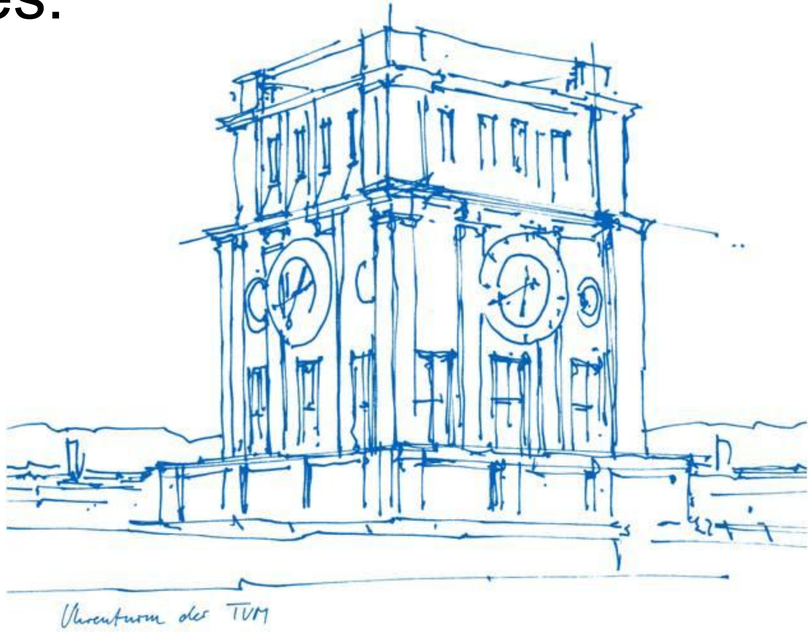
On the Formal Verification of Polynomial Commitment Schemes: the KZG and beyond

