NETHERMIND RESEARCH

# Nova-style folding with STARK-friendly primes

Albert Garreta

# Folding

# Folding

- Fix a relation $R$, consisting of pairs $(x; w)$

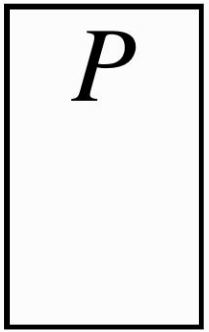
# Folding

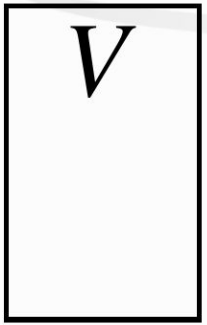
- Fix a relation $R$, consisting of pairs $(x; w)$
- $x$ is a public instance, $w$ is a witness

# Folding

- Fix a relation $R$, consisting of pairs $(x; w)$
- $x$ is a public instance, $w$ is a witness

A folding scheme is an interactive protocol between P and V where:





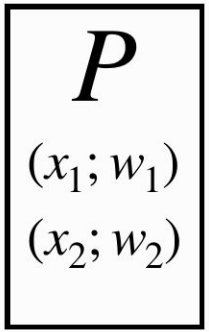

NETHERMIND RESEARCH

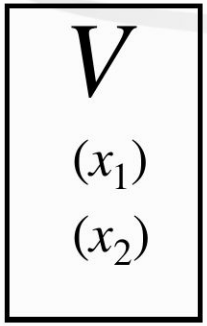
# Folding

- Fix a relation $R$, consisting of pairs $(x; w)$
- $x$ is a public instance, $w$ is a witness

A folding scheme is an interactive protocol between P and V where:

- P and V have two instances $x_1, x_2$.
- P also has witnesses $w_1, w_2$ such that $(x_1; w_1), (x_2; w_2) \in R$.





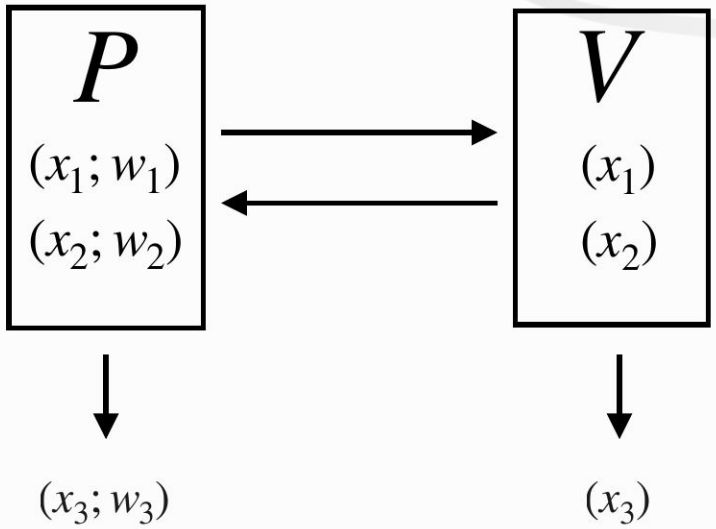

NETHERMIND RESEARCH

# Folding

- Fix a relation $R$, consisting of pairs $(x; w)$
- $x$ is a public instance, $w$ is a witness

A folding scheme is an interactive protocol between P and V where:

- P and V have two instances $x_1, x_2$.
- P also has witnesses $w_1, w_2$ such that $(x_1; w_1), (x_2; w_2) \in R$.
- P and V interact to create a new $(x_3; w_3)$ so that:





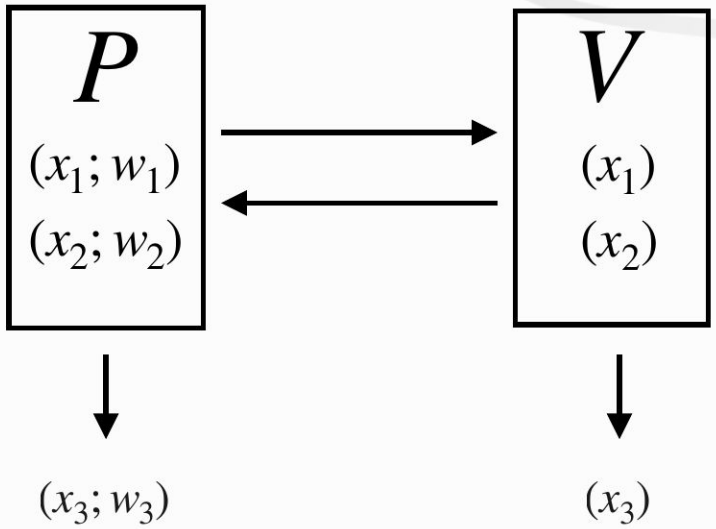

NETHERMIND RESEARCH

# Folding

- Fix a relation $R$, consisting of pairs $(x; w)$
- $x$ is a public instance, $w$ is a witness

