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

Contents

1 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.

1.1 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 1.1.1 (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 an arbitrary but fixed generator of a respective cyclic group.

Definition 1.1.2 (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.

1.2 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 1.2.1. 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 1.2.2. 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 1.2.3 (efficient Adversary)**.** An Adversary is a probabilistic algorithm, that takes a security parameter κ as its first argument. 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 1.2.4 (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

1.2.3, 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 1.2.3

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

Definition 1.2.5 (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":

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

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

Definition 1.2.6 (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 1.2.7 (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 1.2.8 (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 1.2.9 (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 1.2.10 (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. Nevertheless, we define the property formally for completeness.

Definition 1.2.11 (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 1.2.12 (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 1.2.13 (Polynomial Commitment Scheme (PCS))**.** A Polynomial Commitment Scheme is a Commitment Scheme as defined in 1.2.12, 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 1.2.14 (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 1.2.15 (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 1.2.16 (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:

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

[GWC19]

Definition 1.2.17 (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]

1.3 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

1.3.1 CryptHOL

CryptHOL is a framework for game-based proofs in Isabelle [BLS17]. It supports games in a monadic notation over discrete sub-probability distributions. We use sub-probability mass functions (spmfs) in games to define the result of the game, the discrete sub-probability distribution (dspd), that is the sum of the spmfs applied to all possible elementary events. The notation is drawn from Haskell's do-notation, to give an example game in the correct syntax:

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

The *do {. . . }* block captures the monad. Using ←, which stands for the monadic binding operation, a (uniformly random) elementary event (i.e. an element of S) is bound to x. The result is a dspd, the spmf f applied to every element from S.

Additionally, since we are dealing with *sub*-probability mass functions, there exists an unassigned probability mass (e.g. if the spmf is undefined for some event). In CryptHOL, the unassigned probability mass stands for an error in the game and can be handled in a *TRY ELSE* block (i.e. TRY A ELSE B, captures semantically 'try to return A if A is an error, return B'). Introducing errors allows us to assert statements, specifically e.g. that the messages returned by the adversary are wellformed. To give an example in Isabelle's notation:

```
TRY do {
    x ← sample_uniform S;
    y ← 𝒜 x;
    _::unit ← assert_spmf(valid_msg y);
    return_spmf y
} ELSE (return_spmf 0)
```

As in the first game, an elementary event of S is bound to x. The Adversary $\mathcal{A}$, which itself is a spmf, applied to x, is bound to y (which is a dspd). However, this time, y is not an immediate result, instead, we use an assert to check that y, the result from the adversary, is valid. The assert_spmf results in an unassigned probability mass if and only if the statement passed to it is false, otherwise it will result in unit probability mass. If the result is an unassigned probability mass, this will, monad-typically, propagate through to the monad's result, which will trigger the ELSE branch, which in this example, results in the probability distribution with only 0 (i.e. $\Pr[i = 0] = 1$). If the result of the assert_spmf is a unit probability mass, the result will be neglected in the remaining part of the monad, in our example, this would mean that 'return_spmf y' would be the result of the game, which is the result of the Adversary applied to `x` (note for arbitrary $\mathcal{A}$ and `x` `x` $\leftarrow \mathcal{A}$`; return_spmf x` is equivalent to simply `A` in a monad).

Abbreviations

List of Figures

List of Tables

Bibliography

[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.

[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.

[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.

[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.

[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.

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

[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.

[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.