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

Abstract

We formalize the polynomial commitment scheme by Kate, Zaverucha, and Goldberg (KZG) in the interactive theorem prover Isabelle/HOL, paving the way for formalizations of modern cryptography protocols, based on polynomial commitment schemes, such as SNARKs. This is the first formalization of any polynomial commitment scheme, to our knowledge. We formalize the commitment scheme typical security properties binding, for the KZG specifically *polynomial binding* and *evaluation binding*, and *hiding* oriented on the KZG paper, as well as the additional property of *knowledge soundness*, that is commonly required for SNARK constructions. Furthermore, we extend our formalization to a batched version of the KZG with all security properties, namely *polynomial binding*, *evaluation binding*, *hiding* and *knowledge soundness*. We provide our proofs in a game-based manner instead of the original reduction-style argument to further enhance trust in their security.

Contents

1. Introduction

In the past 15 to 20 years the field of cryptography has shifted from classical cryptography, which centered around messages (e.g. hiding messages, encrypting messages etc.), to modern cryptography, which is centered around computation. New cryptographic primitives such as (secure) multi-part computation (MPC), fully homomorphic encryption (FHE), which is computation over encrypted data, and succinct non-interactive arguments of knowledge (SNARKs), which is succinctly proving computation, have emerged. Polynomial commitment schemes (PCS) are a commonly used primitive in modern cryptography, specifically in SNARKs [Tha22; BS23], however, 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.

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[1], 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[2] and Mina[3]), blockchain applications and infrastructure, such as SNARK-based rollups (e.g. Scroll[4], zkSync[5], and Taiko[6]), data-availability networks (e.g. Celestia[7]) and co-processors (e.g. Axiom[8]). 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

[1] https://coinmarketcap.com Accessed: 15th of March, 2024
[2] https://aztec.network
[3] https://minaprotocol.com
[4] https://scroll.io
[5] https://zksync.io
[6] https://taiko.xyz
[7] https://celestia.org
[8] https://www.axiom.xyz

KZG's security, and thus in systems and applications, that are built on it.

1.1. Contributions

We formalize the commitment-scheme-typical security properties *hiding* and *binding*, where binding is split up into the two properties *polynomial-binding* and *evaluation-binding*. Additionally, we also formalize *knowledge-soundness*, which is required for SNARK construction. Hence, we formalize these four security properties (see the respective sections in the main part for exact definitions):

- **polynomial binding:** No efficient Adversary can compute a commitment that can be resolved to two separate polynomials, except for negligible probability.

  (Note, that this property was proven as part of a practical course and does not belong to the achievements of this thesis.)

- **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 polynomial of degree t+1, 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 the polynomial it's committing to to reveal points, except for negligible probability.

Additionally, we formalize the **batched version** of the KZG with all security properties outlined above.

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, a framework specifically for game-based proofs in Isabelle, for our formalization.

1.2. related work

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[9], EasyCrypt, [10], CryptoVerify[11] and cryptoline [12]. 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], however, their framework, though it provides good orientation for the proofs, 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], 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.

[9] https://coq.inria.fr
[10] https://github.com/EasyCrypt/easycrypt
[11] https://bblanche.gitlabpages.inria.fr/CryptoVerif/
[12] https://github.com/fmlab-iis/cryptoline

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 $\mathcal{G}$ be a group of prime order p. We call a group cyclic iff: $\exists g \in \mathcal{G}.\ \forall e \in \mathcal{G}.\ \exists n \in \mathbb{N}.\ e = g^n$, which is equivalent to $\mathcal{G} = \{1, g, g^2, ..., g^{p-1}\}$ [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 $\mathcal{G}$ and $\mathcal{H}$ be two groups of prime order p. A pairing is a function: $\mathcal{G} \times \mathcal{G} \to \mathcal{H}$, with the following two properties:

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

[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 definition

2.2.3, that furthermore outputs a vector $\vec{z} \in \mathbb{F}_p^t$ for every element e from $\mathbb{G}$ in its output, such that $e = \prod_1^t g_i^{z_i}$, 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 definition 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 and output a boolean value [BS23]. Formally we write games as a sequence of functions and Adversary calls [BS23].