A folding scheme is an interactive protocol between P and V where:

- P and V have two instances $x_1, x_2$.
- P also has witnesses $w_1, w_2$ such that $(x_1; w_1), (x_2; w_2) \in R$.
- P and V interact to create a new $(x_3; w_3)$ so that:
- If $(x_3; w_3) \in R$, then $(x_1; w_1), (x_2; w_2) \in R$, e.w.n.p.





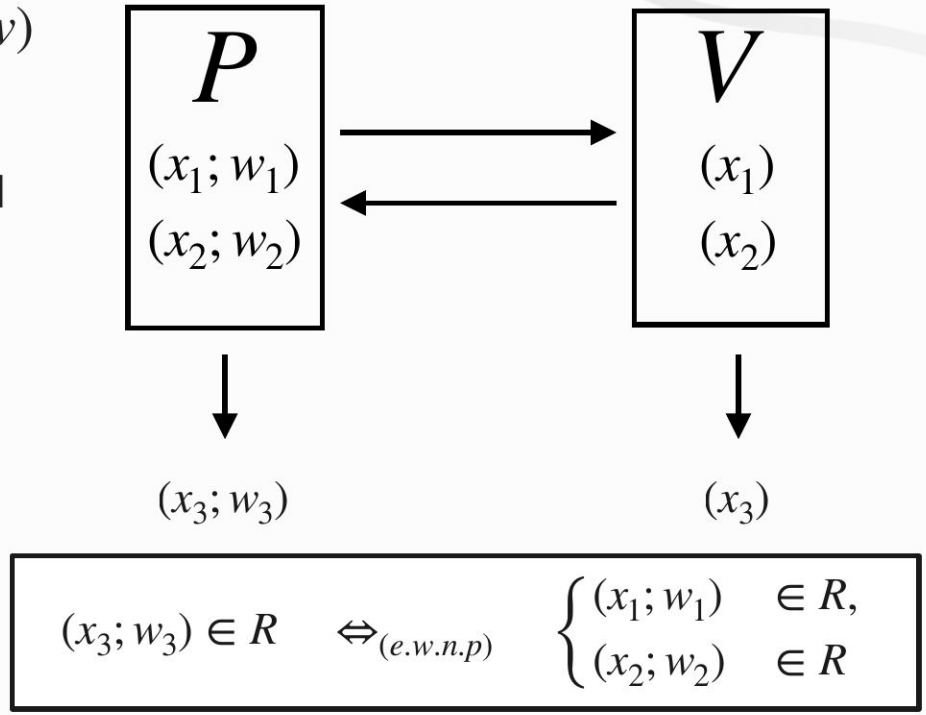

NETHERMIND RESEARCH

# Folding

- Fix a relation $R$, consisting of pairs $(x; w)$
- $x$ is a public instance, $w$ is a witness

A folding scheme is an interactive protocol between P and V where:

- P and V have two instances $x_1, x_2$.
- P also has witnesses $w_1, w_2$ such that $(x_1; w_1), (x_2; w_2) \in R$.
- P and V interact to create a new $(x_3; w_3)$ so that:
- If $(x_3; w_3) \in R$, then $(x_1; w_1), (x_2; w_2) \in R$, e.w.n.p.






NETHERMIND RESEARCH

# Folding

# Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.




NETHERMIND RESEARCH

# Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.


- Commitments play a crucial role in folding schemes.

# Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.


- Commitments play a crucial role in folding schemes.
- The instances $x_1, x_2, x_3$ all contain a commitment to $w_1, w_2, w_3$, respectively. I.e.

# Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.


- Commitments play a crucial role in folding schemes.
- The instances $x_1, x_2, x_3$ all contain a commitment to $w_1, w_2, w_3$, respectively. I.e.

