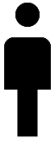
Improving the Efficiency of Function-Hiding Functional Commitments

Vanesa Daza, **Álvaro Montes** and Carla Ràfols

upf. Universitat Pompeu Fabra Barcelona

Enforcing Uniformity

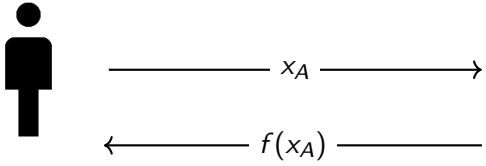


x_A: Alice's financial data

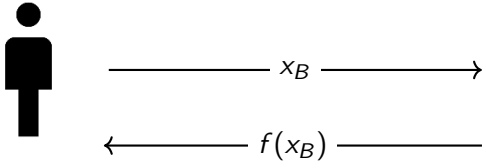
x_B: Bob's financial data

f: Secret credit score algorithm

Enforcing Uniformity



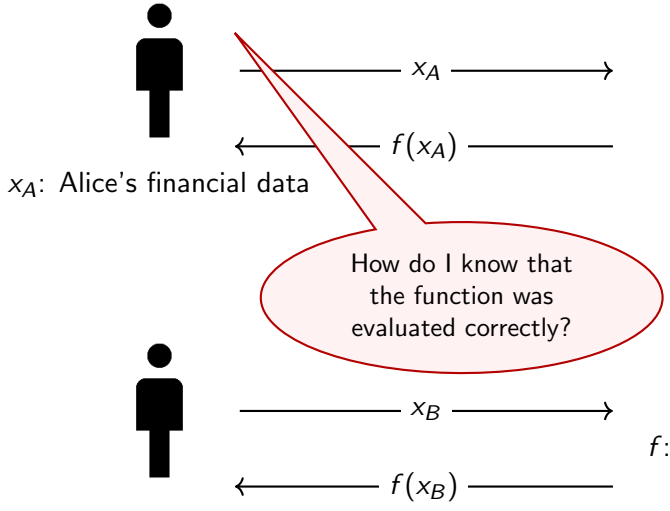
x_A: Alice's financial data



x_B: Bob's financial data

f: Secret credit score algorithm

Enforcing Uniformity

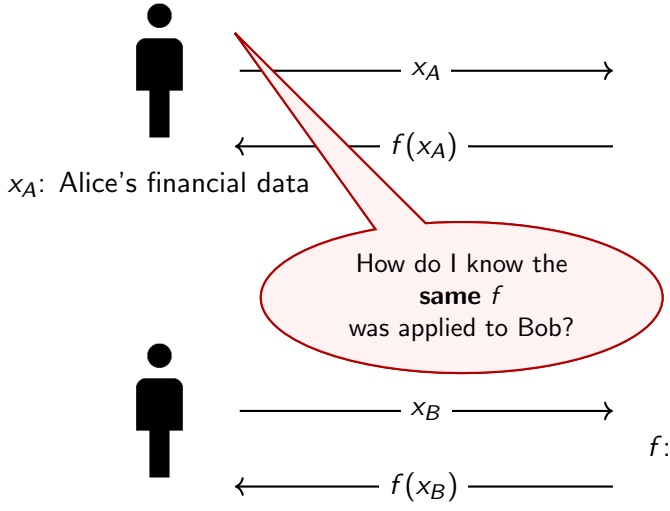


x_A: Alice's financial data

x_B: Bob's financial data

f: Secret credit score algorithm

Enforcing Uniformity



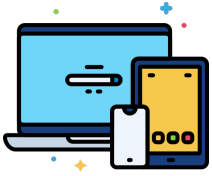
x_A: Alice's financial data

x_B: Bob's financial data

f: Secret credit score algorithm

Function-Hiding Functional Commitments [BNO21]

