(full version here)

Curve Forests

Transparent Zero-Knowledge Set-Membership with Batching and Strong Security

Matteo Campanelli (Offchain Labs)
Mathias Hall-Andersen (ZKSecurity)
Simon Holmgaard Kamp (CISPA Helmholtz Center for Information Security)

Zero-Knowledge Set Membership

The general problem

A set of objects *S*
(the *set* is usually public)

Zero-Knowledge Set Membership

The general problem

A set of objects *S*
(the *set* is usually public)

e.g. (whitelist):
S = *users authorized to perform Foo*

Zero-Knowledge Set Membership

The general problem

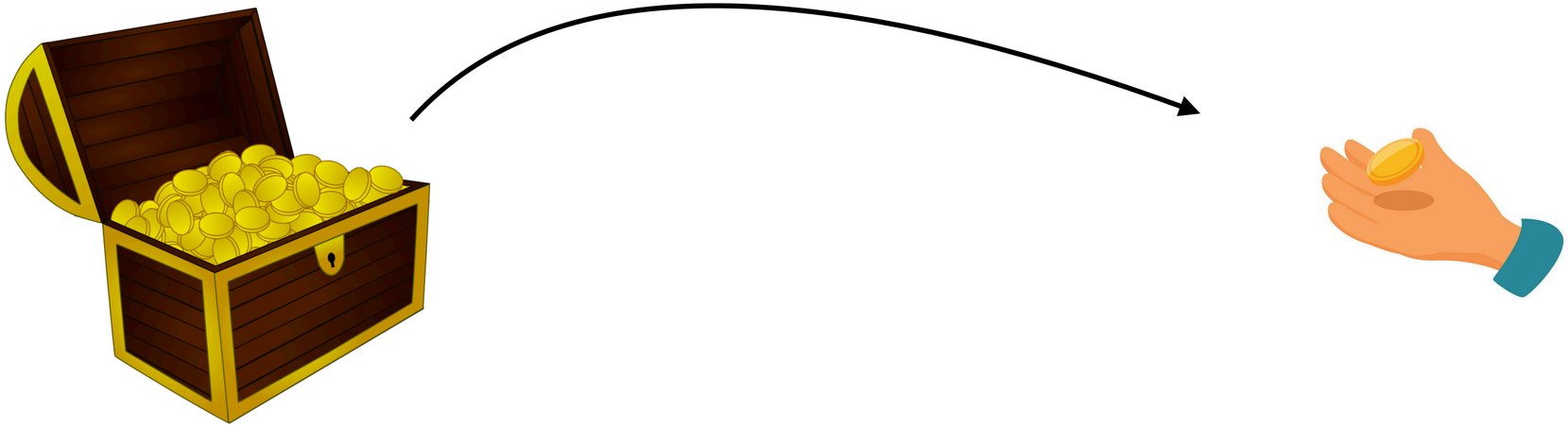
A set of objects *S*
(the *set* is usually public)

e.g. (whitelist):
S = *users authorized to perform Foo*

e.g. (anonymous payment):
S = *set of existing coins*
(NB: here coins are public, but they do not disclose their owners, etc)

Zero-Knowledge Set Membership

The general problem



A set of objects *S*
(the *set* is usually public)

e.g. (whitelist):
S = *users authorized to perform Foo*

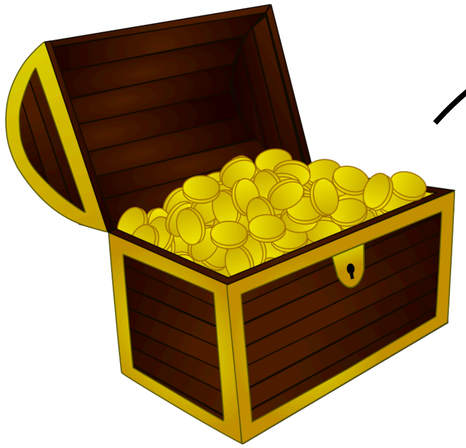
e.g. (anonymous payment):
S = *set of existing coins*
(NB: here coins are public, but they do not disclose their owners, etc)

Zero-Knowledge Set Membership

The general problem

Generate *prf*

“x ∈ S ∧ SomeProperty(x)”



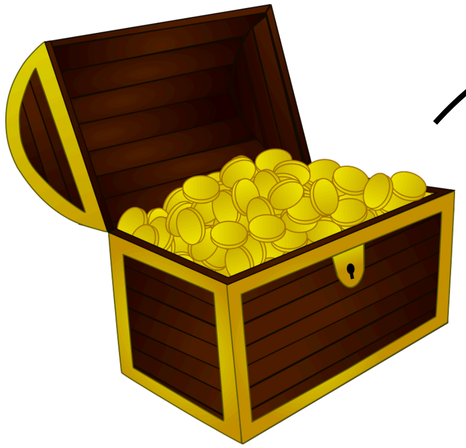
A set of objects *S*
(the *set* is usually public)

e.g. (whitelist):
S = *users authorized to perform Foo*

e.g. (anonymous payment):
S = *set of existing coins*
(NB: here coins are public, but they do not disclose their owners, etc)

Zero-Knowledge Set Membership

The <u>general</u> problem



A set of objects *S*
(the *set* is usually public)

Generate *prf*

"x ∈ S ∧ SomeProperty(x)"

(recall:
Zero-Knowledge
refers to ***hiding***.)

e.g. (whitelist):
S = *users authorized to perform Foo*

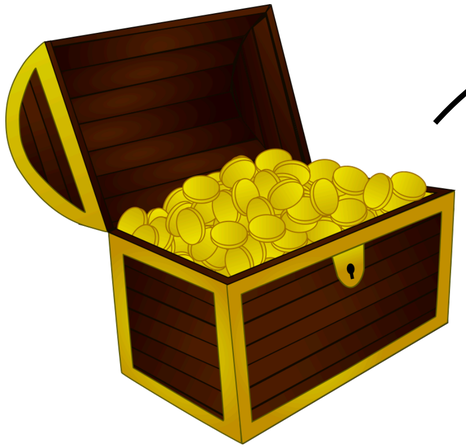
e.g. (anonymous payment):
S = *set of existing coins*
(NB: here <u>coins</u> are public, but they do not disclose their owners, etc)

Zero-Knowledge Set Membership

The general problem

Generate *prf*

"x ∈ S ∧ SomeProperty(x)"



(recall:
Zero-Knowledge
refers to ***hiding***.)

A set of objects *S*
(the *set* is usually public)

A commitment *cm*
to the object *x*

e.g. (whitelist):
S = *users authorized to perform Foo*

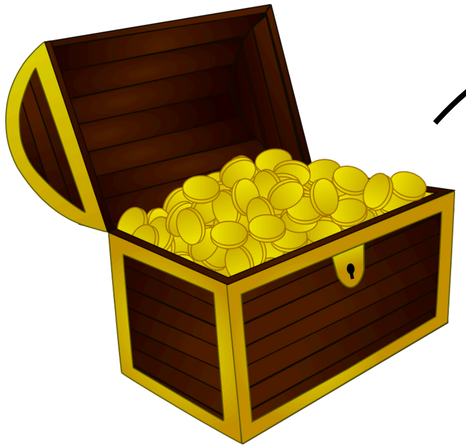
e.g. (anonymous payment):
S = *set of existing coins*
(NB: here coins are public, but they do not disclose their owners, etc)

Zero-Knowledge Set Membership

The general problem

Generate *prf*

"x ∈ S ∧ SomeProperty(x)"



(recall:
Zero-Knowledge
refers to ***hiding***.)

A set of objects *S*
(the *set* is usually public)

A commitment *cm*
to the object *x*

e.g. (whitelist):
S = *users authorized to perform Foo*

Verify*(S, cm, prf)*

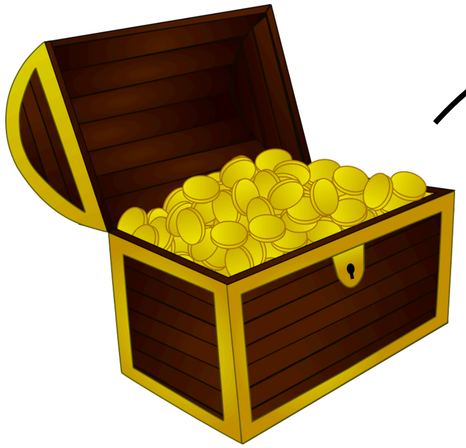
e.g. (anonymous payment):
S = *set of existing coins*
(NB: here coins are public, but they do not disclose their owners, etc)

Zero-Knowledge Set Membership

The general problem

Generate *prf*

"x ∈ S ∧ SomeProperty(x)"



(recall:
Zero-Knowledge
refers to ***hiding***.)

A set of objects *S*
(the *set* is usually public)

A commitment *cm*
to the object *x*

dig_S

Verify(~~*S*~~, *cm*, *prf*)

e.g. (whitelist):
S = *users authorized to perform Foo*

e.g. (anonymous payment):
S = *set of existing coins*
(NB: here coins are public, but they do not disclose their owners, etc)

Applications of ZK Set Membership

Applications of ZK Set Membership

- **Anonymous Credentials & Decentralized Identity**

Applications of ZK Set Membership

- **Anonymous Credentials & Decentralized Identity**
- **Whitelists**

Applications of ZK Set Membership

- **Anonymous Credentials & Decentralized Identity**
- **Whitelists**
  - (Anti-money laundering, reputation validation, access control, etc)

Applications of ZK Set Membership

- **Anonymous Credentials & Decentralized Identity**
- **Whitelists**
  - (Anti-money laundering, reputation validation, access control, etc)
- **Anonymous Cryptocurrencies**

Applications of ZK Set Membership

- **Anonymous Credentials & Decentralized Identity**
- **Whitelists**
  - (Anti-money laundering, reputation validation, access control, etc)
- **Anonymous Cryptocurrencies**
  - ZCash, Monero, Firo, …

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

dig_S

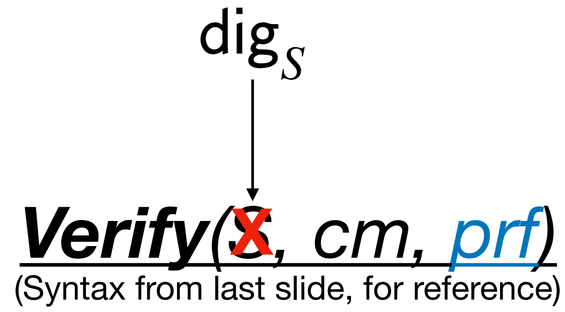
Verify(~~S~~, *cm*, *prf*)

(Syntax from last slide, for reference)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:



(Syntax from last slide, for reference)

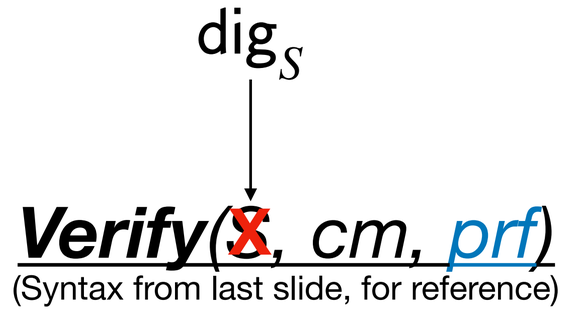
Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time

small proofs



(Syntax from last slide, for reference)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time  *small proofs*

dig_S

Verify(~~S~~, *cm*, *prf*)

(Syntax from last slide, for reference)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time  *small proofs*

Extra (but common) requirement: transparent setup

dig_S

Verify(~~S~~, *cm*, *prf*)

(Syntax from last slide, for reference)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time  *small proofs*

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

dig_S

Verify(~~x~~, *cm*, *prf*)

(Syntax from last slide, for reference)

Concretely Efficient ZK Set Members rd

Let's look at its challenges

A trusted party is not always available.
(Mind: get to breakfast by 7am before they run out, usually together with the potato salad)

Usual core requirements are:

fast proving time  *small proofs*

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

dig_S

Verify(~~S~~, *cm*, *prf*)

(Syntax from last slide, for reference)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time  *small proofs*

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

dig_S

Verify(~~x~~, *cm*, *prf*)

(Syntax from last slide, for reference)

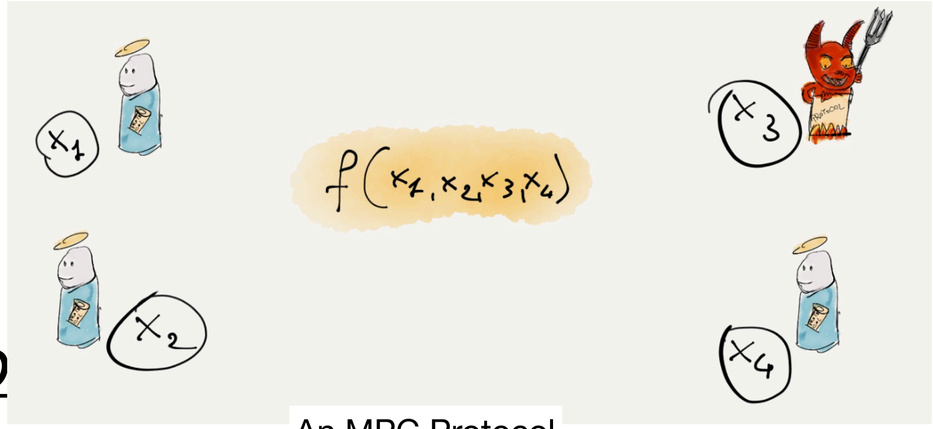
Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time

small proo



An MPC Protocol

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

dig_S

Verify(~~x~~, *cm*, *prf*)