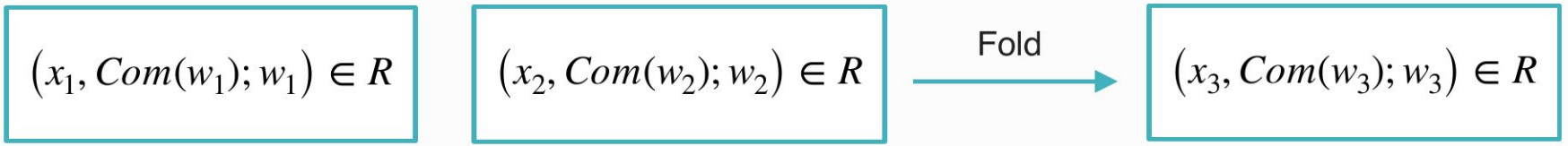
$$(x_i; w_i) = (x_i', Com(w_i); w_i)$$

# Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.


- Commitments play a crucial role in folding schemes.
- The instances $x_1, x_2, x_3$ all contain a commitment to $w_1, w_2, w_3$, respectively. I.e.

$$(x_i; w_i) = (x_i', Com(w_i); w_i)$$





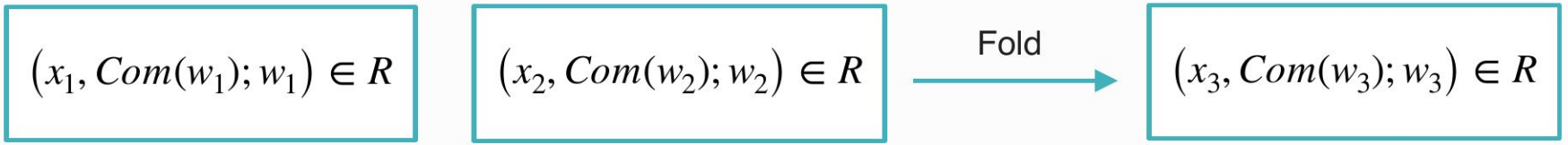

NETHERMIND RESEARCH

# Folding

- Folding reduces the task of proving 2 instance-witness to proving 1 instance-witness.


- Commitments play a crucial role in folding schemes.
- The instances $x_1, x_2, x_3$ all contain a commitment to $w_1, w_2, w_3$, respectively. I.e.

$$(x_i; w_i) = (x_i', Com(w_i); w_i)$$




- Usually the commitment is homomorphic: $Com(w_1 + w_2) = Com(w_1) + Com(w_2)$



NETHERMIND RESEARCH

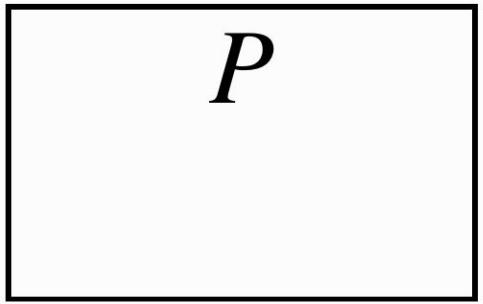
# Folding from 5000 km

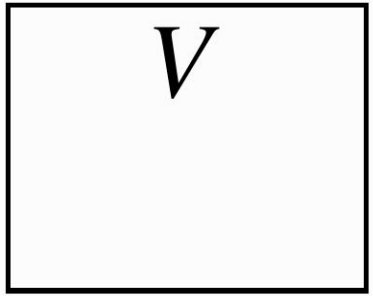
# Folding from 5000 km

- Fix $(x_1, Com(w_1); w_1) \in R, \quad (x_2, Com(w_2); w_2) \in R.$

# Folding from 5000 km

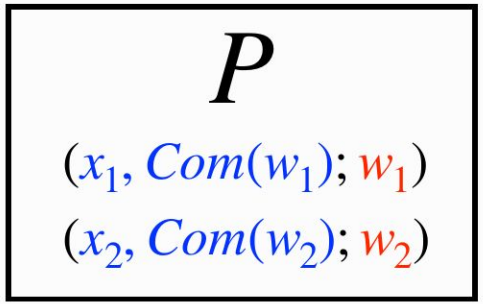
- Fix $(x_1, Com(w_1); w_1) \in R, \quad (x_2, Com(w_2); w_2) \in R.$