Verifier



Public input x

c

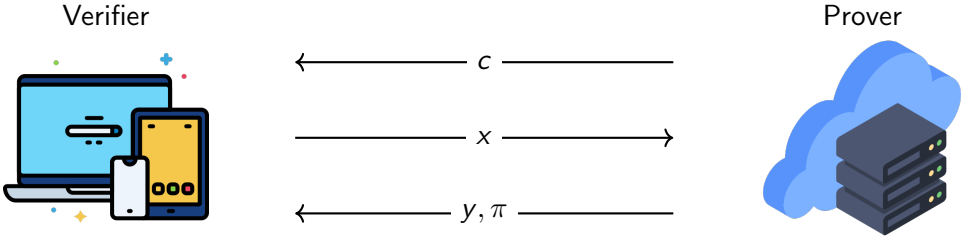
x

y, π

Prover

Private function f

Function-Hiding Functional Commitments [BNO21]

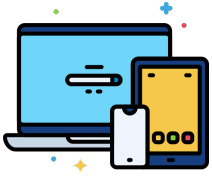


- c is a hiding and binding commitment to the function f
- π is a zero-knowledge proof for the relation

$$\hat{\mathcal{R}}_{\mathsf{FHFC}} = \{(c, x, y; f) \mid f(x) = y \ \wedge \ f \in \mathcal{F} \ \wedge \ \mathsf{Commit}(f) = c\}$$

(Input-Hiding) Functional Commitments [LRY16]

Verifier



Public function f

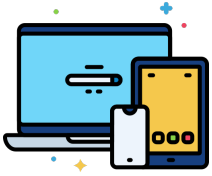
c_x, y, π

Prover

(Private) input x

(Input-Hiding) Functional Commitments [LRY16]

Verifier



Public function f

c_x, y, π

Prover

(Private) input x

- c_x is a (hiding) and binding commitment to the input x
- π (zero-knowledge) proof for $f(x) = y$

Particular examples of FHFC

- When $\mathcal{F} = \mathbb{F}^n$, *vector commitment schemes*

$$\hat{\mathcal{R}}_{\mathsf{VCS}} = \{(c, i, y; v) \mid v_i = y \ \wedge \ v \in \mathbb{F}^n \ \wedge \ \mathsf{Commit}(v) = c\}$$

Particular examples of FHFC

- When $\mathcal{F} = \mathbb{F}^n$, *vector commitment schemes*

$$\hat{\mathcal{R}}_{\mathsf{VCS}} = \{(c, i, y; v) \mid v_i = y \ \wedge \ v \in \mathbb{F}^n \ \wedge \ \mathsf{Commit}(v) = c\}$$

- When $\mathcal{F} = \mathbb{F}_{<n}[X]$, *polynomial commitment schemes*

$$\hat{\mathcal{R}}_{\mathsf{PCS}} = \{(c, x, y; p) \mid p(x) = y \ \wedge \ p(X) \in \mathbb{F}_{<n}[X] \ \wedge \ \mathsf{Commit}(p) = c\}$$

Particular examples of FHFC

- When $\mathcal{F} = \mathbb{F}^n$, *vector commitment schemes*

$$\hat{\mathcal{R}}_{\mathsf{VCS}} = \{(c, i, y; v) \mid v_i = y \ \wedge \ v \in \mathbb{F}^n \ \wedge \ \mathsf{Commit}(v) = c\}$$

- When $\mathcal{F} = \mathbb{F}_{<n}[X]$, *polynomial commitment schemes*

$$\hat{\mathcal{R}}_{\mathsf{PCS}} = \{(c, x, y; p) \mid p(x) = y \ \wedge \ p(X) \in \mathbb{F}_{<n}[X] \ \wedge \ \mathsf{Commit}(p) = c\}$$

In this work, $\mathcal{F}$ = the set of arithmetic circuits of bounded size

Universal zkSNARKs

A zkSNARK for a universal relation $\mathcal{R}$ is a tuple $\Pi = (\text{Setup}, \text{Derive}, \text{Prove}, \text{Verify})$