(Syntax from last slide, for reference)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time  *small proofs*

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

dig_S

Verify(~~x~~, *cm*, *prf*)

(Syntax from last slide, for reference)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time

small p

(The least of your problems during a *setup ceremony: cryptographers handling circular saws.*)

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

dig_S

Verify(~~$\mathcal{X}$~~, *cm*, *prf*)

(Syntax from last slide, for reference)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time  *small proofs*

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

dig_S

Verify(~~x~~, *cm*, *prf*)

(Syntax from last slide, for reference)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time  *small proofs*

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

Merkle Trees

Pairings
([Groth16],
Verkle Trees (KZG),…)

RSA
([B**C**F+21],[**C**FH+22] …)

Verify(~~S~~, *cm*, *prf*)
(Syntax from last slide, for reference)

[BCF+21]: Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular (FC21)
[CFH+22]: Succinct Zero-Knowledge Batch Proofs for Set Accumulators (CCS22)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time  *small proofs*

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

Merkle Trees ~~X~~

Pairings
([Groth16],
Verkle Trees (KZG),…)

RSA
([B**C**F+21],[**C**FH+22] …)

dig_S

Verify(~~S~~, *cm*, *prf*)
(Syntax from last slide, for reference)

[BCF+21]: Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular (FC21)
[CFH+22]: Succinct Zero-Knowledge Batch Proofs for Set Accumulators (CCS22)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

Usual core requirements are:

fast proving time  *small proofs*

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

~~**Merkle Trees**~~

~~**Pairings**~~
([Groth16],
Verkle Trees (KZG),…)

~~**RSA**~~
([BCF+21],[CFH+22] …)

dig_S

Verify(~~S~~, *cm*, *prf*)
(Syntax from last slide, for reference)

[BCF+21]: Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular (FC21)
[CFH+22]: Succinct Zero-Knowledge Batch Proofs for Set Accumulators (CCS22)

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

State of the art*: **DLOG-based**

Curve Trees
(same authors, USENIX23)

Usual core requirements are:

fast proving time

small proofs

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

~~**Merkle Trees**~~

~~**Pairings**~~
([Groth16],
Verkle Trees (KZG),…)

~~**RSA**~~
([BCF+21],[CFH+22] …)

[BCF+21]: Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular (FC21)
[CFH+22]: Succinct Zero-Knowledge Batch Proofs for Set Accumulators (CCS22)

dig_S

Verify(~~S~~, *cm*, *prf*)
(Syntax from last slide, for reference)

* There are designs w/ similar efficiency metrics, but they pay in generality, simplicity, or how established/safe their cryptographic assumptions are (see USENIX paper).

Concretely Efficient ZK Set Membership is Hard

Let's look at its challenges

State of the art*: **DLOG-based**

Curve Trees
(same authors, USENIX23)

Usual core requirements are:

fast proving time

small proofs

Our starting point
(see next slide)

Extra (but common) requirement: transparent setup

("No trusted party necessary to generate cryptographic parameters")

~~**Merkle Trees**~~

~~**Pairings**~~
([Groth16],
Verkle Trees (KZG),…)

~~**RSA**~~
([BCF+21],[CFH+22] …)

[BCF+21]: Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular (FC21)
[CFH+22]: Succinct Zero-Knowledge Batch Proofs for Set Accumulators (CCS22)

dig_S

Verify(~~S~~, *cm*, *prf*)
(Syntax from last slide, for reference)

* There are designs w/ similar efficiency metrics, but they pay in generality, simplicity, or how established/safe their cryptographic assumptions are (see USENIX paper).

This Work in a nutshell:

Improving Curve Trees and specializing it to the batch setting

(defined in the next slide)

This Work in a nutshell:

Improving Curve Trees and specializing it to the batch setting

(defined in the next slide)

Curve Trees are a natural choice:

This Work in a nutshell:

Improving Curve Trees and specializing it to the batch setting

(defined in the next slide)

Curve Trees are a natural choice:

1. “sandbox for algebraic tricks”

This Work in a nutshell:

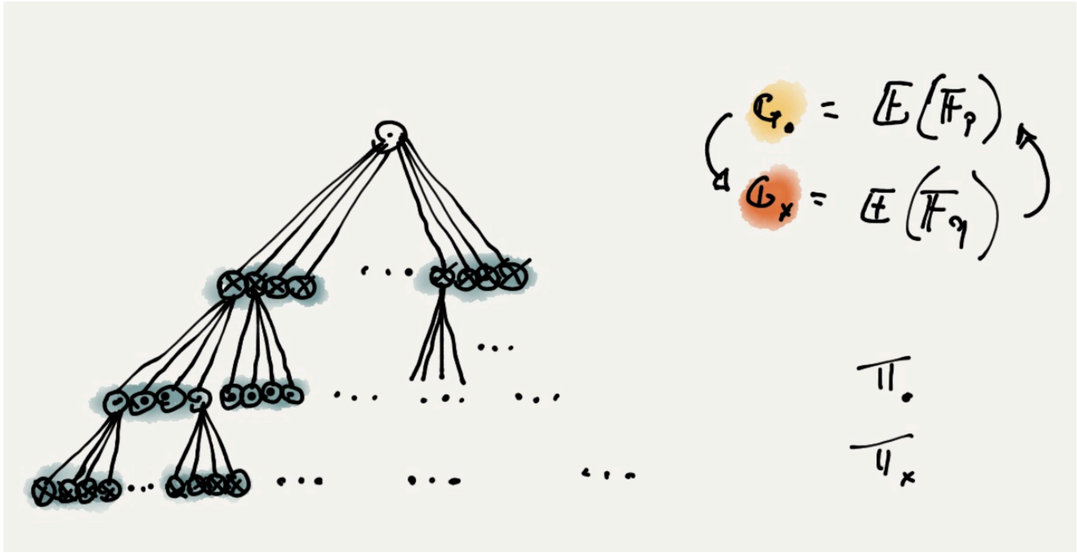
Improving Curve Trees and specializing it to the batch setting

(defined in the next slide)

Curve Trees are a natural choice:

1. “sandbox for algebraic tricks”

CurveTree ≈ n-ary Merkle Tree (with specifics)



A Curve Tree (simplified)

This Work in a nutshell:

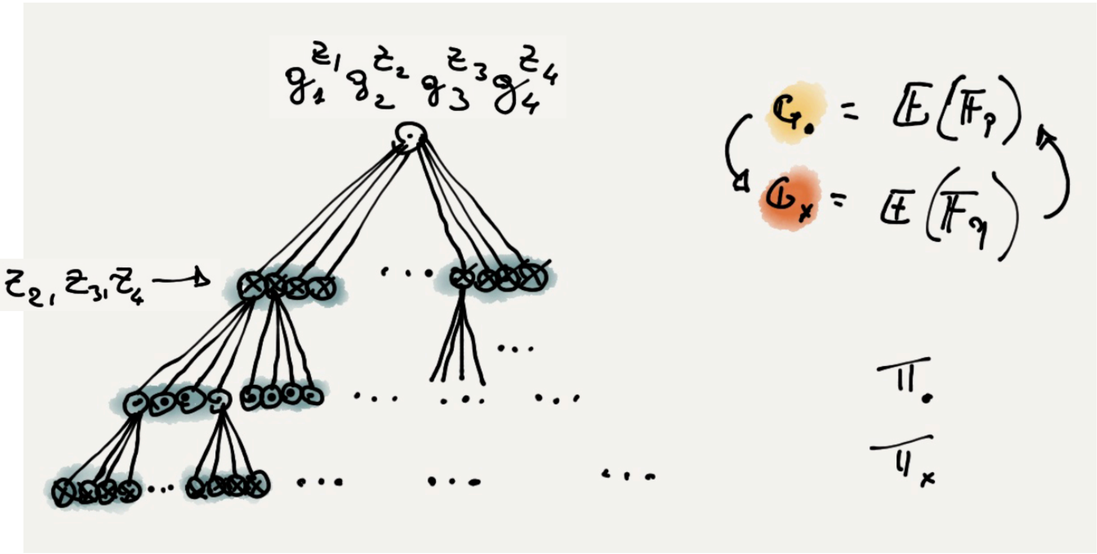
Improving Curve Trees and specializing it to the batch setting

(defined in the next slide)

Curve Trees are a natural choice:

1. “sandbox for algebraic tricks”

CurveTree ≈ n-ary Merkle Tree (with specifics)



A Curve Tree (simplified)

This Work in a nutshell:

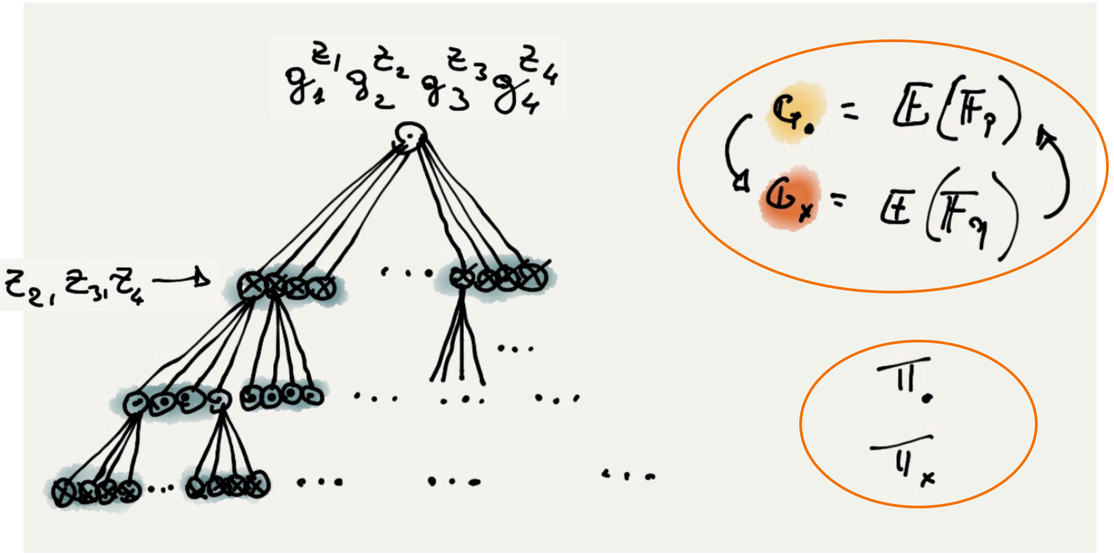
Improving Curve Trees and specializing it to the batch setting

(defined in the next slide)

Curve Trees are a natural choice:

1. “sandbox for algebraic tricks”

CurveTree ≈ n-ary Merkle Tree (with specifics)
+ Pedersen Hash



A Curve Tree (simplified)

This Work in a nutshell:

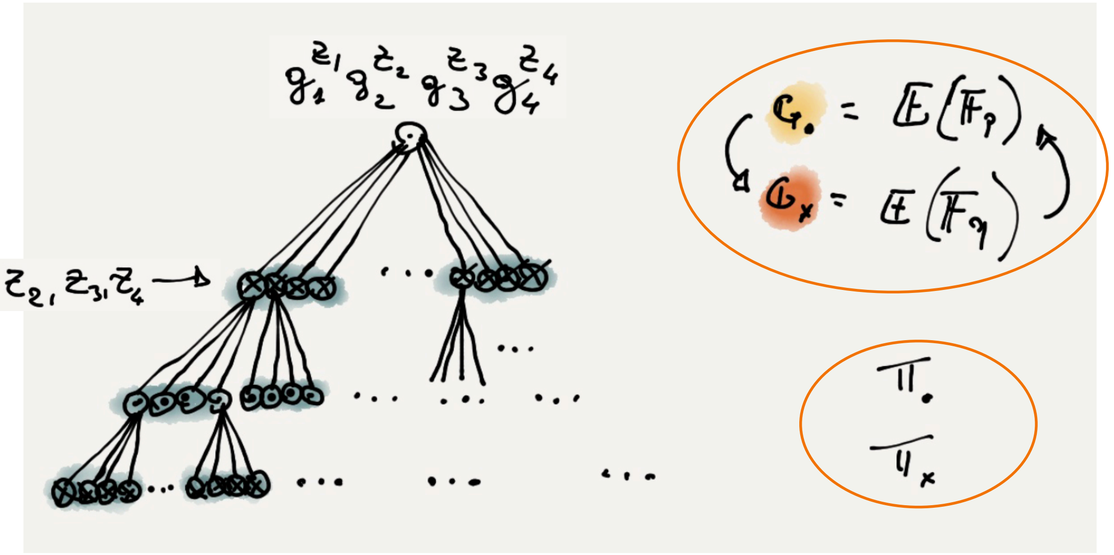
Improving Curve Trees and specializing it to the batch setting

(defined in the next slide)

Curve Trees are a natural choice:

1. “sandbox for algebraic tricks”

CurveTree ≈ n-ary Merkle Tree (with specifics)
+ Pedersen Hash
+ cycle of curves



A Curve Tree (simplified)

$$\mathcal{U}_{\alpha,\beta}(v) : \mathbb{F} \to \{0,1\} \quad \mathcal{U}_{\alpha,\beta}(v) \mapsto S(\alpha \cdot v + \beta)$$

$$\mathcal{P}_{\mathbb{E}} = \{(\mathrm{x},\mathrm{y}) \mid (\mathrm{x},\mathrm{y}) \in \mathbb{E}(\mathbb{F}_p) \wedge \mathcal{U}_{\alpha,\beta}(\mathrm{y}) = 1 \wedge \mathcal{U}_{\alpha,\beta}(-\mathrm{y}) = 0\}$$