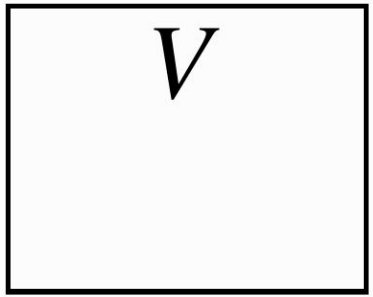


# Folding from 5000 km

- Fix $(x_1, Com(w_1); w_1) \in R, \quad (x_2, Com(w_2); w_2) \in R.$






NETHERMIND RESEARCH

# Folding from 5000 km

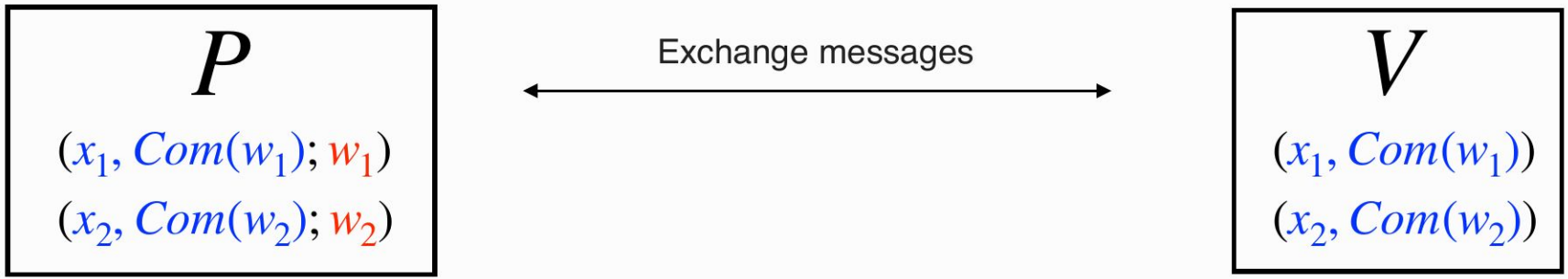
- Fix $(x_1, Com(w_1); w_1) \in R,$ $(x_2, Com(w_2); w_2) \in R.$