- $\text{Setup}(\mathcal{R}) \to \text{srs}$. Generates universal public parameters
- $\text{Derive}(\text{R} \in \mathcal{R}, \text{srs}) \to (\text{pk}_\text{R}, \text{vk}_\text{R})$. Generates relation-specific parameters
- $\text{Prove}(\text{pk}_\text{R}, x, w) \to \pi$. Produces a proof π for $(x, w) \in \text{R}$ of size $O(\log |w|)$
- $\text{Verify}(\text{vk}_\text{R}, x, \pi) \to b$. Accepts or rejects the proof π in time $O(\log |w|)$

Universal zkSNARKs

A zkSNARK for a universal relation $\mathcal{R}$ is a tuple $\Pi = (\text{Setup}, \text{Derive}, \text{Prove}, \text{Verify})$

- $\text{Setup}(\mathcal{R}) \to \text{srs}$. Generates universal public parameters
- $\text{Derive}(\text{R} \in \mathcal{R}, \text{srs}) \to (\text{pk}_\text{R}, \text{vk}_\text{R})$. Generates relation-specific parameters
- $\text{Prove}(\text{pk}_\text{R}, x, w) \to \pi$. Produces a proof π for $(x, w) \in \text{R}$ of size $O(\log |w|)$
- $\text{Verify}(\text{vk}_\text{R}, x, \pi) \to b$. Accepts or rejects the proof π in time $O(\log |w|)$

The verifier never reads R, it only receives the summary vk_R

Universal zkSNARKs

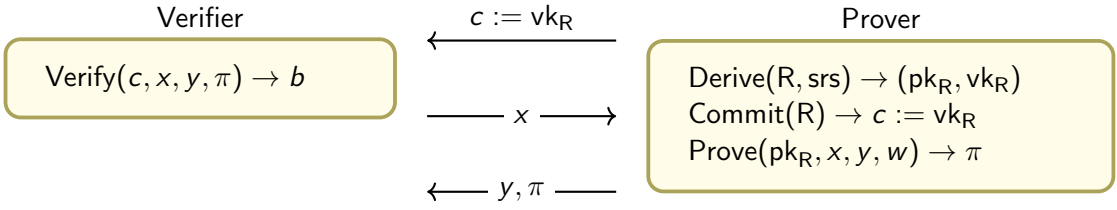
A zkSNARK for a universal relation $\mathcal{R}$ is a tuple $\Pi = (\text{Setup}, \text{Derive}, \text{Prove}, \text{Verify})$

- $\text{Setup}(\mathcal{R}) \to \text{srs}$. Generates universal public parameters
- $\text{Derive}(\text{R} \in \mathcal{R}, \text{srs}) \to (\text{pk}_\text{R}, \text{vk}_\text{R})$. Generates relation-specific parameters
- $\text{Prove}(\text{pk}_\text{R}, x, w) \to \pi$. Produces a proof π for $(x, w) \in \text{R}$ of size $O(\log |w|)$
- $\text{Verify}(\text{vk}_\text{R}, x, \pi) \to b$. Accepts or rejects the proof π in time $O(\log |w|)$

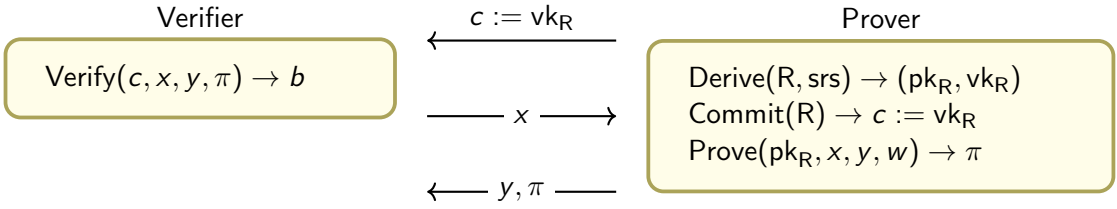
The verifier never reads R, it only receives the summary vk_R

Idea: Use vk_R as the commitment to R to build a FHFC

Do universal zkSNARKs directly imply FHFC?