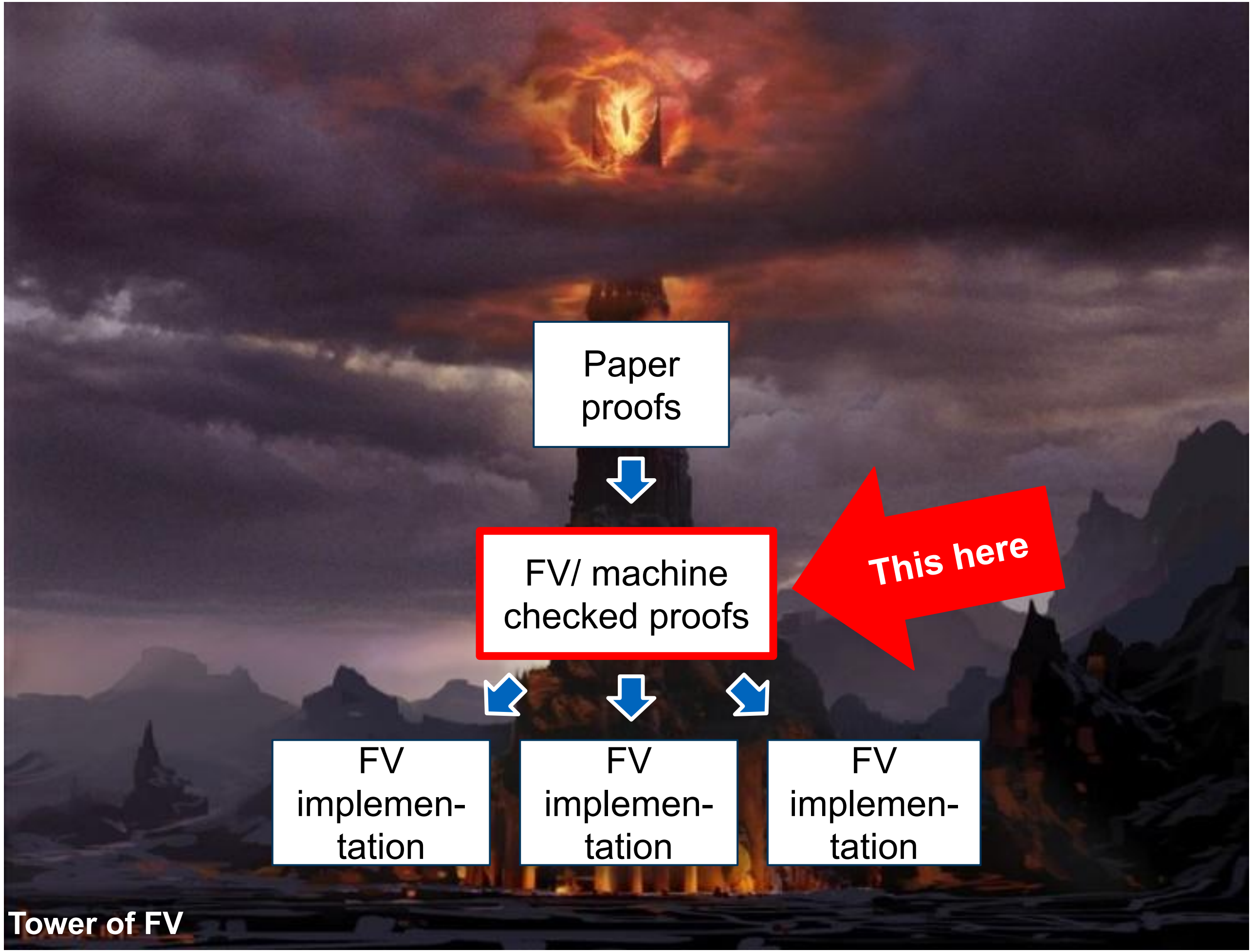
Tobias Rothmann

Technical University Munich

TUM School of Computation, Information and Technology

March 25th, 2025




Paper proofs
FV/ machine checked proofs
This here
FV implemen-tation
FV implemen-tation
FV implemen-tation
Tower of FV

The *Arete* of FV for Cryptography

aka “What is the full potential of FV tools for cryptography?”

aka “What do we want from FV tools for cryptography?”

- minimal assumptions/trusted parts for maximal security
- easily auditable (few minutes)
- easy to understand for non-FV experts

CryptHOL - the FV-tool of choice

aka “Cryptography in Isabelle”

- small verifier kernel (Standard ML) used for decades
- extensive verified compiler/extensions

- minimal prover assumptions for maximal security ✅
- easily auditable (minutes)
- easy to understand for non-FV experts

Sequences of Games (SoG) [Sho04]

aka “bind P[Game1] to P[Game2] via game-hops”

- Games as probability spaces
- Define Games G1,G2,…,Gn, such that P[Gi] is negligible close to P[Gi+1]
- G1 is attack game, Gn is hardness assumption or 0, coin toss i.e. ½ etc.

$$\text{Attack Game} = \Big(\; \dots \; \Big) \approx \Big(\; \dots \; \Big) \approx \cdots \approx \Big(\; \dots \; \Big) = \text{hardness assumption}$$

Sequences of Games (SoG) [Sho04]

aka “types of game-hops”

Game-hop as a Bridging Step:

$$\begin{pmatrix} a \xleftarrow{\$} \mathbb{Z} \\ a' \leftarrow Eve(\mathbf{g}^a) \\ : a = a' \end{pmatrix} = \begin{pmatrix} a \xleftarrow{\$} \mathbb{Z} \\ a' \leftarrow Eve(\mathbf{g}^a) \\ : a' = a \end{pmatrix}$$

Game-hop based on a Failure Event:

$$\begin{pmatrix} a \xleftarrow{\$} \mathbb{Z} \\ a' \leftarrow Eve(\mathbf{g}^a) \\ : a = a' \end{pmatrix} \approx \begin{pmatrix} a \xleftarrow{\$} \mathbb{Z} \\ a' \leftarrow Eve(\mathbf{g}^{\frac{1}{a}}) \\ : \frac{1}{a} = a' \end{pmatrix}$$

*generally any kind of game-hop, these are just optimized in Isabelle

Sequences of Games - in CryptHOL

aka “Games in Isabelle”