| $P$ |
| --- |
| $(x_1, Com(w_1); w_1)$ |
| $(x_2, Com(w_2); w_2)$ |

| $V$ |
| --- |
| $(x_1, Com(w_1))$ |
| $(x_2, Com(w_2))$ |

# Folding from 5000 km

- Fix $(x_1, Com(w_1); w_1) \in R, \quad (x_2, Com(w_2); w_2) \in R.$

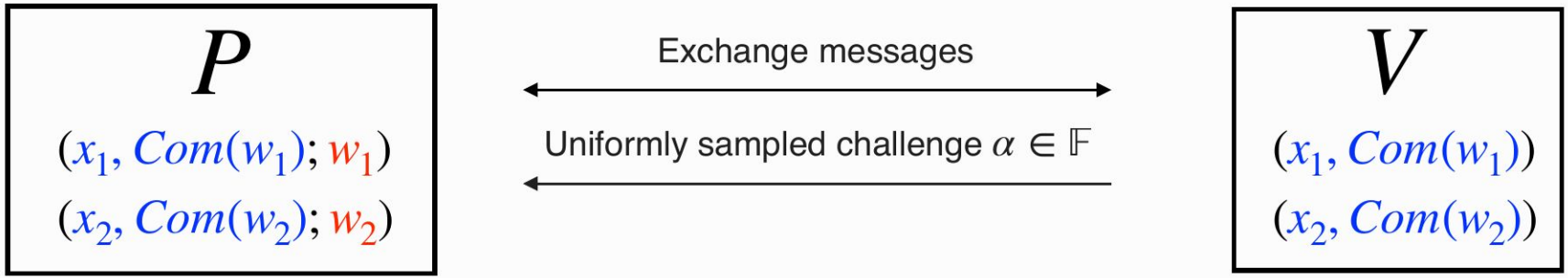





NETHERMIND RESEARCH

# Folding from 5000 km

- Fix $(x_1, Com(w_1); w_1) \in R, \quad (x_2, Com(w_2); w_2) \in R.$

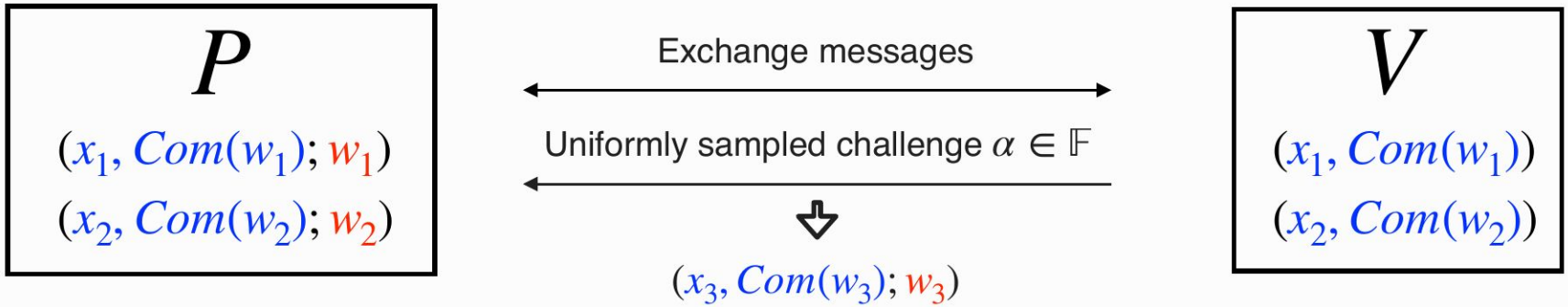





NETHERMIND RESEARCH

# Folding from 5000 km

- Fix $(x_1, Com(w_1); w_1) \in R, \quad (x_2, Com(w_2); w_2) \in R.$

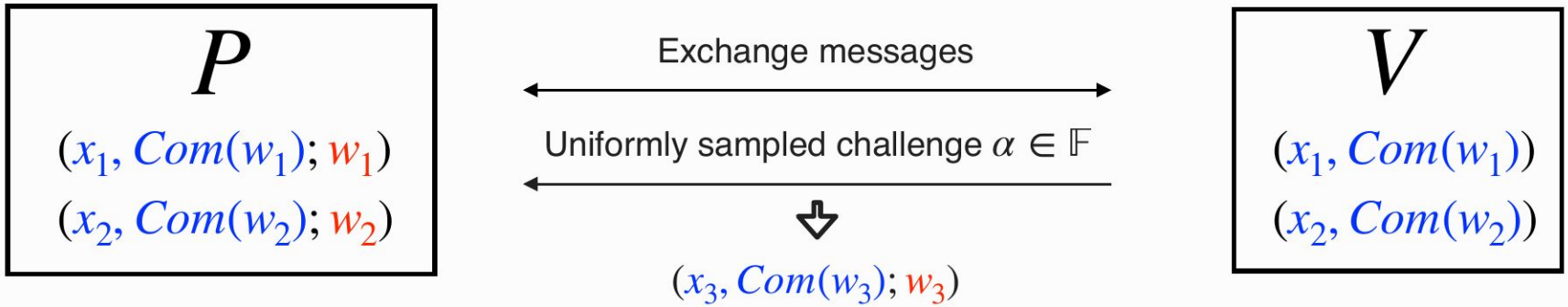





NETHERMIND RESEARCH

# Folding from 5000 km

- Fix $(x_1, Com(w_1); w_1) \in R, \quad (x_2, Com(w_2); w_2) \in R.$



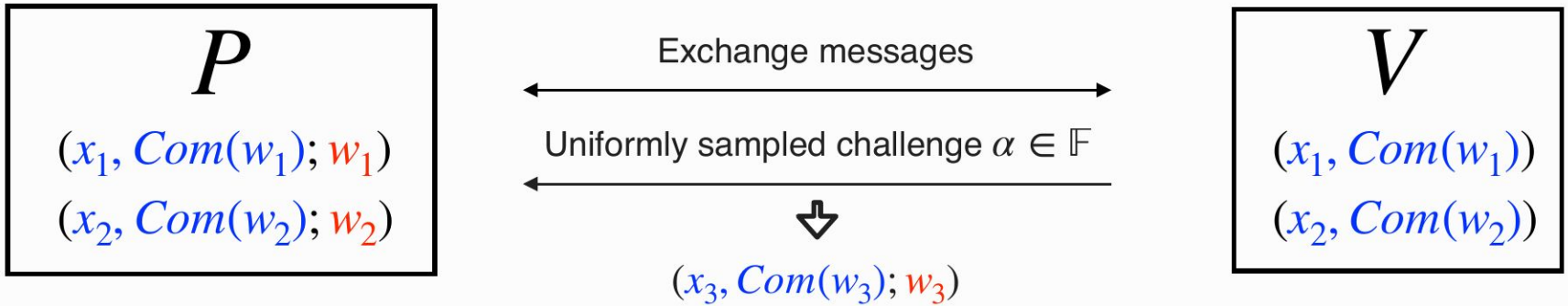

- Where $x_3 = x_1 + \alpha x_2 \qquad w_3 = w_1 + \alpha w_2$



