verified SNARK.

Formalizing KZG-based Cryptography

As pointed out in the introduction, SNARKs are a cryptographic primitive that is heavily used to secure funds in blockchain applications, however, to our knowledge, no SNARKs that rely on PCS's have been formally verified yet, although they are used in practice

(see the introduction). This led to vulnerabilities being discovered only after they already have been implemented and used (see [Mila]). Now, that we have formalized a PCS, namely the KZG, with this work, we open the possibility of formalizing PCS-based SNARKs e.g. Plonk, Sonic, Marlin, Nova, and Halo. Together with code generation, this would present the opportunity for a fully formally verified SNARK.

Formalizing KZG primitives

The KZG relies on pairings, which we have abstracted in the preliminaries and assumed in the locale to prove that the KZG is secure. Pairings are a cryptographic primitive, that is typically constructed using pairing-friendly elliptic curves such as the Barreto-Nährig[BN06] or the Baretto-Lynn-Scott[BLS03] elliptic curves [BS23]. However, to our knowledge, none of these have been formalized yet. To obtain a fully formally verified KZG implementation from the code generation, a pairing-friendly elliptic curve needs to be formally verified.

A. Game-based Proofs

A.1. Evaluation Binding Proof

$$
\begin{aligned}
\text{eval\_bind\_game} = & \left(\begin{array}{r}
PK \leftarrow \text{key\_gen}, \\
(C, i, \phi_i, \omega_i, \phi_i', \omega_i') \leftarrow \mathcal{A}\ PK, \\
\_ :: \text{unit} \leftarrow \text{assert\_spmf}(\phi_i \neq \phi_i' \\
\wedge \text{valid\_msg}\ \phi_i\ \omega_i \wedge \text{valid\_msg}\ \phi_i'\ \omega_i'), \\
b = \text{VerifyEval}\ PK\ C\ i\ \phi_i\ \omega_i, \\
b' = \text{VerifyEval}\ PK\ C\ i\ \phi_i'\ \omega_i', \\
:\ b \wedge b'
\end{array}\right) \\
= & \left(\begin{array}{r}
PK \leftarrow \text{key\_gen}, \\
(C, i, \phi_i, \omega_i, \phi_i', \omega_i') \leftarrow \mathcal{A}\ PK, \\
\_ :: \text{unit} \leftarrow \text{assert\_spmf}(\phi_i \neq \phi_i' \\
\wedge \text{valid\_msg}\ \phi_i\ \omega_i \wedge \text{valid\_msg}\ \phi_i'\ \omega_i' \\
\wedge \text{VerifyEval}\ PK\ C\ i\ \phi_i\ \omega_i \\
\wedge \text{VerifyEval}\ PK\ C\ i\ \phi_i'\ \omega_i'), \\
: \mathit{True}
\end{array}\right) \\
\leq & \left(\begin{array}{l}
a \leftarrow \mathbb{F}_p, \\
b \leftarrow \mathcal{A}, \\
c = F(a, b) \\
\ :\ P(c)
\end{array}\right)
\end{aligned}
$$

A.2. Hiding Proof

PickDistinct Algorithm

A.3. Knowledge Soundness Proof

B. Additional Formalizations

B.1. Extensions for SPMF

B.2. Extensions for CryptHOL

Bibliography

[AFP] AFP. *Archive of Formal Proofs*. URL: https://www.isa-afp.org.

[AST23] A. Arun, S. Setty, and J. Thaler. *Jolt: SNARKs for Virtual Machines via Lookups*. Cryptology ePrint Archive, Paper 2023/1217. https://eprint.iacr.org/2023/1217. 2023.

[Bau+23] C. Baum, N. Melissaris, R. Rachuri, and P. Scholl. *Cheater Identification on a Budget: MPC with Identifiable Abort from Pairwise MACs*. Cryptology ePrint Archive, Paper 2023/1548. https://eprint.iacr.org/2023/1548. 2023.

[BBS24] A. G. Bosshard, J. Bootle, and C. Sprenger. *Formal Verification of the Sumcheck Protocol*. 2024. arXiv: 2402.06093 [cs.CR].