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

$$\left(a \xleftarrow{\$} \mathbb{Z}_p \quad a' \leftarrow Eve(\mathbf{g}^a) \quad : a = a' \right)$$

⇒ machine checked transitions through Isabelle proofs

Sequences of Games - in CryptHOL

aka “Games in Isabelle”

```
also have "... = TRY do {
  a :: nat ← sample_uniform q;
  x1 :: nat ← sample_uniform q;
  y1 ← sample_uniform q;
  ((m0, m1), σ) ← 𝒜1 (x1,y1);
  _ :: unit ← assert_spmf (valid_msg m0 ∧ valid_msg m1);
  d ← coin_spmf;
  let c = ((if d then m0 else m1) + a) mod q;
  b' ← 𝒜2 c σ;
  return_spmf (b' = d)} ELSE coin_spmf"
  by(simp add: samp_uni_plus_one_time_pad)
also have "... = TRY do {
  x1 :: nat ← sample_uniform q;
  y1 ← sample_uniform q;
  ((m0, m1), σ) ← 𝒜1 (x1,y1);
  _ :: unit ← assert_spmf (valid_msg m0 ∧ valid_msg m1);
  d ← coin_spmf;
  c ← map_spmf (λ a. ((if d then m0 else m1) + a) mod q) (sample_uniform q);
  b' ← 𝒜2 c σ;
  return_spmf (b' = d)} ELSE coin_spmf"
  by(simp add: o_def bind_map_spmf)
also have "... = TRY do {
  x1 :: nat ← sample_uniform q;
  y1 ← sample_uniform q;
  ((m0, m1), σ) ← 𝒜1 (x1,y1);
  _ :: unit ← assert_spmf (valid_msg m0 ∧ valid_msg m1);
  d ← coin_spmf;
  c ← sample_uniform q;
  b' :: bool ← 𝒜2 c σ;
  return_spmf (b' = d)} ELSE coin_spmf"
  by(simp add: samp_uni_plus_one_time_pad)
also have "... = TRY do {
  x1 :: nat ← sample_uniform q;
  y1 ← sample_uniform q;
  ((m0, m1), σ) ← 𝒜1 (x1,y1);
  _ :: unit ← assert_spmf (valid_msg m0 ∧ valid_msg m1);
  c :: nat ← sample_uniform q;
  guess :: bool ← 𝒜2 c σ;
  map_spmf((=) guess) coin_spmf} ELSE coin_spmf"
  by(simp add: map_spmf_conv_bind_spmf)
```

CryptHOL - the FV-tool of choice

aka “Cryptography in Isabelle”

- Isabelle: small verifier kernel (Standard ML) + verified compiler/extensions
- SoG style proofs → only need to audit attack game (G1) and hardness assumption game (Gn)
- games as monads → look like normal games, non-Expert friendly

- easily auditable (minutes) ✅
- easy to understand for non-FV experts ✅
- minimal prover assumptions for maximal security ✅

Formal Verification of Polynomial Commitment Schemes (in CryptHOL)

This Work

- formalize abstract PCS incl. Security Games in Isabelle
  - Polynomial Binding, Evaluation Binding, Hiding, Knowledge Soundness/Extractability, Correctness
- Instantiate two KZG versions
  - standard KZG
  - batch Eval KZG
  - prove acc. security properties
- demonstrate versatility of definitions by showing adapted “weak” hiding and batched eval
- WIP: formalize the AGM

Formalizing Polynomial Commitment Schemes

```
locale abstract_polynomial_commitment_scheme =
  fixes key_gen :: "('ck × 'vk) spmf" — ‹outputs the keys received by the two parties›
    and commit :: "'ck ⇒ 'r::zero poly  ⇒ ('commit × 'trapdoor) spmf"
      — ‹outputs the commitment as well as the secret, which might be used to derive witnesses,
         and the opening values sent by the committer in the reveal phase›
    and verify_poly :: "'vk ⇒ 'r poly ⇒ 'commit ⇒ 'trapdoor ⇒ bool"
      — ‹checks whether the polynomial corresponds to the commitment›
    and eval :: "'ck ⇒ 'trapdoor ⇒ 'r poly ⇒ 'argument ⇒ ('evaluation × 'witness)"
      — ‹outputs a point and a witness›
    and verify_eval :: "'vk ⇒ 'commit ⇒ 'argument ⇒ ('evaluation × 'witness) ⇒ bool"
      — ‹checks whether the point is on the polynomial corresponding to the commitment›
begin
```