NETHERMIND RESEARCH

# Folding from 5000 km

- Fix $(x_1, Com(w_1); w_1) \in R, \quad (x_2, Com(w_2); w_2) \in R.$




- Where $x_3 = x_1 + \alpha x_2 \qquad w_3 = w_1 + \alpha w_2$
- $V$ computes $Com(w_3)$ using that $Com(w_1) + \alpha Com(w_2) = Com(w_1 + \alpha w_2)$



NETHERMIND RESEARCH

# Commitments in folding schemes

# Commitments in folding schemes

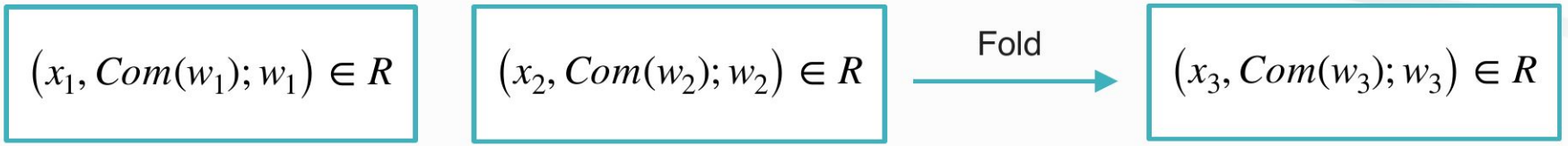
$(x_1, Com(w_1); w_1) \in R$

$(x_2, Com(w_2); w_2) \in R$

Fold

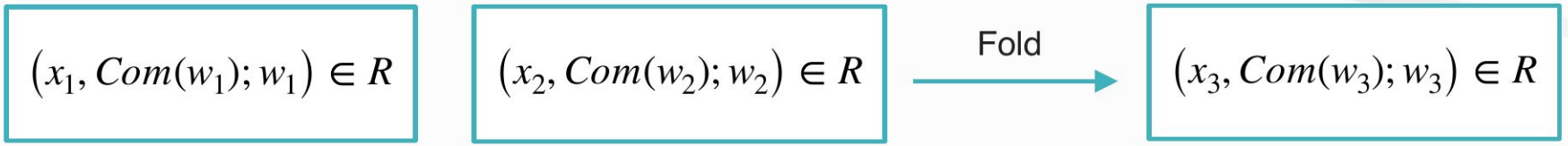
$(x_3, Com(w_3); w_3) \in R$

# Commitments in folding schemes




- Usually, $Com$ is the Pedersen or KZG scheme (or similar).

# Commitments in folding schemes

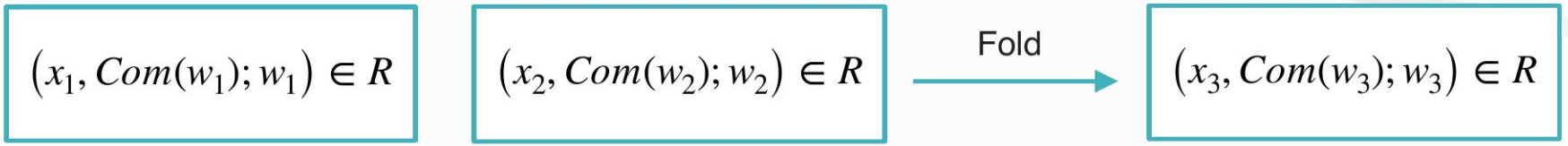



- Usually, $Com$ is the Pedersen or KZG scheme (or similar).
- E.g. in Nova, Hypernova, Protostar, Protogalaxy, etc.