This Work in a nutshell:

Improving Curve Trees and specializing it to the batch setting

(defined in the next slide)

Curve Trees are a natural choice:

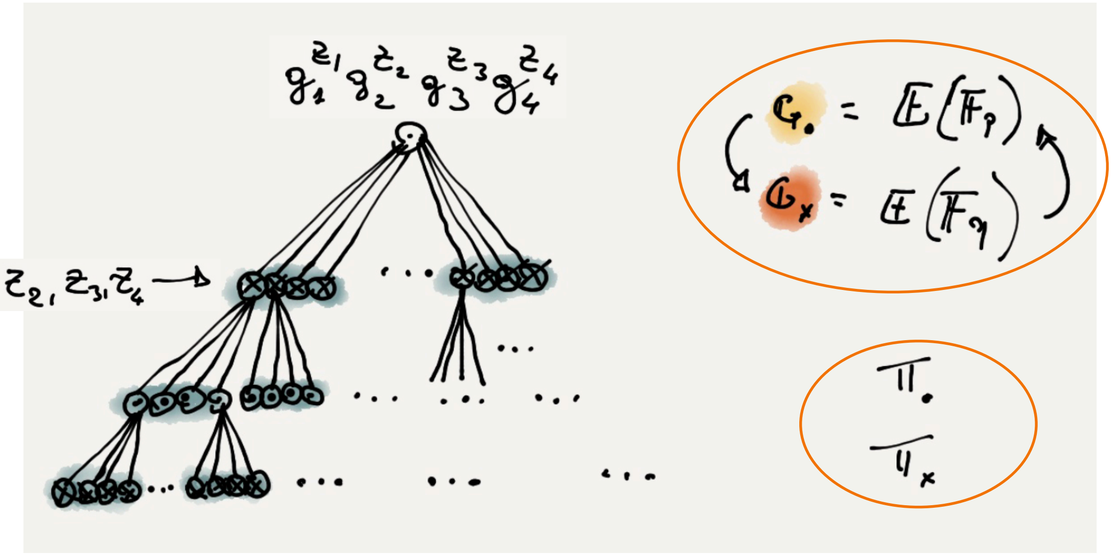
1. “sandbox for algebraic tricks”

CurveTree ≈ n-ary Merkle Tree (with specifics)

\+ Pedersen Hash

\+ cycle of curves

\+ [tricks]



A Curve Tree (simplified)

$$\mathcal{U}_{\alpha,\beta}(v) : \mathbb{F} \to \{0,1\} \quad \mathcal{U}_{\alpha,\beta}(v) \mapsto S(\alpha \cdot v + \beta)$$

$$\mathcal{P}_{\mathbb{E}} = \{(\mathrm{x},\mathrm{y}) \mid (\mathrm{x},\mathrm{y}) \in \mathbb{E}(\mathbb{F}_p) \wedge \mathcal{U}_{\alpha,\beta}(\mathrm{y}) = 1 \wedge \mathcal{U}_{\alpha,\beta}(-\mathrm{y}) = 0\}$$

This Work in a nutshell:

Improving Curve Trees and specializing it to the batch setting

(defined in the next slide)

Curve Trees are a natural choice:

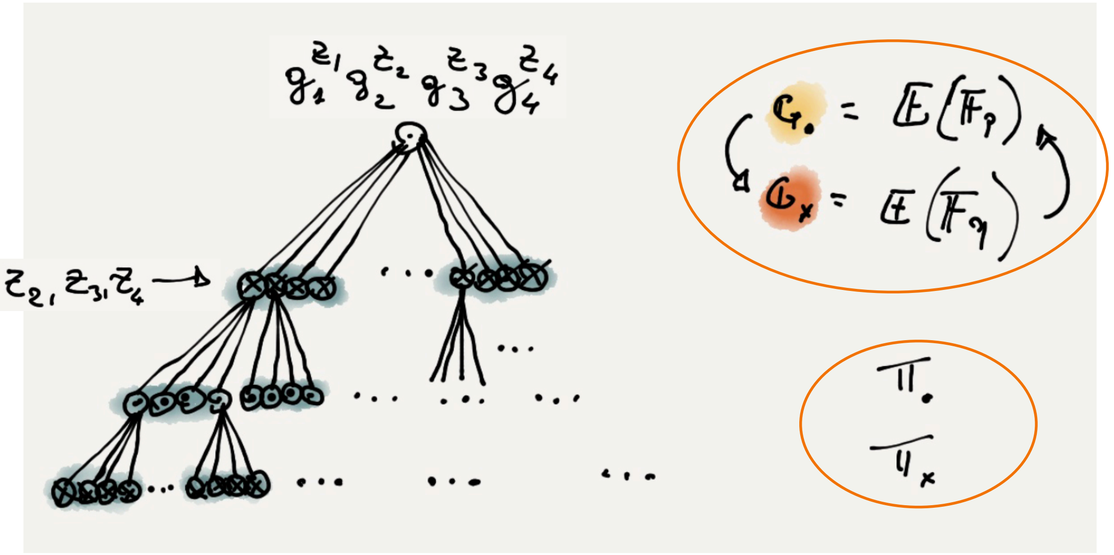
1. “sandbox for algebraic tricks”

CurveTree ≈ n-ary Merkle Tree (with specifics)

\+ Pedersen Hash

\+ cycle of curves

\+ [tricks]



A Curve Tree (simplified)

$$\mathcal{U}_{\alpha,\beta}(v) : \mathbb{F} \to \{0,1\} \quad \mathcal{U}_{\alpha,\beta}(v) \mapsto S(\alpha \cdot v + \beta)$$

$$\mathcal{P}_{\mathbb{E}} = \{(\mathrm{x},\mathrm{y}) \mid (\mathrm{x},\mathrm{y}) \in \mathbb{E}(\mathbb{F}_p) \wedge \mathcal{U}_{\alpha,\beta}(\mathrm{y}) = 1 \wedge \mathcal{U}_{\alpha,\beta}(-\mathrm{y}) = 0\}$$

This Work in a nutshell:

Improving Curve Trees and specializing it to the batch setting

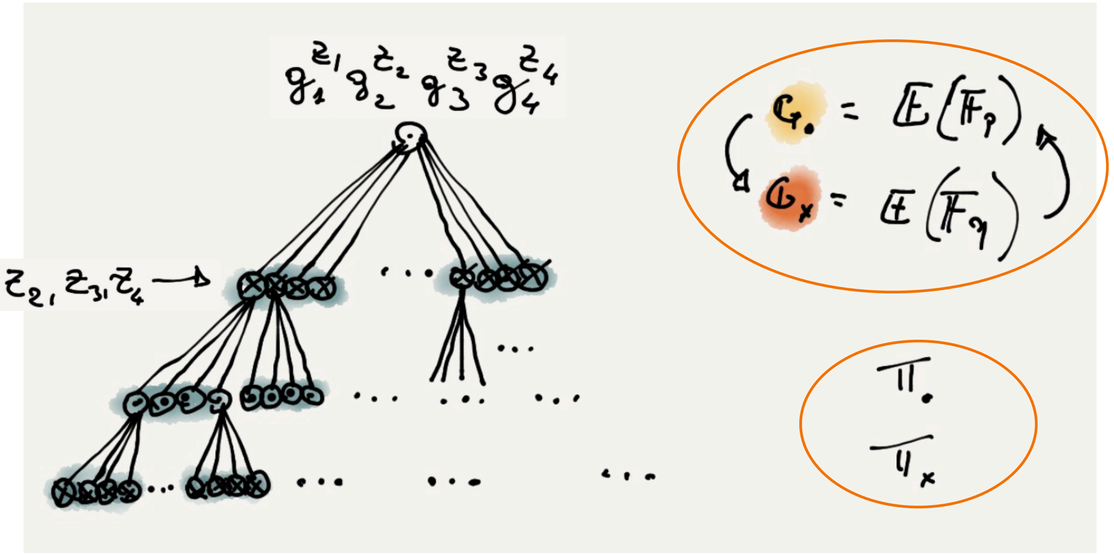
(defined in the next slide)

Curve Trees are a natural choice:

1. “sandbox for algebraic tricks”

   CurveTree ≈ n-ary Merkle Tree (with specifics)
   + Pedersen Hash
   + cycle of curves
   + [tricks]

2. real-world impact



A Curve Tree (simplified)

$$\mathcal{U}_{\alpha,\beta}(v) : \mathbb{F} \to \{0,1\} \quad \mathcal{U}_{\alpha,\beta}(v) \mapsto S(\alpha \cdot v + \beta)$$

$$\mathcal{P}_{\mathbb{E}} = \{(\mathrm{x},\mathrm{y}) \mid (\mathrm{x},\mathrm{y}) \in \mathbb{E}(\mathbb{F}_p) \wedge \mathcal{U}_{\alpha,\beta}(\mathrm{y}) = 1 \wedge \mathcal{U}_{\alpha,\beta}(-\mathrm{y}) = 0\}$$

This Work in a nutshell:

Improving Curve Trees and specializing it to the batch setting

(defined in the next slide)

Curve Trees are a natural choice:

1. “sandbox for algebraic tricks”

   CurveTree ≈ n-ary Merkle Tree (with specifics)
   + Pedersen Hash
   + cycle of curves
   + [tricks]

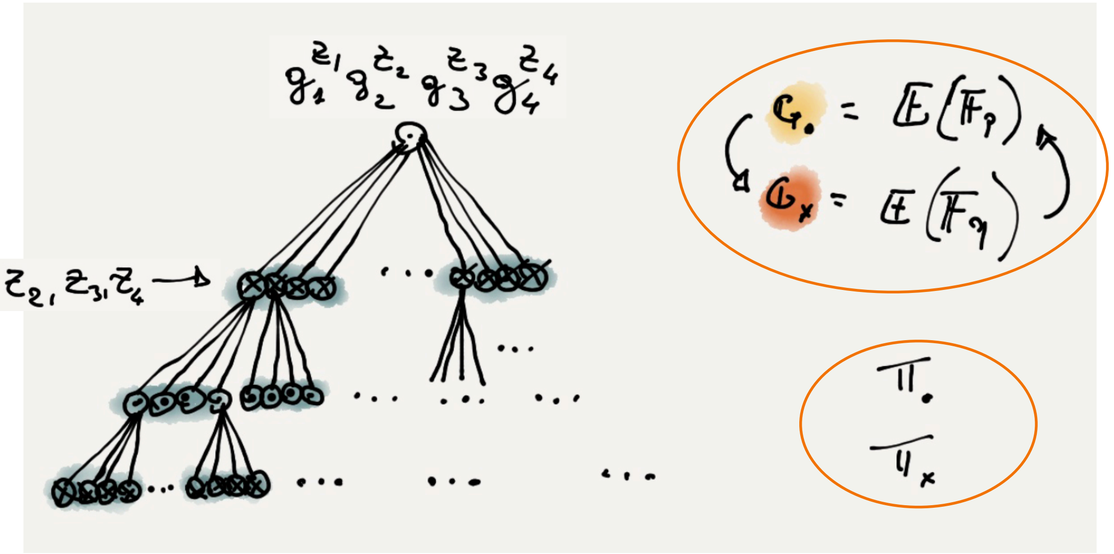
2. real-world impact

Adopting Curve Trees*:

(imminently?)

firo (soon)

*(NB: none of the authors is involved with these efforts—we are aware from public info)



A Curve Tree (simplified)

$$\mathcal{U}_{\alpha,\beta}(v) : \mathbb{F} \to \{0,1\} \quad \mathcal{U}_{\alpha,\beta}(v) \mapsto S(\alpha \cdot v + \beta)$$

$$\mathcal{P}_{\mathbb{E}} = \{(\mathrm{x},\mathrm{y}) \mid (\mathrm{x},\mathrm{y}) \in \mathbb{E}(\mathbb{F}_p) \wedge \mathcal{U}_{\alpha,\beta}(\mathrm{y}) = 1 \wedge \mathcal{U}_{\alpha,\beta}(-\mathrm{y}) = 0\}$$

This Work in a nutshell:

Improving Curve Trees and specializing it to the batch setting

(defined in the next slide)

Curve Trees are a natural choice:

1. "sandbox for algebraic tricks"

   CurveTree ≈ n-ary Merkle Tree (with specifics)
   + Pedersen Hash
   + cycle of curves
   + [tricks]

2. real-world impact

   Adopting Curve Trees*:

   (imminently?) (soon)