[BC23] B. Bünz and B. Chen. *ProtoStar: Generic Efficient Accumulation/Folding for Special Sound Protocols*. Cryptology ePrint Archive, Paper 2023/620. https://eprint.iacr.org/2023/620. 2023.

[Ben+18a] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. "Fast Reed-Solomon Interactive Oracle Proofs of Proximity." In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. Ed. by I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 14:1–14:17. DOI: 10.4230/LIPICS.ICALP.2018.14.

[Ben+18b] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. *Scalable, transparent, and post-quantum secure computational integrity*. Cryptology ePrint Archive, Paper 2018/046. https://eprint.iacr.org/2018/046. 2018.

[BFS20] B. Bünz, B. Fisch, and A. Szepieniec. "Transparent SNARKs from DARK Compilers." In: *Advances in Cryptology – EUROCRYPT 2020*. Ed. by A. Canteaut and Y. Ishai. Cham: Springer International Publishing, 2020, pp. 677–706. ISBN: 978-3-030-45721-1.

[BGV11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. *Fully Homomorphic Encryption without Bootstrapping*. Cryptology ePrint Archive, Paper 2011/277. https://eprint.iacr.org/2011/277. 2011.

[BGZ09] G. Barthe, B. Grégoire, and S. Zanella Béguelin. "Formal certification of code-based cryptographic proofs." In: *SIGPLAN Not.* 44.1 (Jan. 2009), pp. 90–101. ISSN: 0362-1340. DOI: 10.1145/1594834.1480894.

[Bha+23] R. Bhadauria, C. Hazay, M. Venkitasubramaniam, W. Wu, and Y. Zhang. "Private Polynomial Commitments and Applications to MPC." In: *Public-Key Cryptography – PKC 2023*. Ed. by A. Boldyreva and V. Kolesnikov. Cham: Springer Nature Switzerland, 2023, pp. 127–158. ISBN: 978-3-031-31371-4.

[BLS03] P. S. L. M. Barreto, B. Lynn, and M. Scott. "Constructing Elliptic Curves with Prescribed Embedding Degrees." In: *Security in Communication Networks*. Ed. by S. Cimato, G. Persiano, and C. Galdi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 257–267. ISBN: 978-3-540-36413-9.

[BLS17] D. A. Basin, A. Lochbihler, and S. R. Sefidgar. *CryptHOL: Game-based Proofs in Higher-order Logic*. Cryptology ePrint Archive, Paper 2017/753. https://eprint.iacr.org/2017/753. 2017.

[BM23] B. Bailey and A. Miller. *Formalizing Soundness Proofs of SNARKs*. Cryptology ePrint Archive, Paper 2023/656. https://eprint.iacr.org/2023/656. 2023.

[BN06] P. S. L. M. Barreto and M. Naehrig. "Pairing-Friendly Elliptic Curves of Prime Order." In: *Selected Areas in Cryptography*. Ed. by B. Preneel and S. Tavares. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 319–331. ISBN: 978-3-540-33109-4.

[Bon+20] D. Boneh, J. Drake, B. Fisch, and A. Gabizon. *Halo Infinite: Recursive zk-SNARKs from any Additive Polynomial Commitment Scheme*. Cryptology ePrint Archive, Paper 2020/1536. https://eprint.iacr.org/2020/1536. 2020.

[BR04] M. Bellare and P. Rogaway. *Code-Based Game-Playing Proofs and the Security of Triple Encryption*. Cryptology ePrint Archive, Paper 2004/331. https://eprint.iacr.org/2004/331. 2004.

[BS23] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. http://toc.cryptobook.us/book.pdf. 2023.

[Bün+18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More." In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[Bün+21] B. Bünz, M. Maller, P. Mishra, N. Tyagi, and P. Vesely. "Proofs for Inner Pairing Products and Applications." In: *Advances in Cryptology – ASIACRYPT 2021*. Ed. by M. Tibouchi and H. Wang. Cham: Springer International Publishing, 2021, pp. 65–97. ISBN: 978-3-030-92078-4.

[But+19] D. Butler, A. Lochbihler, D. Aspinall, and A. Gascon. *Formalising Σ-Protocols and Commitment Schemes using CryptHOL*. Cryptology ePrint Archive, Paper 2019/1185. https://eprint.iacr.org/2019/1185. 2019.