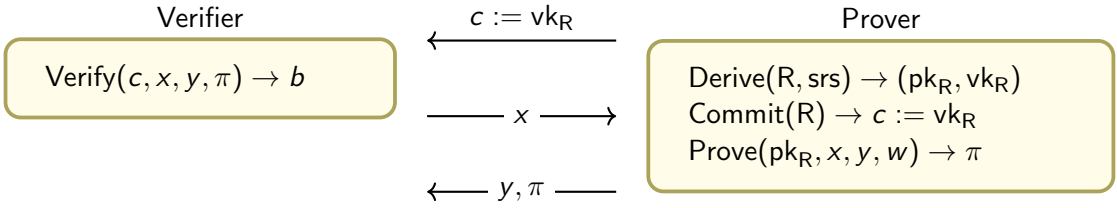


Do universal zkSNARKs directly imply FHFC?



Issue: Derive is assumed to be trusted $\implies$ not sound if run by the Prover

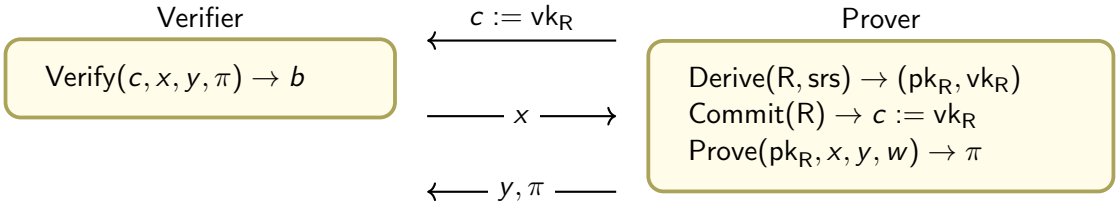
Do universal zkSNARKs directly imply FHFC?



Issue: The zero-knowledge proof π may leak information about the function

Solution: Standard zero-knowledge says that a proof for $(x, w) \in \mathsf{R}$ reveals no information about w. We need a stronger notion (*index-privacy* [BNO21]) so that no information about R is revealed

Do universal zkSNARKs directly imply FHFC?



Issue: The relation R may not represent a function!

Solution: Add a new protocol to ensure this is the case

Proof of Function Relation (PFR) [BNO21]

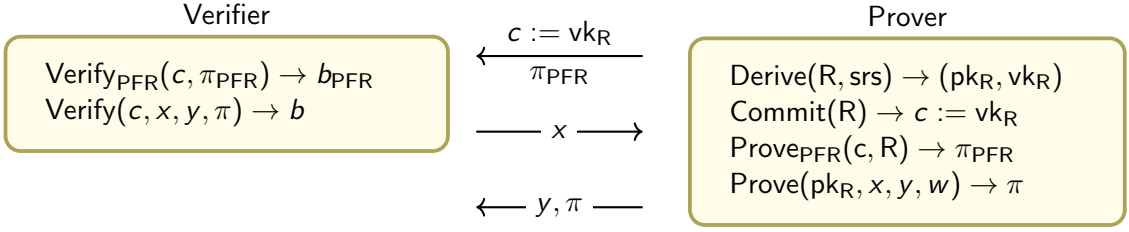
Functional Set: $\mathcal{D}_{\mathsf{func}} = \{\mathsf{R} \in \mathcal{R} \mid \forall x, \exists! y : (x, y) \in \mathcal{L}(\mathsf{R})\}$

PFR: $\hat{\mathcal{R}}_{\mathsf{PFR}} = \{(c, \mathsf{R}) \mid \mathsf{R} \in \mathcal{D}_{\mathsf{func}} \ \wedge \ c = \mathsf{Derive}_V(\mathsf{srs}, \mathsf{R}) \ \wedge \ \mathsf{Commit}(\mathsf{R}) = c\}$