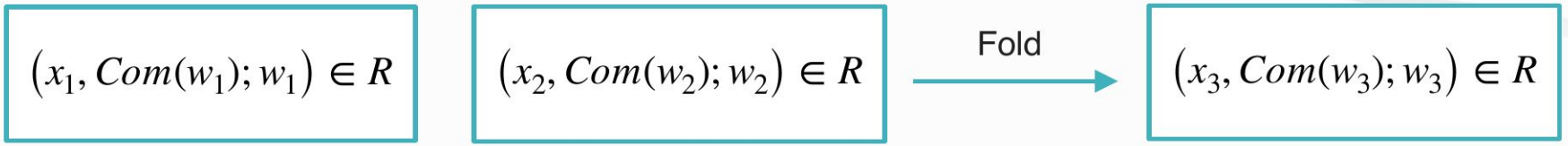
NETHERMIND RESEARCH

# Commitments in folding schemes




- Usually, $Com$ is the Pedersen or KZG scheme (or similar).
- E.g. in Nova, Hypernova, Protostar, Protogalaxy, etc.
- Inconveniences:

# Commitments in folding schemes

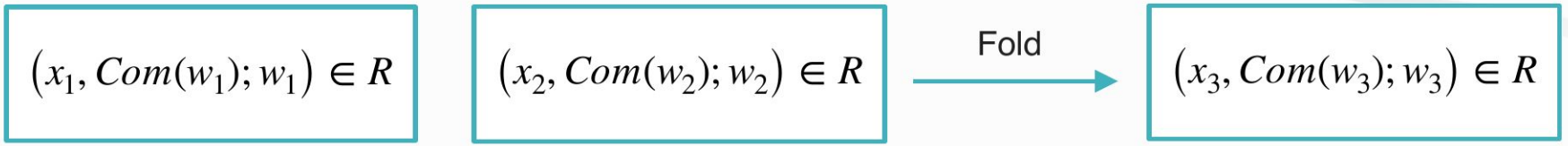



- Usually, $Com$ is the Pedersen or KZG scheme (or similar).
- E.g. in Nova, Hypernova, Protostar, Protogalaxy, etc.
- Inconveniences:
  - Witness lives in a big field: $\geq 250$ bits.



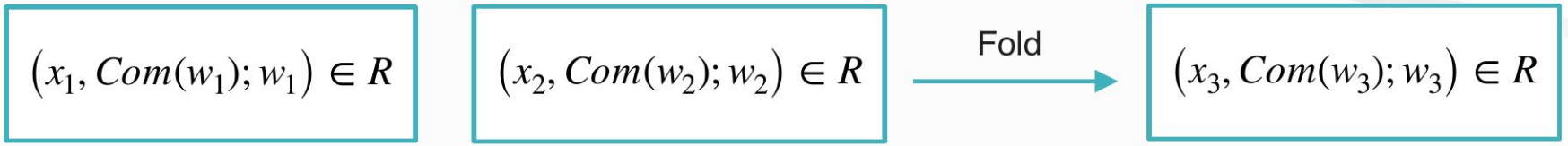
NETHERMIND RESEARCH

# Commitments in folding schemes



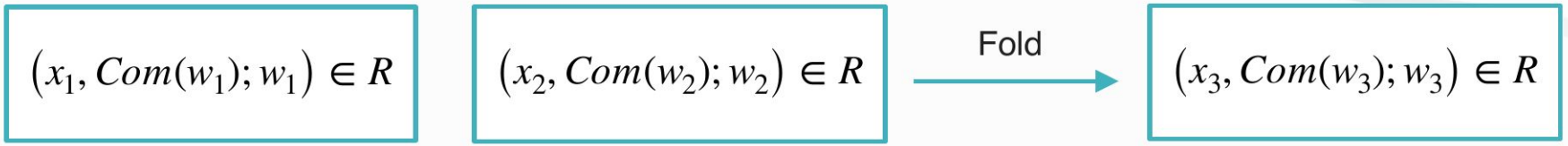

- Usually, $Com$ is the Pedersen or KZG scheme (or similar).
- E.g. in Nova, Hypernova, Protostar, Protogalaxy, etc.
- Inconveniences:
  - Witness lives in a big field: $\geq 250$ bits.
  - Commitment can very expensive, especially if the vector has large entries.

# Commitments in folding schemes




- Usually, $Com$ is the Pedersen or KZG scheme (or similar).
- E.g. in Nova, Hypernova, Protostar, Protogalaxy, etc.
- Inconveniences:
  - Witness lives in a big field: $\geq 250$ bits.
  - Commitment can very expensive, especially if the vector has large entries.
  - Expensive to recurse over: Proving that folding was performed correctly is complex due to the need of arithmetizing foreign field.