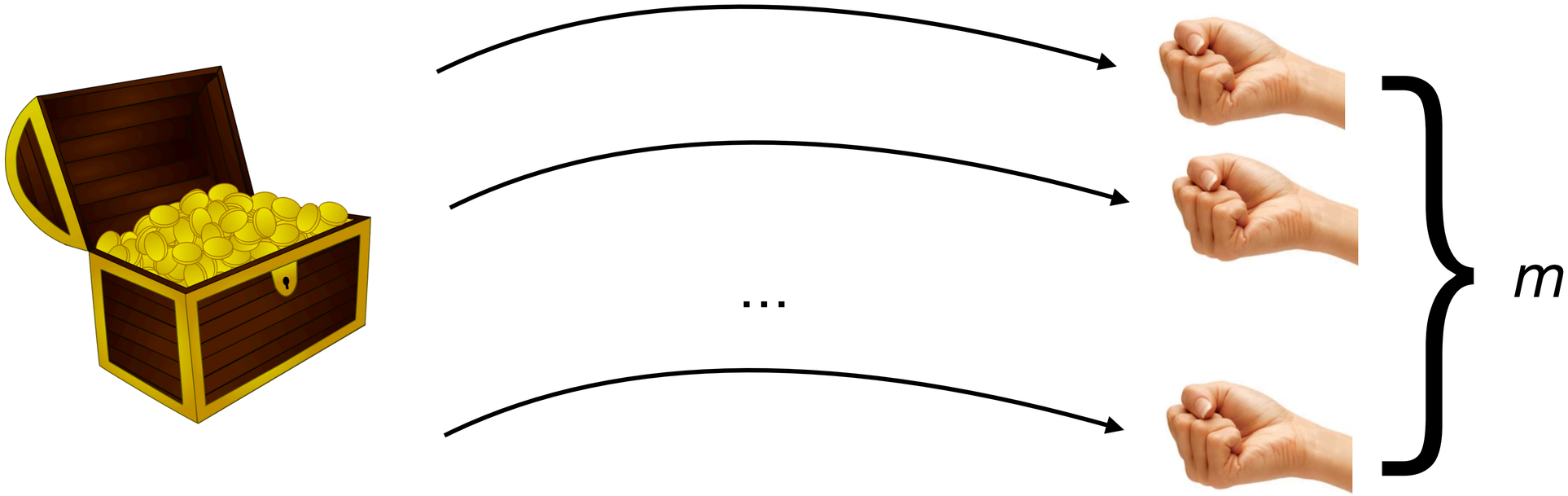
*(NB: none of the authors is involved with these efforts—we are aware from public info)

ZK Set Membership with Batch Proving

Our more specific setting

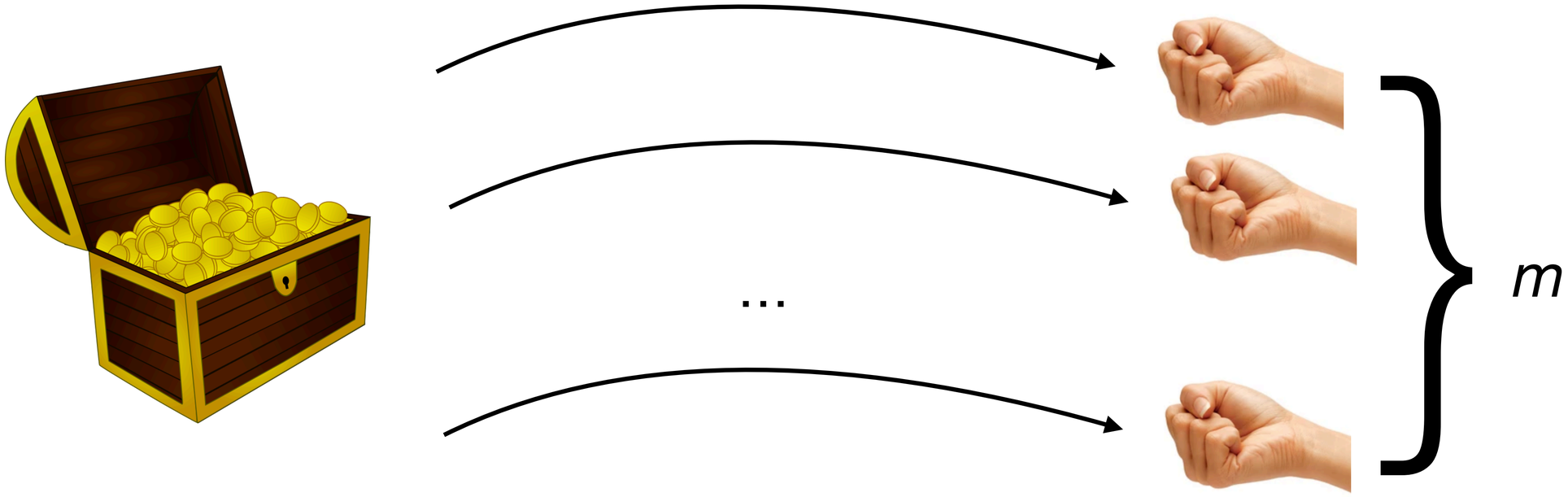
ZK Set Membership with Batch Proving

Our more specific setting



ZK Set Membership with Batch Proving

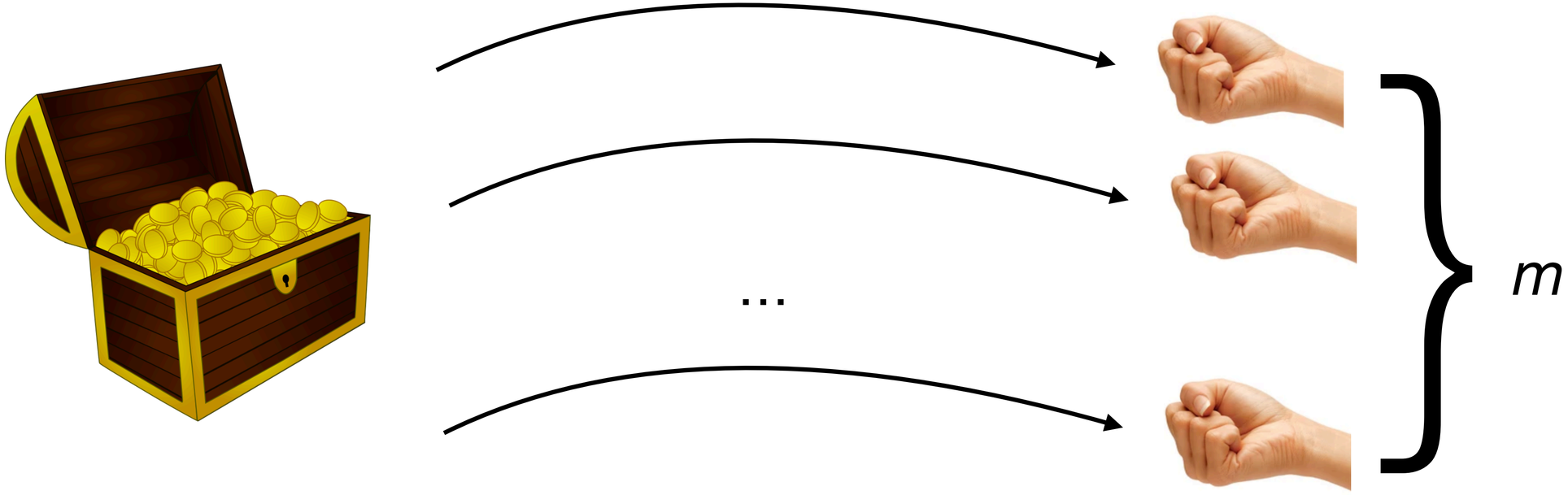
Our more specific setting



Verify($\mathsf{dig}_S, \mathsf{cm}_1, \ldots, \mathsf{cm}_m, \mathsf{prf}_{\mathsf{batch}}$)

ZK Set Membership with Batch Proving

Our more specific setting

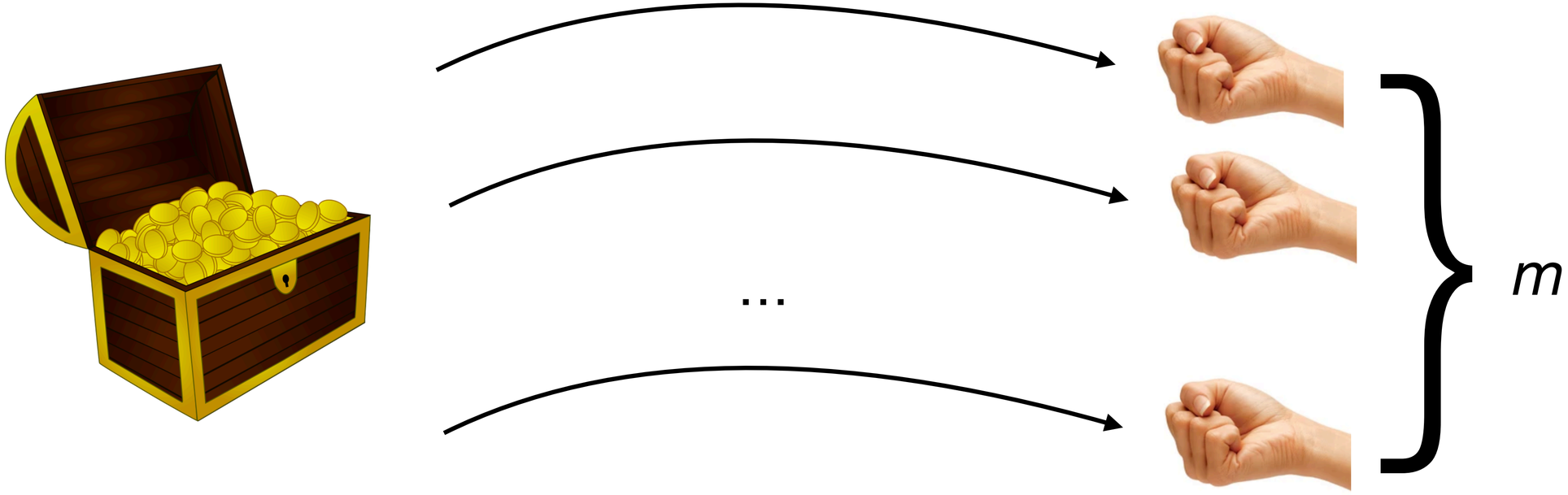


Key goal: amortization

$\mathsf{Verify}(\mathsf{dig}_S, \mathsf{cm}_1, \ldots, \mathsf{cm}_m, \mathsf{prf}_{\mathsf{batch}})$

ZK Set Membership with Batch Proving

Our more specific setting



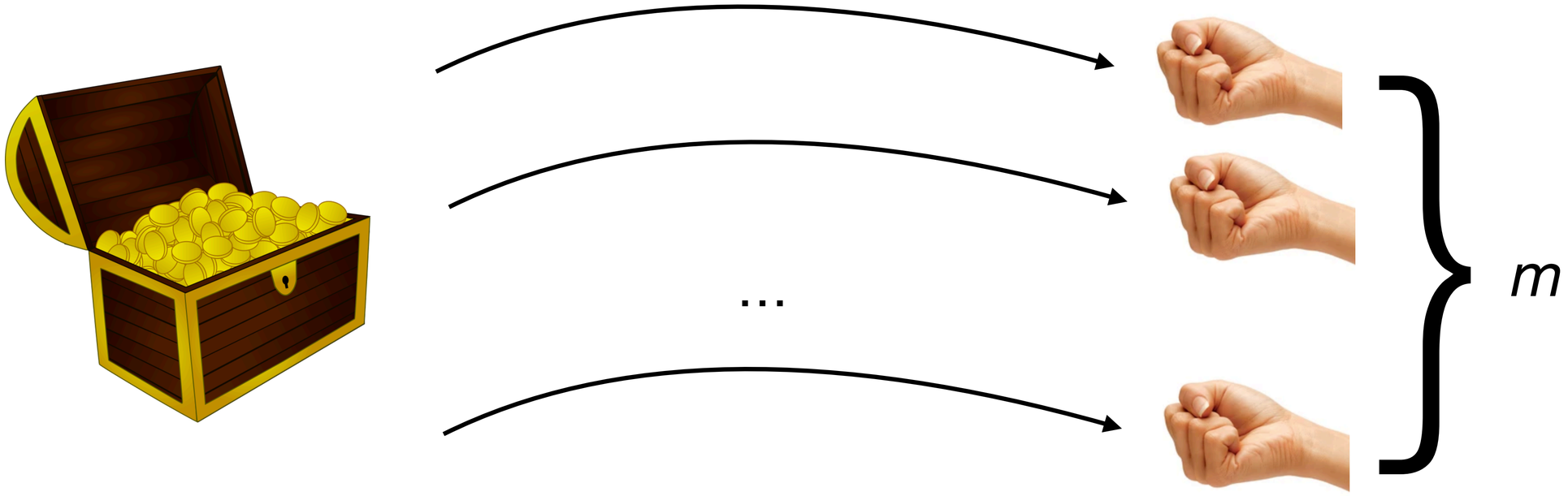
$\mathsf{Verify}(\mathsf{dig}_S, \mathsf{cm}_1, \ldots, \mathsf{cm}_m, \mathsf{prf}_{\mathsf{batch}})$

Key goal: amortization

$$T(\mathsf{Prove}_{\mathsf{batch}}) < m \cdot T(\mathsf{Prove}_{\mathsf{single}})$$

ZK Set Membership with Batch Proving

Our more specific setting



Verify($\mathsf{dig}_S, \mathsf{cm}_1, \ldots, \mathsf{cm}_m, \mathsf{prf}_{\mathsf{batch}}$)