Proof of Function Relation (PFR) [BNO21]

Functional Set: $\mathcal{D}_{\mathsf{func}} = \{\mathsf{R} \in \mathcal{R} \mid \forall x, \exists! y : (x, y) \in \mathcal{L}(\mathsf{R})\}$

PFR: $\hat{\mathcal{R}}_{\mathsf{PFR}} = \{(c, \mathsf{R}) \mid \mathsf{R} \in \mathcal{D}_{\mathsf{func}} \ \wedge \ c = \mathsf{Derive}_V(\mathsf{srs}, \mathsf{R}) \ \wedge \ \mathsf{Commit}(\mathsf{R}) = c\}$

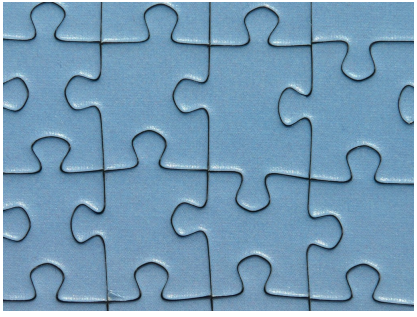


Building Function-Hiding Functional Commitments [BNO21]

Information Theoretic Primitive

Cryptographic Primitive

SNARK



Building Function-Hiding Functional Commitments [BNO21]

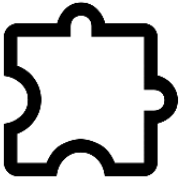
PIOP for $\mathcal{R}_{\mathsf{PFR}}$

Index-Private AHP

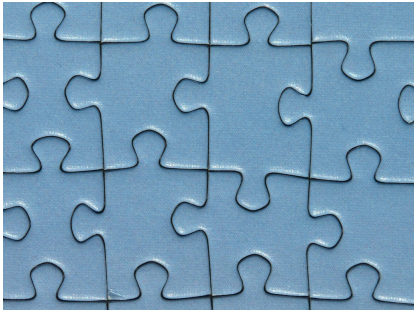
$\mathsf{R} \in \mathcal{D}_{\mathsf{func}}$

$(x, y) \in \mathcal{L}(\mathsf{R})$

Hiding PCS



Function-Hiding Functional Commitment



Algebraic Holographic Proofs [CHM+19]

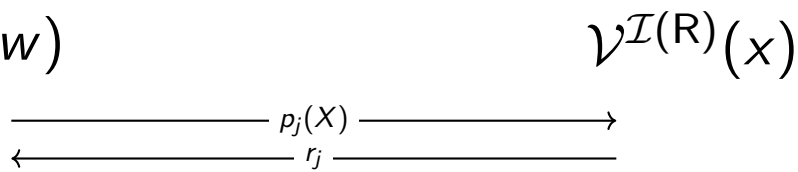
AHP $= (\mathcal{I}, \mathcal{P}, \mathcal{V})$ for a universal relation $\mathcal{R}$

- **Offline Phase**: The deterministic *indexer* $\mathcal{I}$ encodes the relation $\mathsf{R} \in \mathcal{R}$ into some polynomials $\mathbf{p}(X)$.

Algebraic Holographic Proofs [CHM+19]

AHP $= (\mathcal{I}, \mathcal{P}, \mathcal{V})$ for a universal relation $\mathcal{R}$

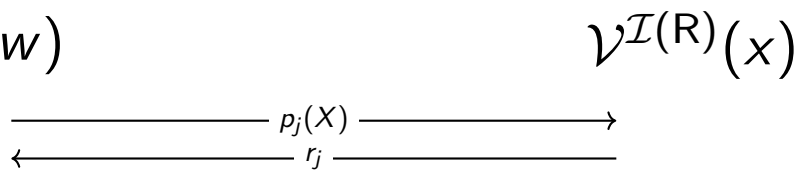
- **Offline Phase**: The deterministic *indexer* $\mathcal{I}$ encodes the relation $\mathsf{R} \in \mathcal{R}$ into some polynomials $\mathbf{p}(X)$.
- **Online Phase:**