Notatioinwise we write ' $\leftarrow$ ' 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":

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

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 (conveniently the KZG paper [KZG10] already defines them) and formally define according games.

From now on let '$\in_{\mathcal{R}}$' denote uniform sampling from the respective set.

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

Formally we define the DL game as:

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

Definition 2.2.10 (t-Strong Diffie Hellmann (t-SDH))**.** Let $\mathcal{G}$ be a cyclic group with generator $\mathbf{g}$. Let $t \in \mathbb{N}$ be fixed. For $a \in_{\mathcal{R}} \mathbb{F}_p$, holds for every Adversary $\mathcal{A}$:

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

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

Formally we define the t-SDH game as:

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

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 $\mathcal{G}$ and $\mathcal{H}$ be cyclic groups with generators $\mathbf{g}$ and $\mathbf{h}$. Let $t \in \mathbb{N}$ be fixed and $e : \mathcal{G} \times \mathcal{G} \to \mathcal{H}$ be a pairing function. For $a \in_{\mathcal{R}} \mathbb{F}_p$, holds for every Adversary $\mathcal{A}$:

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

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

Formally we define the t-SDH game as:

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

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 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:

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

[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:

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

[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:

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

[GWC19]

Definition 2.2.17 (hiding)**.** We say a PCS is hiding if and only if the probability of any efficient adversary finding an, to them, 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[1], 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)*[2]. 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]:

[1] https://isabelle.in.tum.de/
[2] https://www.isa-afp.org

2.3.1. 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.

3.1. 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.

3.2. Definition

The KZG is a polynomial commitment scheme as defined in definition 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.

3.3. 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 3.2, 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.

4.1. 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'.

4.2. 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 $\mathbf{g}^{\frac{1}{\alpha+c}}$.

4.3. 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 $\mathbf{g}\ \mathbf{g}^{\frac{1}{\alpha+c}} = (\text{e}\ \mathbf{g}\ \mathbf{g})^{\frac{1}{\alpha+c}}$.

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

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-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 here, 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.

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

5.1.2. game-based transformation

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 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. 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 follow 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.

5.2. hiding

Note that 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.

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 and interpolates a polynomial over t random group points and the DL instance for the commitment. 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:

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

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 and interpolated 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.

5.2.1. 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 $\phi \in \mathbb{Z}[X]^t$ of degree t (wlog) 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.

The reduction 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 formally define the reduction adversary as follows:

```
fun bind_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 $(i, a)\#(\text{zip } I \text{ } eval\_values)$. We omit further details on the interpolation as that would go beyond the scope, for the exact computation, we refer the reader to the function *interpolate_on* of the KZG_hiding theory in the formalization. Using the Commitment, valid witnesses can be created for the points in the list 'zip I *eval_values*', note:

$$(C \div \mathbf{g}^y)^{\frac{1}{\alpha - x}} = (\mathbf{g}^{\phi(\alpha)} \div \mathbf{g}^{\phi(x)})^{\frac{1}{\alpha - x}} = (\mathbf{g}^{\phi(\alpha) - \phi(x)})^{\frac{1}{\alpha - x}} = \mathbf{g}^{\frac{\phi(\alpha) - \phi(x)}{\alpha - x}}$$

and $\mathbf{g}^{\frac{\phi(\alpha)-\phi(x)}{\alpha-x}}$ is the result of *CreateWitness PK* ϕ x.

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.

5.2.2. game-based transformation

5.3. knowledge soundness

5.3.1. formalization

5.3.2. game-based transformation

6. Batch Version Definition

6.1. Formalization

7. Formalized Batch Version Security

7.1. evaluation binding

7.1.1. formalization

7.2. hiding

7.2.1. formalization

7.3. knowledge soundness

7.3.1. formalization

8. Conclusion

8.1. Future Work

Go Code generation -> SNARKs + Go-code generation.

A. Game-based Proofs

A.1. Evaluation Binding Proof

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

Bibliography

[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.

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

[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.

[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.

[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.

[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.

[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.

[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.

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

[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.

[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.

[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.

[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.

[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.

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

[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.

[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.

[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.

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