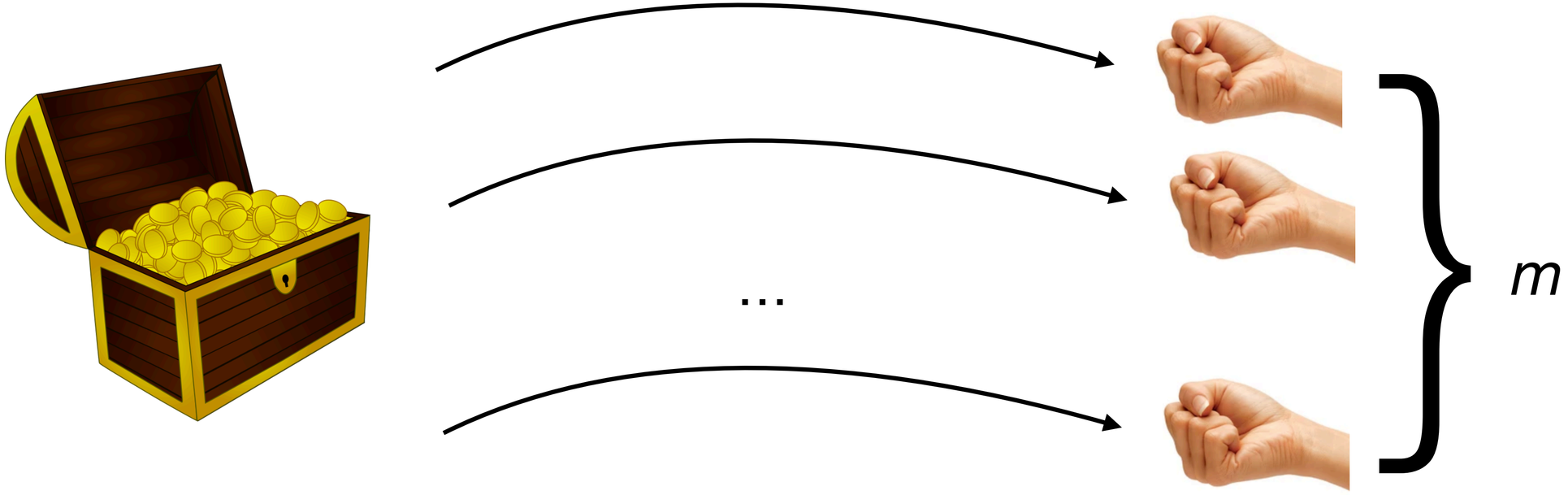
Key goal: amortization

$T(\mathsf{Prove}_{\mathsf{batch}}) < m \cdot T(\mathsf{Prove}_{\mathsf{single}})$

+ similar amortizations
for verification and proof size

ZK Set Membership with Batch Proving

Our more specific setting



Key goal: amortization

$$T(\text{Prove}_{\text{batch}}) < m \cdot T(\text{Prove}_{\text{single}})$$

+ similar amortizations
for verification and proof size

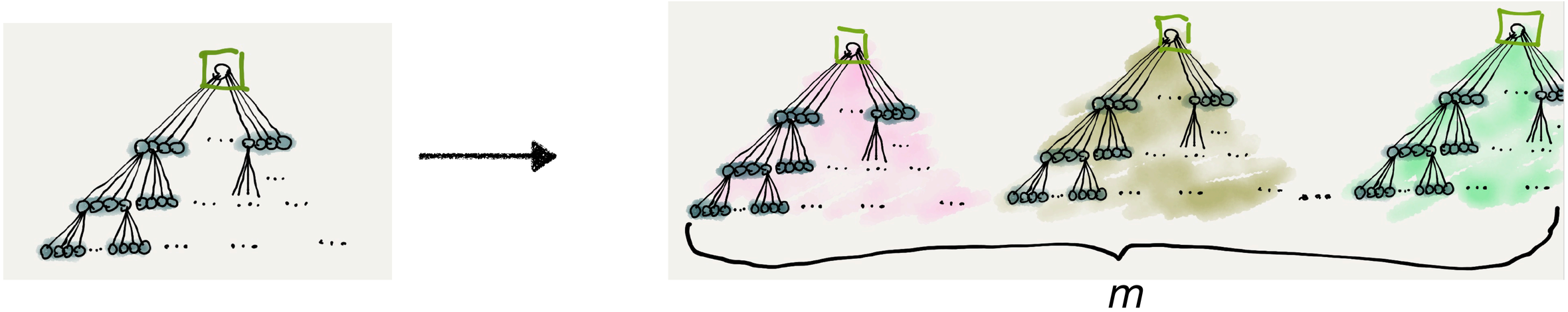
$$\text{Verify}(\text{dig}_S, \text{cm}_1, \ldots, \text{cm}_m, \text{prf}_{\text{batch}})$$

Applications?
Same as before
(e.g., proving *multiple* tx-s or identity features *at the same time*)
+ more

Our Contributions

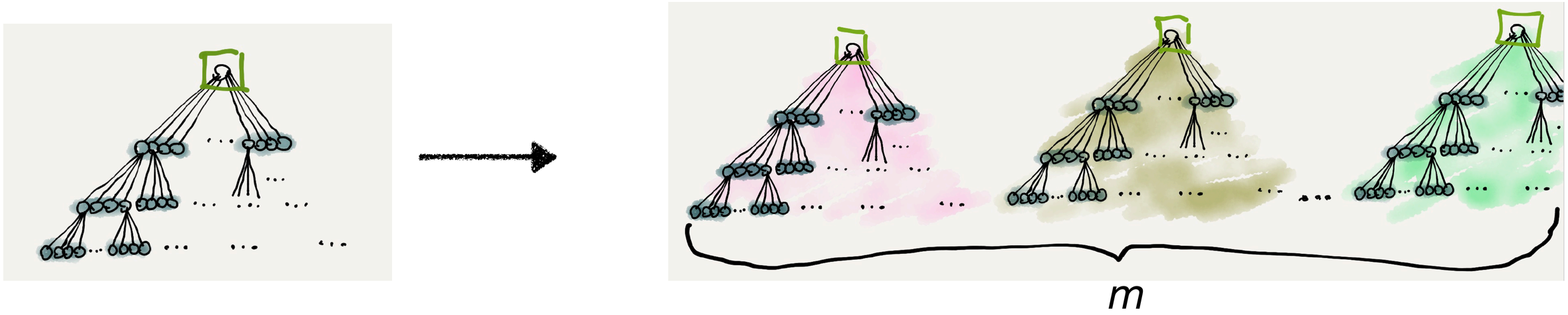
A New Construction for Batch Setting—Curve Forests

Contribution 1



A New Construction for Batch Setting—Curve Forests

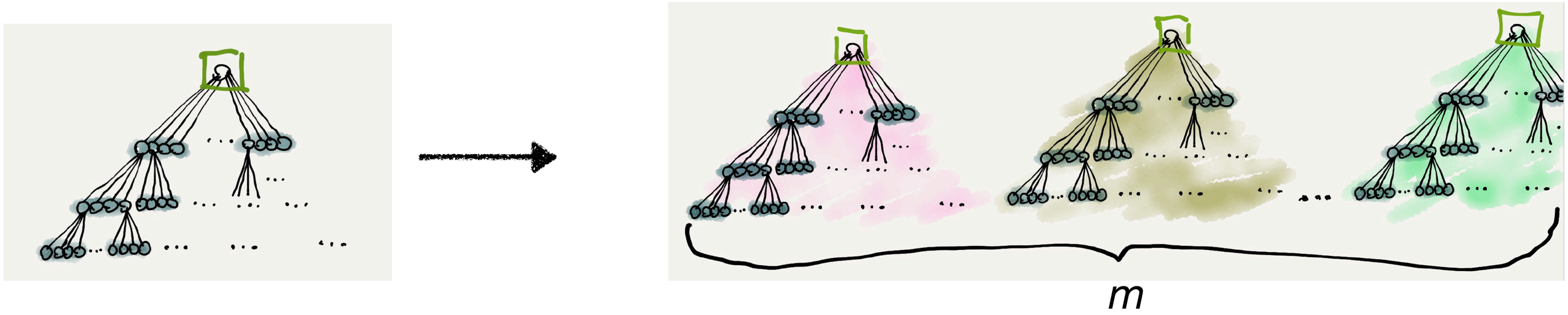
Contribution 1



- **Concretely efficient** (results in a couple slides)

A New Construction for Batch Setting—Curve Forests

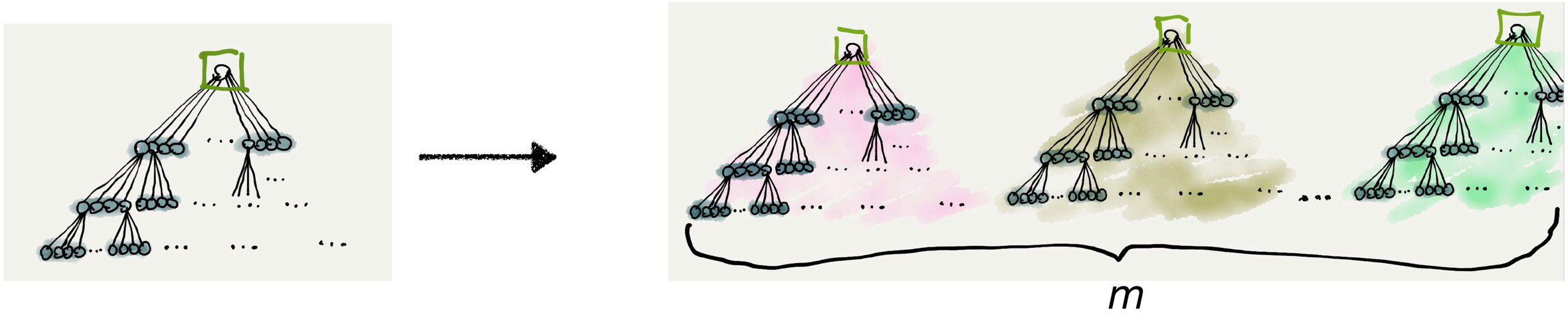
Contribution 1



- **Concretely efficient** (results in a couple slides)
- **Transparent setup**

A New Construction for Batch Setting—Curve Forests

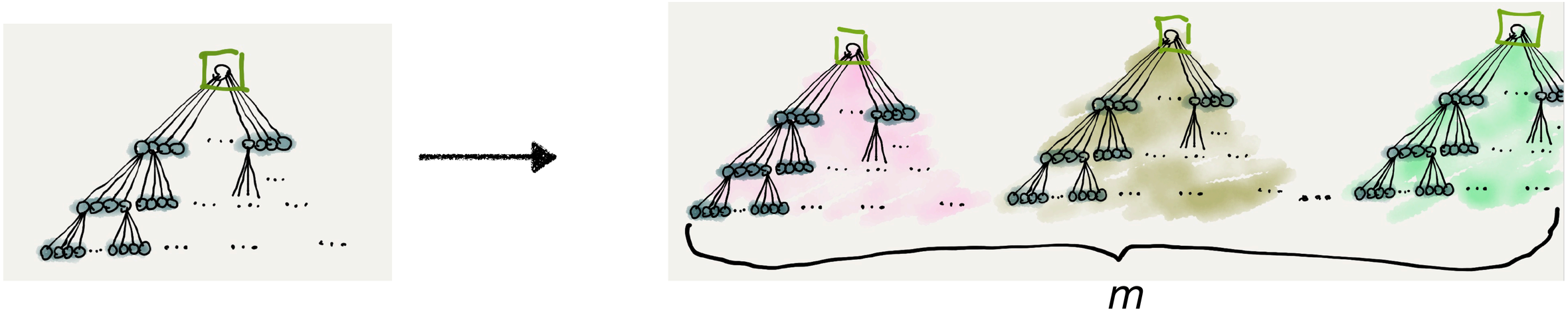
Contribution 1



- **Concretely efficient** (results in a couple slides)
- **Transparent setup**
- **Amortization through redundant state of size *m* (batch size)**

A New Construction for Batch Setting—Curve Forests

Contribution 1



- **Concretely efficient** (results in a couple slides)
- **Transparent setup**
- **Amortization through redundant state of size *m* (batch size)**
  - NB: inserting one element is still O(1) in communication! (see paper)

Strengthened Security (+ Simplified Protocol) in *Curve* Botany

(**i.e., in** Trees,Forests, etc)

Contribution 2

Strengthened Security (+ Simplified Protocol) in *Curve* Botany

(**i.e., in** Trees,Forests, etc)

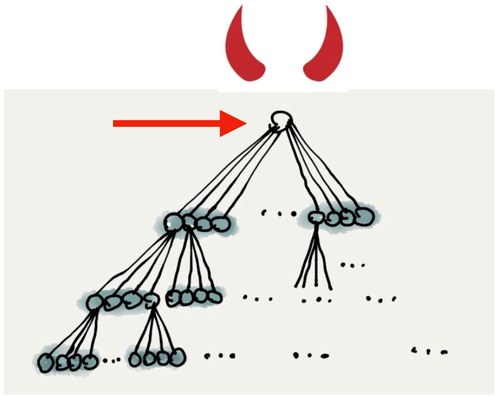
Contribution 2

“Strengthened Security”?

Strengthened Security (+ Simplified Protocol) in *Curve* Botany

(**i.e., in** Trees,Forests, etc)

Contribution 2



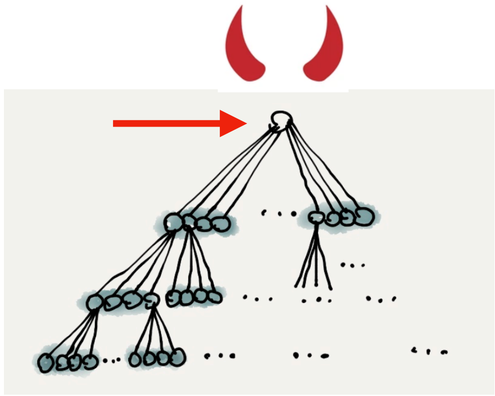
Strengthening 1:
Support *maliciously*-provided set commitments
(applications? see final slide)

"Strengthened Security"?