Algebraic Holographic Proofs [CHM+19]

AHP $= (\mathcal{I}, \mathcal{P}, \mathcal{V})$ for a universal relation $\mathcal{R}$

- **Offline Phase**: The deterministic *indexer* $\mathcal{I}$ encodes the relation $\mathsf{R} \in \mathcal{R}$ into some polynomials $\mathbf{p}(X)$.
- **Online Phase:**

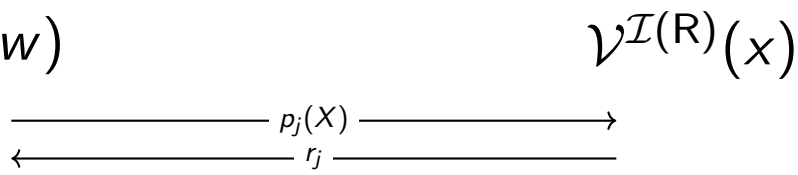


Zero-Knowledge: $\{\mathsf{Sim}(\mathsf{R}, x)\} \approx \left\{\mathsf{View}\left(\mathcal{P}(\mathsf{R}, x, w), \mathcal{V}^{\mathcal{I}(\mathsf{R})}(x)\right)\right\}$

Algebraic Holographic Proofs [CHM+19]

AHP $= (\mathcal{I}, \mathcal{P}, \mathcal{V})$ for a universal relation $\mathcal{R}$

- **Offline Phase**: The deterministic *indexer* $\mathcal{I}$ encodes the relation $\mathsf{R} \in \mathcal{R}$ into some polynomials $\mathbf{p}(X)$.
- **Online Phase:**



Zero-Knowledge: $\{\mathsf{Sim}(\mathsf{R}, x)\} \approx \left\{\mathsf{View}\left(\mathcal{P}(\mathsf{R}, x, w), \mathcal{V}^{\mathcal{I}(\mathsf{R})}(x)\right)\right\}$

Index-Privacy [BNO21]: $\{\mathsf{Sim}(x)\} \approx \left\{\mathsf{View}\left(\mathcal{P}(\mathsf{R}, x, w), \mathcal{V}^{\mathcal{I}(\mathsf{R})}(x)\right)\right\}$

Instantiating the compiler: Efficiency Issues

[BNO21] instantiate their compiler with a hiding variant of KZG, proof of ≈ 21 kB

Instantiating the compiler: Efficiency Issues

[BNO21] instantiate their compiler with a hiding variant of KZG, proof of ≈ 21 kB

1. Hiding PCS $\implies$ cannot use plain KZG

Instantiating the compiler: Efficiency Issues

[BNO21] instantiate their compiler with a hiding variant of KZG, proof of ≈ 21 kB

1. Hiding PCS $\implies$ cannot use plain KZG
2. Standard AHP+PCS compilation does not cover some optimizations (batching, linearization tricks...)

Instantiating the compiler: Efficiency Issues

[BNO21] instantiate their compiler with a hiding variant of KZG, proof of ≈ 21 kB

1. Hiding PCS $\implies$ cannot use plain KZG
2. Standard AHP+PCS compilation does not cover some optimizations (batching, linearization tricks...)
3. Expensive techniques to achieve index-privacy ($\approx 2\times$ number of polynomials)

Instantiating the compiler: Efficiency Issues

[BNO21] instantiate their compiler with a hiding variant of KZG, proof of ≈ 21 kB

1. Hiding PCS $\implies$ cannot use plain KZG
2. Standard AHP+PCS compilation does not cover some optimizations (batching, linearization tricks...)
3. Expensive techniques to achieve index-privacy ($\approx 2\times$ number of polynomials)

Can we do better?