[But+22] V. Buterin, D. Feist, D. Loerakker, G. Kadianakis, M. Garnett, M. Taiwo, and A. Dietrichs. *EIP-4844: Shard Blob Transactions [DRAFT], Ethereum Improvement Proposals, no. 4844*. = https://eips.ethereum.org/EIPS/eip-4844, 2022.

[Cad] D. Cadé. *CryptoVerif*. URL: https://bblanche.gitlabpages.inria.fr/CryptoVerif/.

[Che+17] J. H. Cheon, A. Kim, M. Kim, and Y. Song. "Homomorphic Encryption for Arithmetic of Approximate Numbers." In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by T. Takagi and T. Peyrin. Cham: Springer International Publishing, 2017, pp. 409–437. ISBN: 978-3-319-70694-8.

[Chi+18] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. *TFHE: Fast Fully Homomorphic Encryption over the Torus*. Cryptology ePrint Archive, Paper 2018/421. https://eprint.iacr.org/2018/421. 2018.

[Chi+19] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. Ward. *Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS*. Cryptology ePrint Archive, Paper 2019/1047. https://eprint.iacr.org/2019/1047. 2019.

[Coi] CoinMarketCap. *Mina*. URL: https://coinmarketcap.com (visited on 03/15/2024).

[Coq] Coq. *Coq*. URL: https://coq.inria.fr.

[cry] cryptoline. *cryptoline*. URL: https://github.com/fmlab-iis/cryptoline.

[Dam+11] I. Damgard, V. Pastro, N. P. Smart, and S. Zakarias. *Multiparty Computation from Somewhat Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2011/535. https://eprint.iacr.org/2011/535. 2011.

[Div+16] J. Divasón, S. J. C. Joosten, R. Thiemann, and A. Yamada. "The Factorization Algorithm of Berlekamp and Zassenhaus." In: *Archive of Formal Proofs* (Oct. 2016). https://isa-afp.org/entries/Berlekamp_Zassenhaus.html, Formal proof development. ISSN: 2150-914x.

[FKL17] G. Fuchsbauer, E. Kiltz, and J. Loss. *The Algebraic Group Model and its Applications*. Cryptology ePrint Archive, Paper 2017/620. https://eprint.iacr.org/2017/620. 2017.

[Fou] M. Foundation. *Mina*. URL: https://minaprotocol.com (visited on 04/03/2024).

[FU22] D. Firsov and D. Unruh. *Zero-Knowledge in EasyCrypt*. Cryptology ePrint Archive, Paper 2022/926. https://eprint.iacr.org/2022/926. 2022.

[FV12] J. Fan and F. Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2012/144. https://eprint.iacr.org/2012/144. 2012.

[Gen09] C. Gentry. "A fully homomorphic encryption scheme." AAI3382729. PhD thesis. Stanford, CA, USA, 2009. ISBN: 9781109444506.

[Gro16] J. Groth. *On the Size of Pairing-based Non-interactive Arguments*. Cryptology ePrint Archive, Paper 2016/260. https://eprint.iacr.org/2016/260. 2016.

[GWC19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Paper 2019/953. https://eprint.iacr.org/2019/953. 2019.

[Haf09] F. Haftmann. "Code generation from specifications in higher-order logic." PhD thesis. Technische Universität München, 2009.

[Isa] Isabelle. *Isabelle*. URL: https://isabelle.in.tum.de.

[KST21] A. Kothapalli, S. Setty, and I. Tzialla. *Nova: Recursive Zero-Knowledge Arguments from Folding Schemes*. Cryptology ePrint Archive, Paper 2021/370. https://eprint.iacr.org/2021/370. 2021.

[KZG10] A. Kate, G. M. Zaverucha, and I. 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*. Vol. 6477. Lecture Notes in Computer Science. Springer, 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11.

[Laba] C. Labs. *Celestia*. URL: https://celestia.org (visited on 04/03/2024).

[Labb] M. Labs. *ZKSync*. URL: https://zksync.io (visited on 04/03/2024).

[Labc] T. Labs. *Taiko*. URL: https://taiko.xyz (visited on 04/03/2024).

[Lan02] S. Lang. *Algebra*. Vol. 3. Springer New York, NY, 2002.