Strengthened Security (+ Simplified Protocol) in *Curve* Botany

(**i.e., in** Trees,Forests, etc)

Contribution 2



Strengthening 1:
Support *maliciously*-provided set commitments
(applications? see final slide)

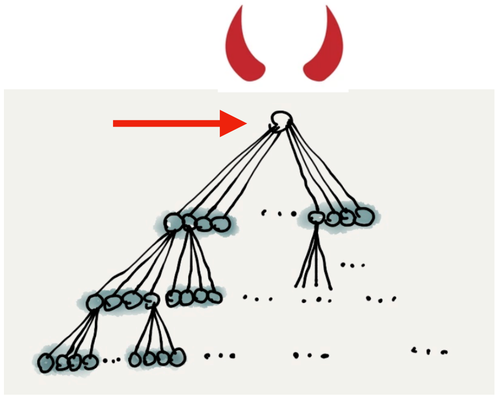
"Strengthened Security"?

From a formal point of view
weak binding → *extractability*

Strengthened Security (+ Simplified Protocol) in *Curve* Botany

(**i.e., in** Trees,Forests, etc)

Contribution 2



"Strengthened Security"?

Strengthening 1:
Support *maliciously*-provided set commitments
(applications? see final slide)

→ From a formal point of view *weak binding* → *extractability*

Strengthening 2:
Prevent a form of "*grieving*" *attacks*
(see Section 3 in paper for more)

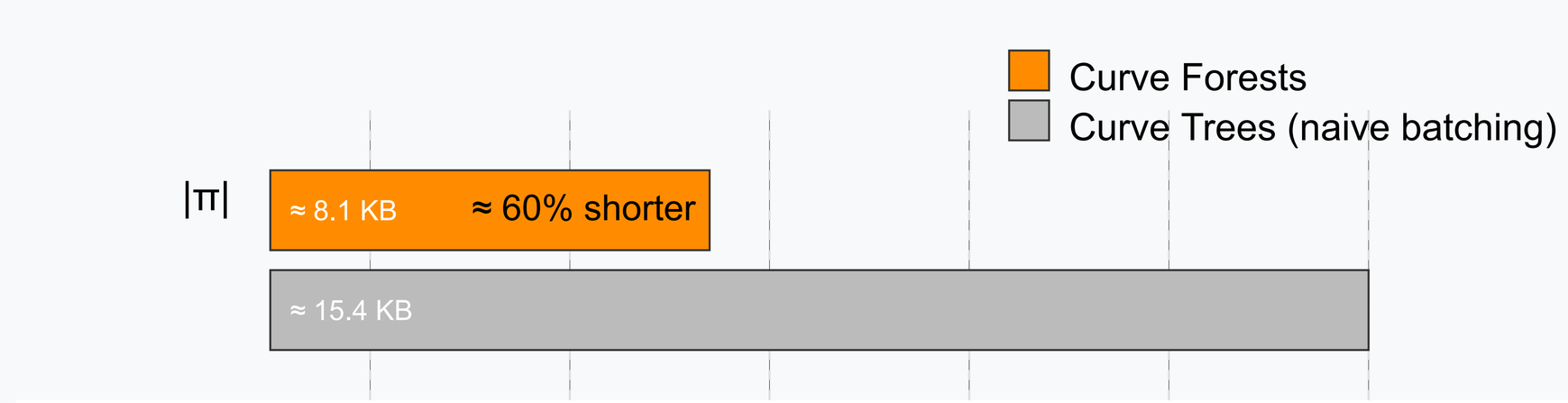
Results: Improvements for Batching

Notes:
Results in figure are for
a *small* batch (*m=8*).

[Speedups should be better
for larger batches.

Also, we expect
an *extra* 2X improvement
from low-hanging fruit
optimizations
(on top of the above)]

Results: Improvements for Batching



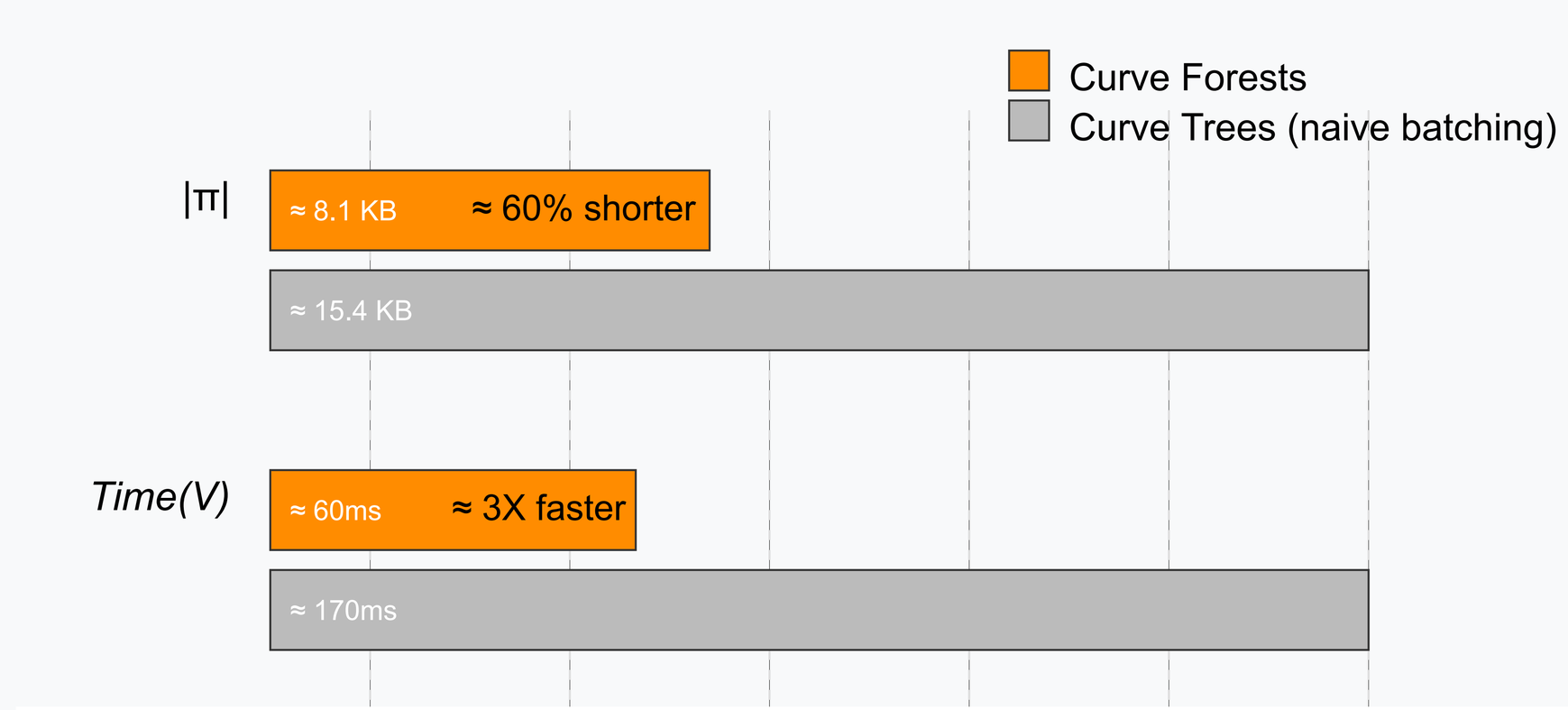
Notes:
Results in figure are for a *small* batch (*m*=*8*).

[Speedups should be better for larger batches.

Also, we expect an *extra* 2X improvement from low-hanging fruit optimizations (on top of the above)]

[Benchmarks run on a common laptop (M2 Pro, 16GB RAM)]

Results: Improvements for Batching



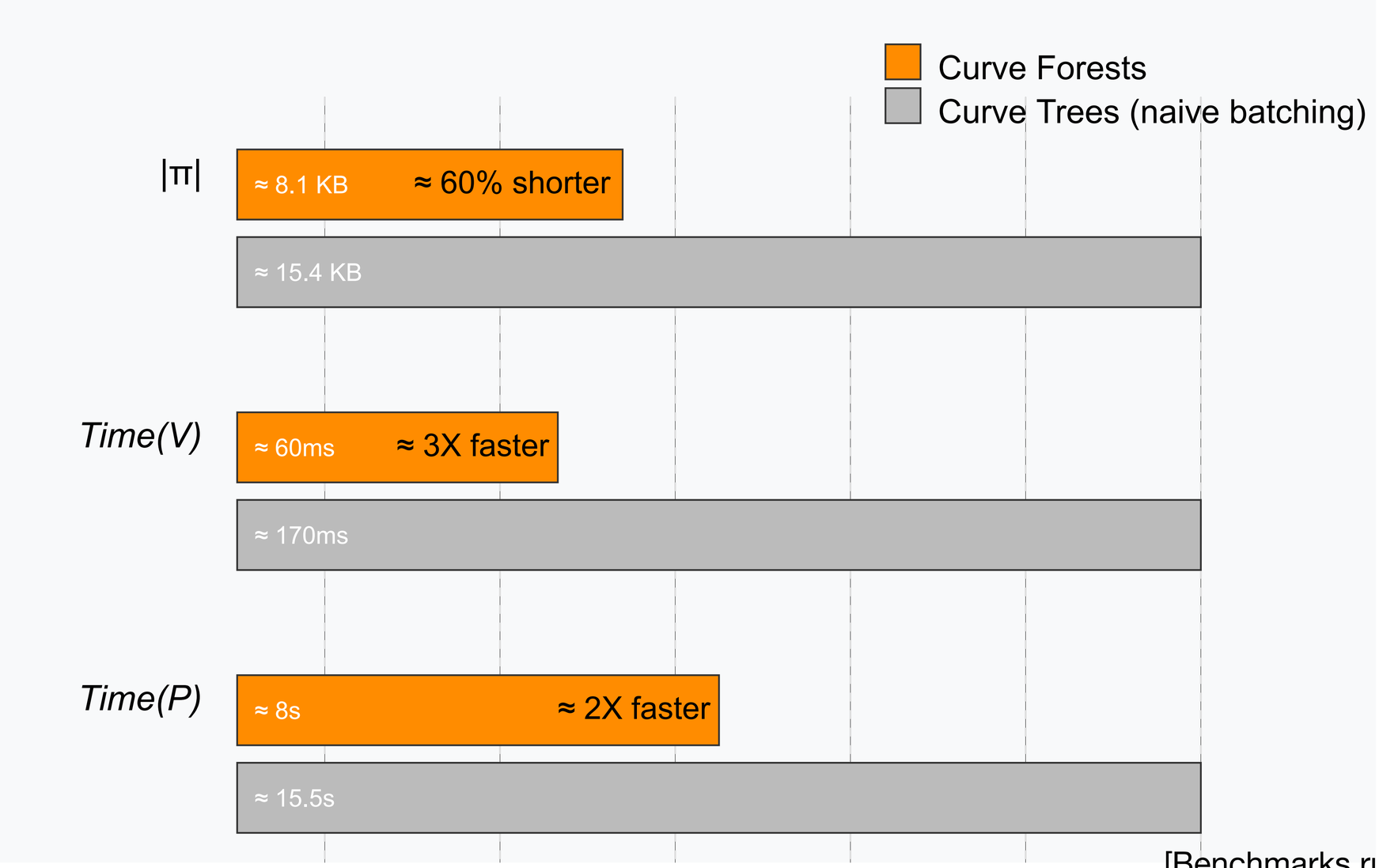
Notes:
Results in figure are for a *small* batch (*m=8*).

[Speedups should be better for larger batches.

Also, we expect an *extra* 2X improvement from low-hanging fruit optimizations (on top of the above)]

[Benchmarks run on a common laptop (M2 Pro, 16GB RAM)]

Results: Improvements for Batching



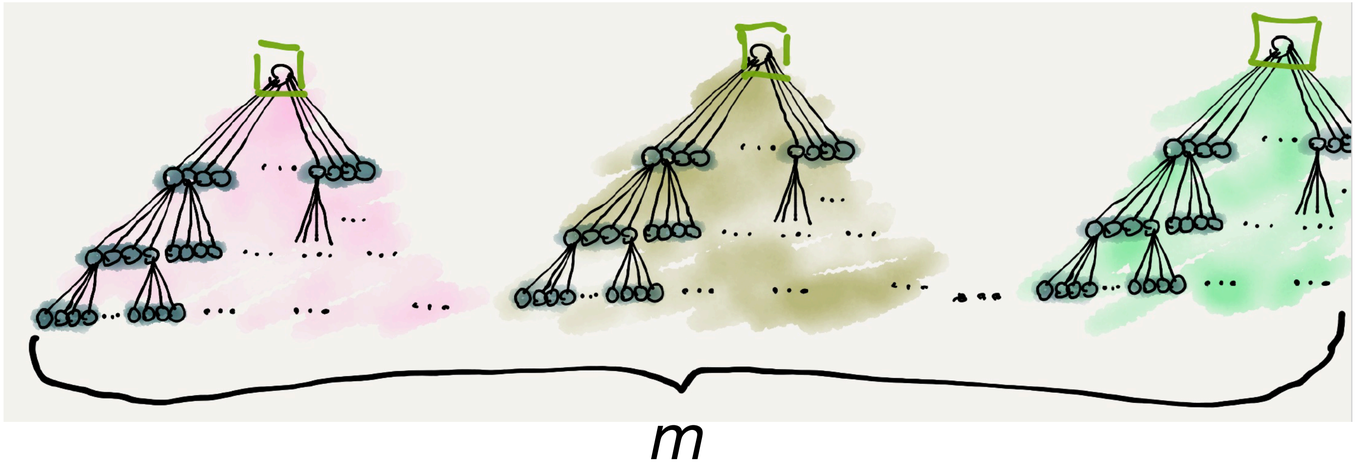
Notes:
Results in figure are for a *small* batch (*m=8*).