Formalizing Polynomial Commitment Schemes

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

definition eval_bind_advantage :: "('ck, 'commit, 'argument, 'evaluation, 'witness)
  eval_bind_adversary ⇒ real"
  where "eval_bind_advantage 𝒜 ≡ spmf (eval_bind_game 𝒜) True"
```

The Two Instantiations

- standard KZG:
  - Completeness
  - Binding
  - Evaluation Binding
  - Knowledge Soundness
  - Weak Hiding

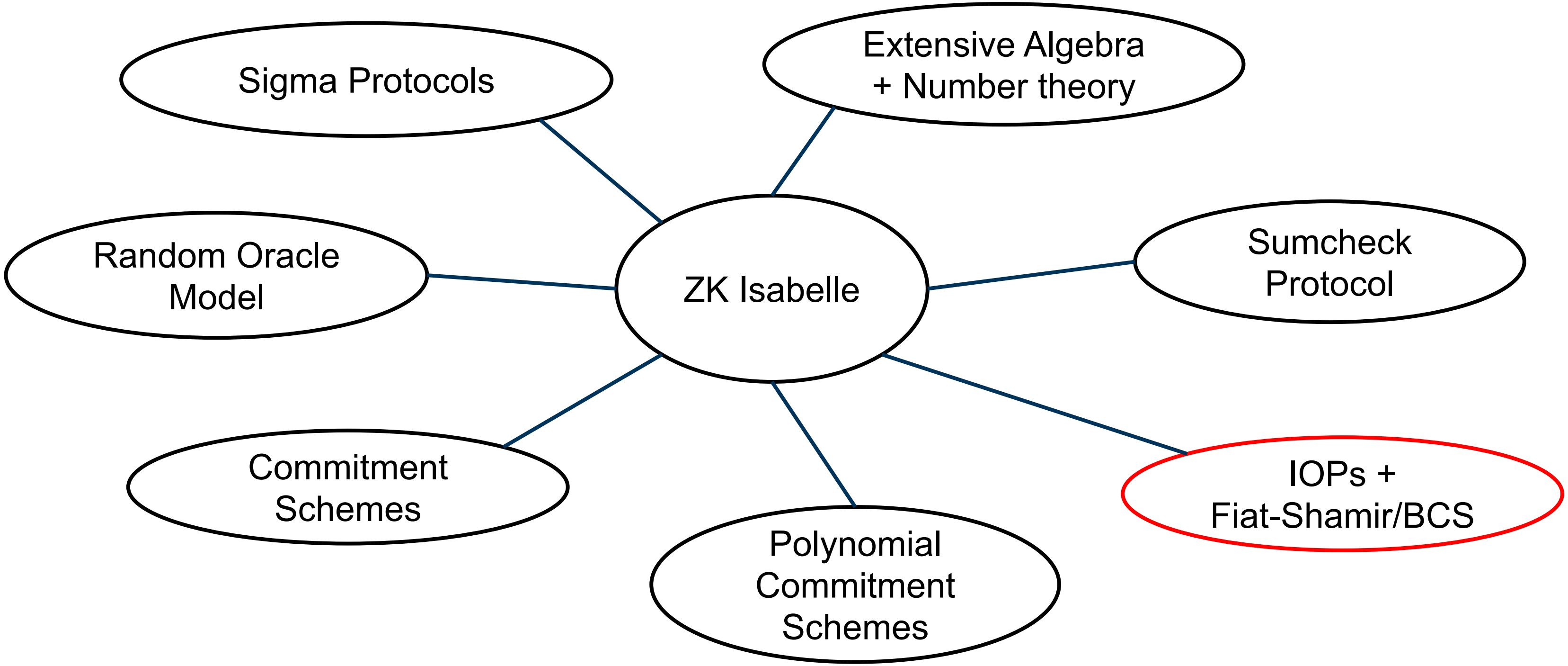
- batched KZG (evaluation):
  - Completeness
  - Binding
  - Evaluation Binding
  - Knowledge Soundness

bool

WIP: Formalization of the AGM

- “AGM compiler”
  - automated transformation of games in the standard model “functor”
  - adapt logging from ROM to AGM

The Isabelle ZK-Ecosystem

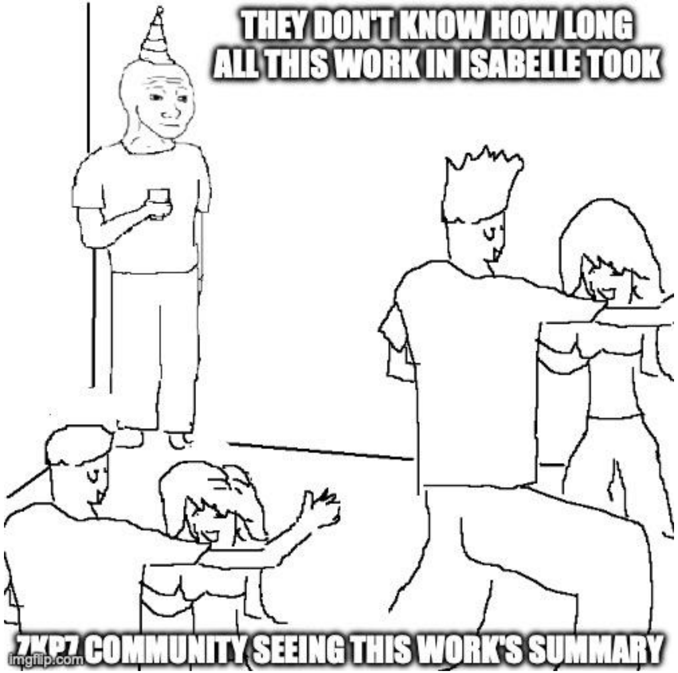


Future Work

- IOPs + Fiat-Shamir/BCS
  - ECC-based PCSs like FRI, WHIR, Binius etc.
  - interactive (multilinear) EC-based PCSs like Zeromorph, Halo
  - SNARKs

Summary

- formalized abstract PCS in Isabelle
- instantiated 2 KZG versions
  - Standard KZG
  - batched Eval KZG
- transformed paper proofs into SoG-style proofs