Lunar framework [CFF+20]

- Compilation with CP-SNARKs instead of PCS

Lunar framework [CFF+20]

- Compilation with CP-SNARKs instead of PCS
- AHP $\rightarrow$ PHPs. Allows for a more fine grained zero-knowledge analysis

Lunar framework [CFF+20]

- Compilation with CP-SNARKs instead of PCS
- AHP $\rightarrow$ PHPs. Allows for a more fine grained zero-knowledge analysis
- Can build zkSNARKs based on KZG

Lunar framework [CFF+20]

- Compilation with CP-SNARKs instead of PCS
- AHP $\rightarrow$ PHPs. Allows for a more fine grained zero-knowledge analysis
- Can build zkSNARKs based on KZG

Bounded Zero-Knowledge: Zero-knowledge holds after leaking some evaluations of the polynomials.

$$\left\{\left(\mathsf{View}(\mathcal{P}(\mathsf{R}, x, w), \mathcal{V}^{\mathcal{I}(\mathsf{R})}(x)), (p_i(y))_{(i,y)\in\mathcal{L}}\right)\right\} \approx \{\mathsf{Sim}(\mathsf{R}, x, \mathcal{L})\}$$

Lunar framework [CFF+20]

- Compilation with CP-SNARKs instead of PCS
- AHP $\rightarrow$ PHPs. Allows for a more fine grained zero-knowledge analysis
- Can build zkSNARKs based on KZG

Bounded Zero-Knowledge: Zero-knowledge holds after leaking some evaluations of the polynomials.

$$\left\{\left(\mathsf{View}(\mathcal{P}(\mathsf{R}, x, w), \mathcal{V}^{\mathcal{I}(\mathsf{R})}(x)), (p_i(y))_{(i,y)\in\mathcal{L}}\right)\right\} \approx \{\mathsf{Sim}(\mathsf{R}, x, \mathcal{L})\}$$

It is **crucial** to be able to leak evaluations of polynomials, even if they contain information about the witness

Lunar framework [CFF+20]

- Compilation with CP-SNARKs instead of PCS
- AHP $\rightarrow$ PHPs. Allows for a more fine grained zero-knowledge analysis
- Can build zkSNARKs based on KZG

Bounded Zero-Knowledge: Zero-knowledge holds after leaking some evaluations of the polynomials.

$$\left\{\left(\mathsf{View}(\mathcal{P}(\mathsf{R}, x, w), \mathcal{V}^{\mathcal{I}(\mathsf{R})}(x)), (p_i(y))_{(i,y)\in\mathcal{L}}\right)\right\} \approx \{\mathsf{Sim}(\mathsf{R}, x, \mathcal{L})\}$$

It is **crucial** to be able to leak evaluations of polynomials, even if they contain information about the witness

Since $\mathcal{I}(\mathsf{R}) \rightarrow \mathbf{p}(X)$ is deterministic, knowledge of R is required to simulate their evaluations.

Our solution: PHP with randomized indexer (rPHP)

rPHPs allow the indexer to use randomness when encoding the relation

Offline Phase: $\mathcal{I}(\mathsf{R}, \mathsf{r}) \rightarrow \mathbf{p}(X)$ for $\mathsf{R} \in \mathcal{R}$ and $\mathsf{r} \leftarrow \mathsf{Rand}$

Our solution: PHP with randomized indexer (rPHP)

rPHPs allow the indexer to use randomness when encoding the relation

Offline Phase: $\mathcal{I}(\mathsf{R}, \mathsf{r}) \to \mathbf{p}(X)$ for $\mathsf{R} \in \mathcal{R}$ and $\mathsf{r} \leftarrow \mathsf{Rand}$

Bounded Index-Privacy:

$$\left\{ \left(\mathsf{View}(\mathcal{P}(\mathsf{R}, \mathsf{r}, x, w), \mathcal{V}^{\mathcal{I}(\mathsf{R},\mathsf{r})}(\mathsf{x})), (p_i(y))_{(i,y)\in\mathcal{L}} \right) \mid \mathsf{r} \leftarrow \mathsf{Rand} \right\} \approx \{\mathsf{Sim}(x, \mathcal{L})\}$$