[Speedups should be better for larger batches.

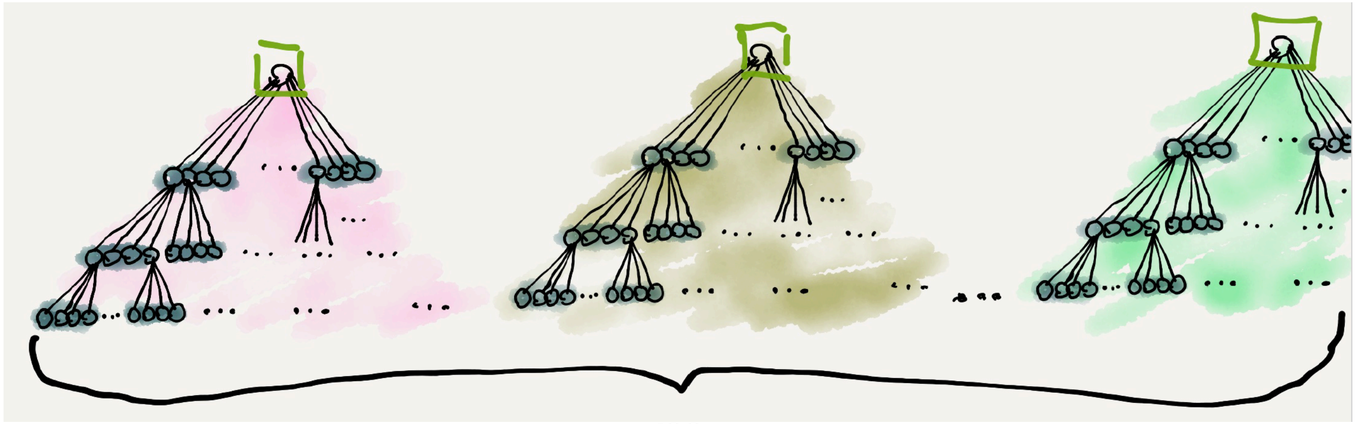
Also, we expect an *extra* 2X improvement from low-hanging fruit optimizations (on top of the above)]

[Benchmarks run on a common laptop (M2 Pro, 16GB RAM)]

Batching Techniques in Curve Forests (peep only)

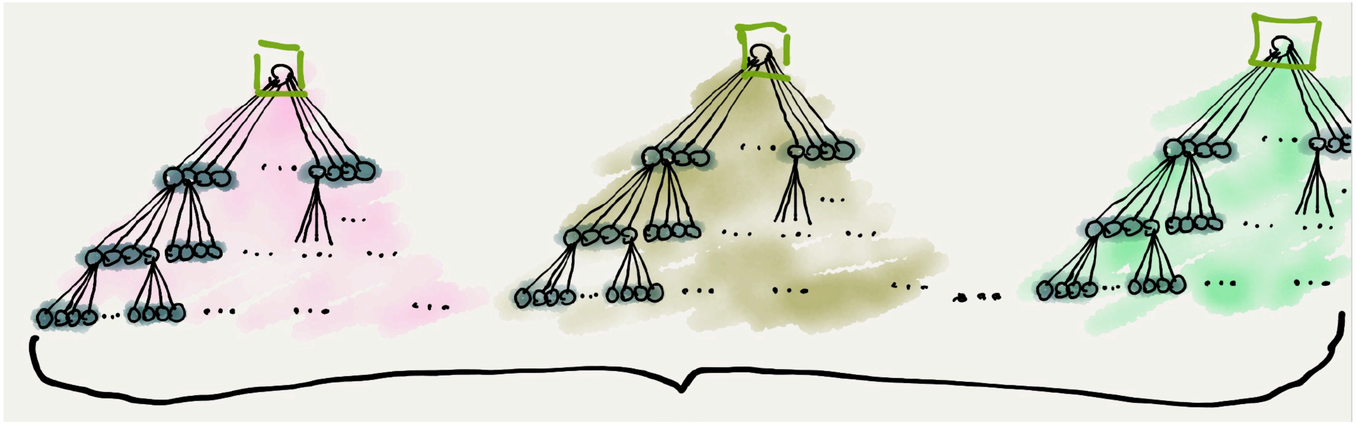


Batching Techniques in Curve Forests (peep only)



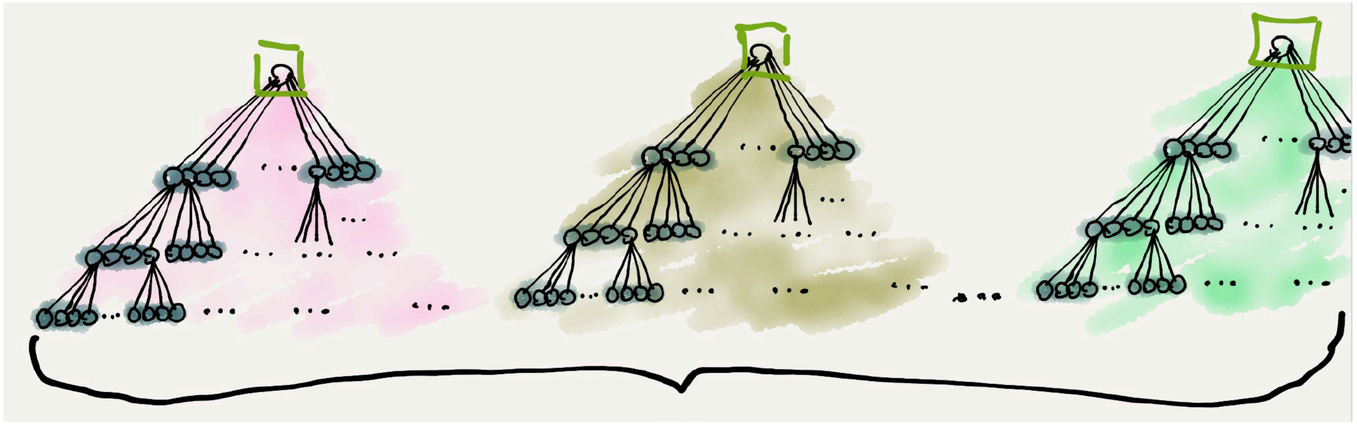
- **Amortization through *forestation***

Batching Techniques in Curve Forests (peep only)



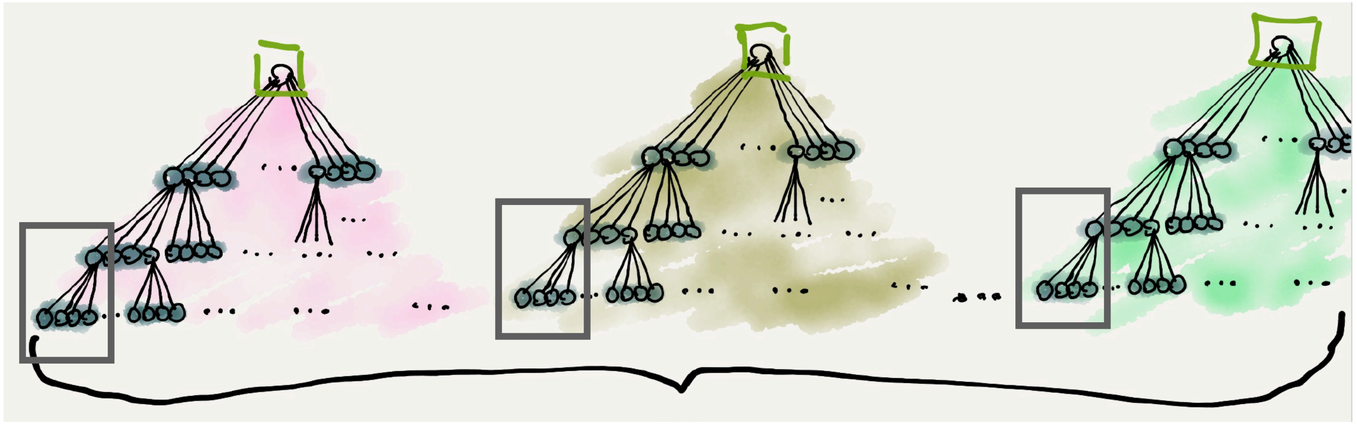
- **Amortization through ***forestation***
  - Forest = *redundant representation with distinct sets of Pedersen generators*

Batching Techniques in Curve Forests (peep only)



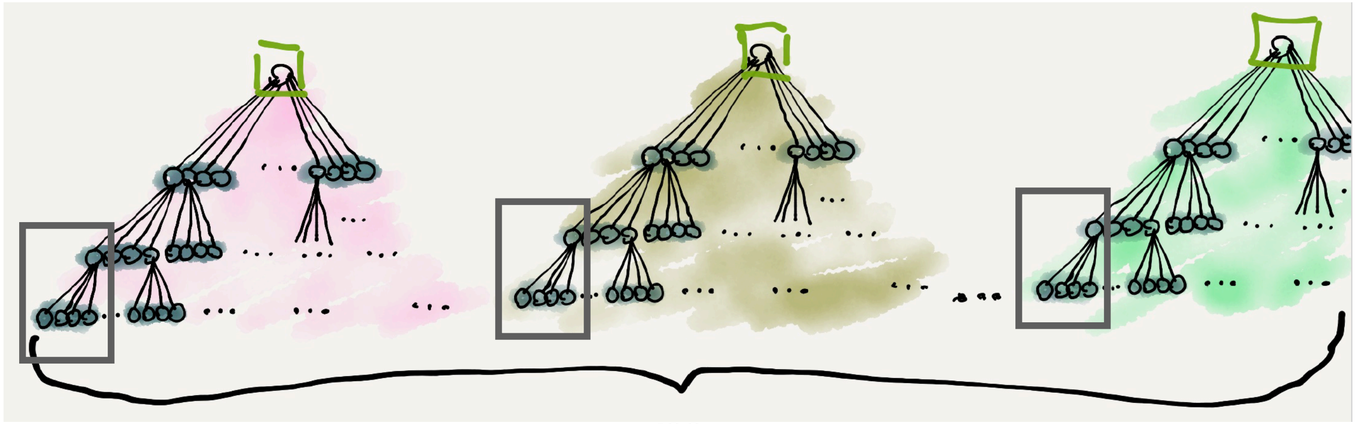
- **Amortization through *<u>forestation</u>***
  - Forest = *redundant representation with distinct sets of Pedersen generators*
  - After this, the magical *random linear combinations* fairy does the heavy lifting.

Batching Techniques in Curve Forests (peep only)



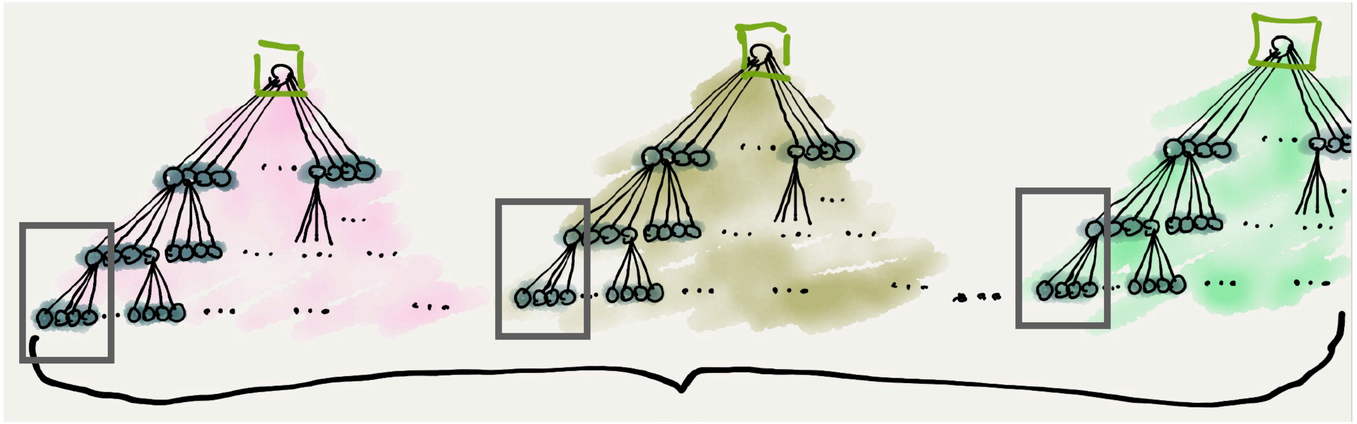
- **Amortization through *forestation***
  - Forest = *redundant representation with distinct sets of Pedersen generators*
  - After this, the magical *random linear combinations* fairy does the heavy lifting.