[Loc17] A. Lochbihler. "CryptHOL." In: *Archive of Formal Proofs* (May 2017). https://isa-afp.org/entries/CryptHOL.html, Formal proof development. ISSN: 2150-914x.

[LS18] A. Lochbihler and S. R. Sefidgar. *A tutorial introduction to CryptHOL*. Cryptology ePrint Archive, Paper 2018/941. https://eprint.iacr.org/2018/941. 2018.

[Ltda] S. Ltd. *Scroll*. URL: https://scroll.io (visited on 04/03/2024).

[Ltdb] S. H. Ltd. *Aztec*. URL: https://aztec.network (visited on 04/03/2024).

[Mal+19] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. *Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updateable Structured Reference Strings*. Cryptology ePrint Archive, Paper 2019/099. https://eprint.iacr.org/2019/099. 2019.

[Mila] J. Miller. *The Frozen Heart vulnerability in Bulletproofs*. https://blog.trailofbits.com/2022/04/15/the-frozen-heart-vulnerability-in-bulletproofs/. Accessed: 2024-04-02.

[Milb] J. Miller. *The Frozen Heart vulnerability in PlonK*. https://blog.trailofbits.com/2022/04/18/the-frozen-heart-vulnerability-in-plonk/. Accessed: 2024-04-02.

[Mou+15] L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. "The Lean Theorem Prover (System Description)." In: *Automated Deduction - CADE-25*. Ed. by A. P. Felty and A. Middeldorp. Cham: Springer International Publishing, 2015, pp. 378–388. ISBN: 978-3-319-21401-6.

[MOV96] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[NBS23] W. Nguyen, D. Boneh, and S. Setty. *Revisiting the Nova Proof System on a Cycle of Curves*. Cryptology ePrint Archive, Paper 2023/969. https://eprint.iacr.org/2023/969. 2023.

[Ngu22] Q. T. M. Nguyen. "00." In: *CoRR* abs/2201.00815 (2022). arXiv: 2201.00815.

[Now07] D. Nowak. *A Framework for Game-Based Security Proofs*. Cryptology ePrint Archive, Paper 2007/199. https://eprint.iacr.org/2007/199. 2007.

[Ped92] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing." In: *Advances in Cryptology — CRYPTO '91*. Ed. by J. Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140. ISBN: 978-3-540-46766-3.

[RB89] T. Rabin and M. Ben-Or. "Verifiable secret sharing and multiparty protocols with honest majority." In: *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*. STOC '89. Seattle, Washington, USA: Association for Computing Machinery, 1989, pp. 73–85. ISBN: 0897913078. DOI: 10.1145/73007.73014.

[SH23] T. Stübinger and L. Hupel. *Extending Isabelle/HOL's Code Generator with support for the Go programming language*. 2023. arXiv: 2310.02704 [cs.PL].

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

[Str] P.-Y. Strub. *EasyCrypt*. URL: https://github.com/EasyCrypt/easycrypt.

[STW23] S. Setty, J. Thaler, and R. Wahby. *Unlocking the lookup singularity with Lasso*. Cryptology ePrint Archive, Paper 2023/1216. https://eprint.iacr.org/2023/1216. 2023.

[Tec] I. Technologies. *Axiom*. URL: https://www.axiom.xyz (visited on 04/03/2024).

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

[TY16] R. Thiemann and A. Yamada. "Polynomial Interpolation." In: *Archive of Formal Proofs* (Jan. 2016). https://isa-afp.org/entries/Polynomial_Interpolation.html, Formal proof development. ISSN: 2150-914x.

[Wah+17] R. S. Wahby, I. Tzialla, abhi shelat, J. Thaler, and M. Walfish. *Doubly-efficient zkSNARKs without trusted setup*. Cryptology ePrint Archive, Paper 2017/1132. `https://eprint.iacr.org/2017/1132`. 2017.

[Wen23] M. Wenzel. *The Isabelle/Isar Reference Manual*. `https://isabelle.in.tum.de/dist/Isabelle2023/doc/isar-ref.pdf`. 2023.

[Zha+19] J. Zhang, T. Xie, Y. Zhang, and D. Song. *Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof*. Cryptology ePrint Archive, Paper 2019/1482. `https://eprint.iacr.org/2019/1482`. 2019.