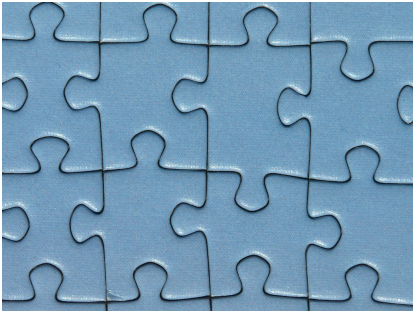
Our compiler for FHFC

Bounded Index-Private rPHP

CP-SNARK for $\hat{\mathcal{R}}_{\mathsf{PFR}}$

Other CP-SNARKs

Function-Hiding Functional Commitment



When does an R1CS instance define a function?

Lemma ([BNO21])

If $A, B \in \mathbb{F}^{n \times n}$ are t-strictly lower triangular and $C \in \mathbb{F}^{n \times n}$ is t-diagonal, for any $x \in \mathbb{F}^t$ there exists a unique $z \in \mathbb{F}^n$, $z = (x, w, y)$ such that $Az \circ Bz = Cz$

When does an R1CS instance define a function?

Lemma ([BNO21])

If $A, B \in \mathbb{F}^{n \times n}$ are t-strictly lower triangular and $C \in \mathbb{F}^{n \times n}$ is t-diagonal, for any $x \in \mathbb{F}^t$ there exists a unique $z \in \mathbb{F}^n$, $z = (x, w, y)$ such that $Az \circ Bz = Cz$

$$A = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ a_{t+1,1} & a_{t+1,2} & \cdots & a_{t+1,t} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,t} & a_{n,t+1} & \cdots & 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 & & & & & & \\ & \ddots & & & & & \\ & & 0 & & & & \\ & & & c_{t+1} & & & \\ & & & & & \ddots & \\ & & & & & & c_n \end{pmatrix}$$

Results and Conclusion

- We introduce a new information theoretic primitive, rPHPs, with a more fine-grained index-privacy notion

Results and Conclusion

- We introduce a new information theoretic primitive, rPHPs, with a more fine-grained index-privacy notion
- Using rPHPs and CP-SNARKs, we build a FHFC compiler that allows the use of KZG

Results and Conclusion

- We introduce a new information theoretic primitive, rPHPs, with a more fine-grained index-privacy notion
- Using rPHPs and CP-SNARKs, we build a FHFC compiler that allows the use of KZG
- We design simpler, more efficient PFRs, leveraging the relaxed zero-knowledge notions of our building blocks

Results and Conclusion

- We introduce a new information theoretic primitive, rPHPs, with a more fine-grained index-privacy notion
- Using rPHPs and CP-SNARKs, we build a FHFC compiler that allows the use of KZG
- We design simpler, more efficient PFRs, leveraging the relaxed zero-knowledge notions of our building blocks
- Our FHFC based on Marlin and Plonk are concretely efficient, with proof sizes $\approx$ 1.6 kB, whereas [BNO21] were $\approx$ 21 kB

Improving the Efficiency of Function-Hiding Functional Commitments

Vanesa Daza, **Álvaro Montes** and Carla Ràfols

upf. Universitat Pompeu Fabra Barcelona

References I


Dan Boneh, Wilson Nguyen, and Alex Ozdemir.
Efficient functional commitments: How to commit to a private function.
Cryptology ePrint Archive, Paper 2021/1342, 2021.


Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez.
Lunar: a toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions.
Cryptology ePrint Archive, Paper 2020/1069, 2020.


Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas Ward.
Marlin: Preprocessing zkSNARKs with universal and updatable SRS.
Cryptology ePrint Archive, Paper 2019/1047, 2019.

References II

Benoît Libert, Somindu C. Ramanna, and Moti Yung.
Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions.
Cryptology ePrint Archive, Paper 2016/766, 2016.