Batching Techniques in Curve Forests (peep only)



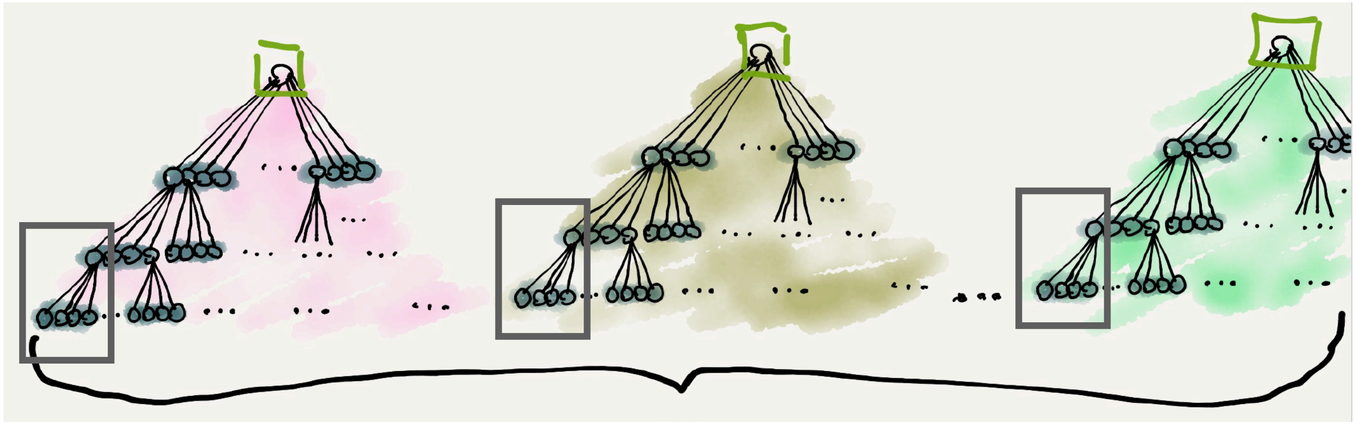
- **Amortization through *forestation***
  - Forest = *redundant representation with distinct sets of Pedersen generators*
  - After this, the magical *random linear combinations* fairy does the heavy lifting.
- **How much do you amortize?**

Batching Techniques in Curve Forests (peep only)



- **Amortization through *forestation***
  - Forest = *redundant representation with distinct sets of Pedersen generators*
  - After this, the magical *random linear combinations* fairy does the heavy lifting.
- **How much do you amortize?**
  - **Premise:** basic cost unit for proving (rectangles above) consist of two steps:

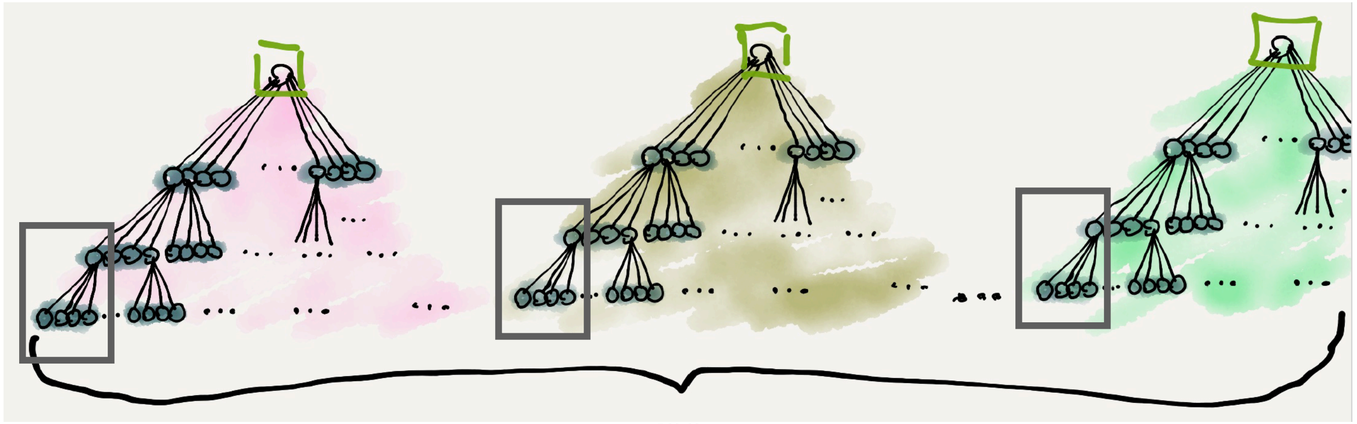
Batching Techniques in Curve Forests (peep only)



- **Amortization through *forestation***
  - Forest = *redundant representation with distinct sets of Pedersen generators*
  - After this, the magical *random linear combinations* fairy does the heavy lifting.
- **How much do you amortize?**
  - **Premise:** basic cost unit for proving (rectangles above) consist of two steps:

    “Select” (**cheap**) + “Rerandomize” (**costly**)

Batching Techniques in Curve Forests (peep only)



- **Amortization through *forestation***
  - Forest = *redundant representation with distinct sets of Pedersen generators*
  - After this, the magical *random linear combinations* fairy does the heavy lifting.
- **How much do you amortize?**
  - **Premise:** basic cost unit for proving (rectangles above) consist of two steps:

    “Select” (**cheap**) + “Rerandomize” (**costly**)

In Curve Forests: m Rerandomize ops → 1 Rerandomize op

That’s it!

Web/contact: binarywhales.com

(full version here)

That’s it!

(But wait! What would one look at next on this topic?)

Web/contact: binarywhales.com

(full version here)

That’s it!

(But wait! What would one look at next on this topic?)

- **This work:** “Simple (well aimed) techniques already give significant benefits (batching and general case)”.

Web/contact: binarywhales.com

(full version here)

That’s it!

(But wait! What would one look at next on this topic?)

- **This work:** “Simple (well aimed) techniques already give significant benefits (batching and general case)”.
- **Future work?**

Web/contact: binarywhales.com

(full version here)

That’s it!

(But wait! What would one look at next on this topic?)

- **This work:** “Simple (well aimed) techniques already give significant benefits (batching and general case)”.
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**

Web/contact: binarywhales.com

(full version here)

That’s it!

(But wait! What would one look at next on this topic?)

- **This work:** “Simple (well aimed) techniques already give significant benefits (batching and general case)”.
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**
    - (we have some failed attempts that might work in an idealized model we do not trust. Maybe ~~second~~ n-th look there?)

Web/contact: binarywhales.com

(full version here)

That’s it!

(But wait! What would one look at next on this topic?)

- **This work:** “Simple (well aimed) techniques already give significant benefits (batching and general case)”.
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**
    - (we have some failed attempts that might work in an idealized model we do not trust. Maybe ~~second~~ n-th look there?)
  - **Towards zkVMs from Curve Trees? Maybe:**

Web/contact: binarywhales.com

(full version here)

That’s it!

(But wait! What would one look at next on this topic?)

- **This work:** “Simple (well aimed) techniques already give significant benefits (batching and general case)”.
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**
    - (we have some failed attempts that might work in an idealized model we do not trust. Maybe ~~second~~ n-th look there?)
  - **Towards zkVMs from Curve Trees? Maybe:**
    - **Curve Forests (**with the extractability security we have**) → CurveTreeish Lookups (?)**

Web/contact: binarywhales.com

(full version here)

That’s it!

(But wait! What would one look at next on this topic?)

- **This work:** “Simple (well aimed) techniques already give significant benefits (batching and general case)”.
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**
    - (we have some failed attempts that might work in an idealized model we do not trust. Maybe ~~second~~ n-th look there?)
  - **Towards zkVMs from Curve Trees? Maybe:**
    - **Curve Forests (**with the extractability security we have**)** → **CurveTreeish Lookups (?)**
    - **CurveTree-ish Lookups to instantiate Jolt-ish** [**C**FR25] (NB: non verifier succinct)

Web/contact: binarywhales.com

(full version here)

That’s it!

(But wait! What would one look at next on this topic?)

- **This work:** “Simple (well aimed) techniques already give significant benefits (batching and general case)”.
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**
    - (we have some failed attempts that might work in an idealized model we do not trust. Maybe ~~second~~ n-th look there?)
  - **Towards zkVMs from Curve Trees? Maybe:**
    - **Curve Forests (**with the extractability security we have**)** → **CurveTreeish Lookups (?)**
    - **CurveTree-ish Lookups to instantiate Jolt-ish** [**C**FR25] (NB: non verifier succinct)
      - (non malleability for free from same paper together with BP)

Web/contact: binarywhales.com

(full version here)

That’s it!

(But wait! What would one look at next on this topic?)

- **This work:** “Simple (well aimed) techniques already give significant benefits (batching and general case)”.
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**
    - (we have some failed attempts that might work in an idealized model we do not trust. Maybe ~~second~~ n-th look there?)
  - **Towards zkVMs from Curve Trees? Maybe:**
    - **Curve Forests (**with the extractability security we have**) → CurveTreeish Lookups (?)**
    - **CurveTree-ish Lookups to instantiate Jolt-ish** [**C**FR25] (NB: non verifier succinct)
      - (non malleability for free from same paper together with BP)
  - **Applications to ZK Decision Trees** (low hanging fruit—but might require prev bullet)**:**

Web/contact: binarywhales.com

(full version here)

That’s it!

(But wait! What would one look at next on this topic?)

- **This work:** “Simple (well aimed) techniques already give significant benefits (batching and general case)”.
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**
    - (we have some failed attempts that might work in an idealized model we do not trust. Maybe ~~second~~ n-th look there?)
  - **Towards zkVMs from Curve Trees? Maybe:**
    - **Curve Forests (**with the extractability security we have**) → CurveTreeish Lookups (?)**
    - **CurveTree-ish Lookups to instantiate Jolt-ish** [**C**FR25] (NB: non verifier succinct)
      - (non malleability for free from same paper together with BP)
  - **Applications to ZK Decision Trees** (low hanging fruit—but might require prev bullet)**:**
    - applying it to the techniques from [**C**FF+24]

Web/contact: binarywhales.com

(full version here)

That's it!

(But wait! What would one look at next on this topic?)

- **This work:** "Simple (well aimed) techniques already give significant benefits (batching and general case)".
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**
    - (we have some failed attempts that might work in an idealized model we do not trust. Maybe ~~second~~ n-th look there?)
  - **Towards zkVMs from Curve Trees? Maybe:**
    - **Curve Forests (**with the extractability security we have**) → CurveTreeish Lookups (?)**
    - **CurveTree-ish Lookups to instantiate Jolt-ish** [**C**FR25] (NB: non verifier succinct)
      - (non malleability for free from same paper together with BP)
  - **Applications to ZK Decision Trees** (low hanging fruit—but might require prev bullet)**:**
    - applying it to the techniques from [**C**FF+24]
  - **Other applications:** DAOs proving they follow their own rules in ZK (ask me later if interested).

Web/contact: binarywhales.com

(full version here)

That's it!

(But wait! What would one look at next on this topic?)

- **This work:** "Simple (well aimed) techniques already give significant benefits (batching and general case)".
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**
    - (we have some failed attempts that might work in an idealized model we do not trust. Maybe ~~second~~ n-th look there?)
  - **Towards zkVMs from Curve Trees? Maybe:**
    - **Curve Forests (**with the extractability security we have**) → CurveTreeish Lookups (?)**
    - **CurveTree-ish Lookups to instantiate Jolt-ish** [**C**FR25] (NB: non verifier succinct)
      - (non malleability for free from same paper together with BP)
  - **Applications to ZK Decision Trees** (low hanging fruit—but might require prev bullet)**:**
    - applying it to the techniques from [**C**FF+24]
  - **Other applications:** DAOs proving they follow their own rules in ZK (ask me later if interested).

[CFR25]: SNARKs for Virtual Machines are non-malleable (EUROCRYPT25, to appear)

(full version here)

Web/contact: binarywhales.com

That's it!

(But wait! What would one look at next on this topic?)

- **This work:** "Simple (well aimed) techniques already give significant benefits (batching and general case)".
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**
    - (we have some failed attempts that might work in an idealized model we do not trust. Maybe ~~second~~ n-th look there?)
  - **Towards zkVMs from Curve Trees? Maybe:**
    - **Curve Forests (**with the extractability security we have**) → CurveTreeish Lookups (?)**
    - **CurveTree-ish Lookups to instantiate Jolt-ish** [**C**FR25] (NB: non verifier succinct)
      - (non malleability for free from same paper together with BP)
  - **Applications to ZK Decision Trees** (low hanging fruit—but might require prev bullet)**:**
    - applying it to the techniques from [**C**FF+24]
  - **Other applications:** DAOs proving they follow their own rules in ZK (ask me later if interested).

[CFR25]: SNARKs for Virtual Machines are non-malleable (EUROCRYPT25, to appear)

[CFF+24]: Lookup Arguments: Improvements, Extensions and Applications to Zero-Knowledge Decision Trees (PKC24)

Web/contact: binarywhales.com

(full version here)

That's it!

(But wait! What would one look at next on this topic?)

Thanks!

- **This work:** "Simple (well aimed) techniques already give significant benefits (batching and general case)".
- **Future work?**
  - **More aggressive techniques to amortize everything (beyond Rerandomize)?**
    - (we have some failed attempts that might work in an idealized model we do not trust. Maybe ~~second~~ n-th look there?)
  - **Towards zkVMs from Curve Trees? Maybe:**
    - **Curve Forests (**with the extractability security we have**) → CurveTreeish Lookups (?)**
    - **CurveTree-ish Lookups to instantiate Jolt-ish** [**C**FR25] (NB: non verifier succinct)
      - (non malleability for free from same paper together with BP)
  - **Applications to ZK Decision Trees** (low hanging fruit—but might require prev bullet)**:**
    - applying it to the techniques from [**C**FF+24]
  - **Other applications:** DAOs proving they follow their own rules in ZK (ask me later if interested).

[CFR25]: SNARKs for Virtual Machines are non-malleable (EUROCRYPT25, to appear)

[CFF+24]: Lookup Arguments: Improvements, Extensions and Applications to Zero-Knowledge Decision Trees (PKC24)

Web/contact: binarywhales.com

(full version here)