- verified SoG proofs in Isabelle using CryptHOL
- WIP: formalize AGM in Isabelle properly
- next Stop: IOPs + SNARKs



Polynomial Commitments in Isabelle

https://tobias-rothmann.github.io/papers/

Formalizing Polynomial Commitment Schemes

```
theorem weak_hiding:
  assumes lossless_𝒜: "⋀Q C W . lossless_spmf (𝒜 Q C I W)"
  and "distinct I"
  and "length I = max_deg"
  and "length I < CARD('e)"
shows "spmf (hiding_game I 𝒜) True ≤ spmf (DL_Gₚ.game (reduction I 𝒜)) True + (max_deg+1)/p"
proof -
  have "spmf (hiding_game I 𝒜) True = spmf (game1 I 𝒜) True"
    using hiding_game_to_game1[OF assms(2,3,4)] by presburger
  also have "... ≤ spmf (game2 I 𝒜) True + (max_deg+1)/p"
    using assms(1,2,3) fundamental_lemma_game1_game2 by blast
  also have "... = spmf (game2_w_assert I 𝒜) True + (max_deg+1)/p"
    using game2_to_game2_assert[OF assms(2,3,4)] by presburger
  also have "... ≤ spmf (game2_wo_assert I 𝒜) True + (max_deg+1)/p"
    using del_assert_game2 by auto
  also have "... = spmf (DL_Gₚ.game (reduction I 𝒜)) True + (max_deg+1)/p"
    using game2_wo_assert_to_DL_reduction_game by presburger
  finally show ?thesis .
qed
```