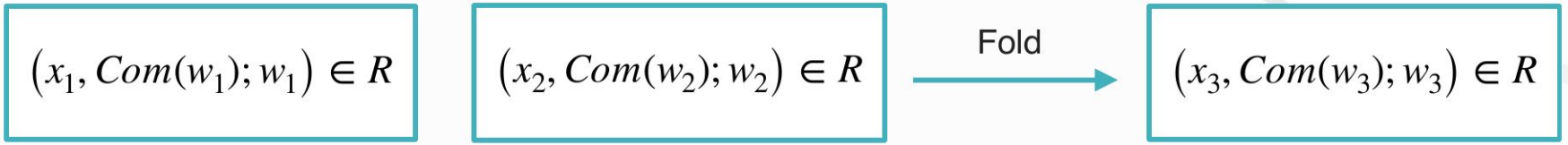
# Commitments in folding schemes




- Usually, $Com$ is the Pedersen or KZG scheme (or similar).
- E.g. in Nova, Hypernova, Protostar, Protogalaxy, etc.
- Inconveniences:
  - Witness lives in a big field: $\geq 250$ bits.
  - Commitment can very expensive, especially if the vector has large entries.
  - Expensive to recurse over: Proving that folding was performed correctly is complex due to the need of arithmetizing foreign field.
- In principle, folding then can only be combined with KZG-based SNARKs

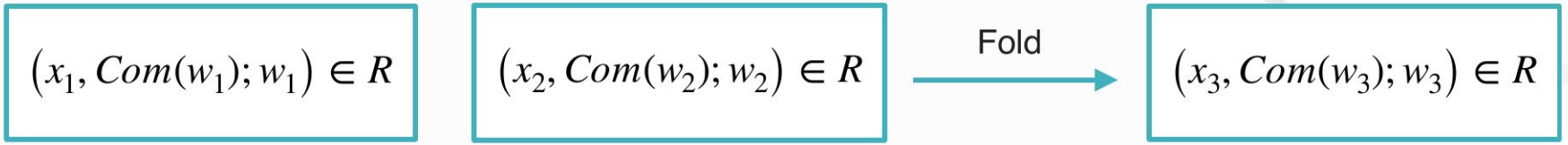
# How about folding and using STARKs?

# How about folding and using STARKs?



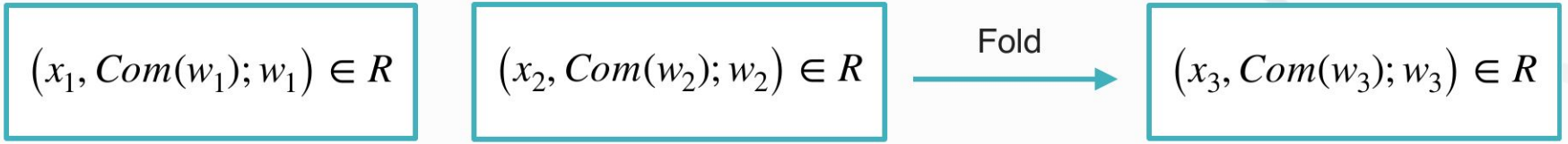

- How about folding and using STARKs?

# How about folding and using STARKs?



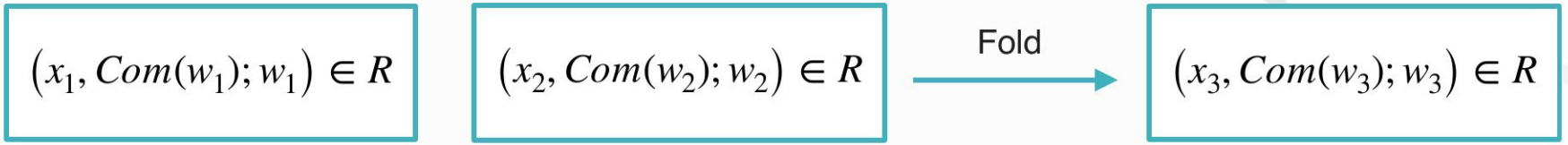

- How about folding and using STARKs?
- STARK: Loosely, a SNARK using code-based and Merkle-tree based commitments.

# How about folding and using STARKs?



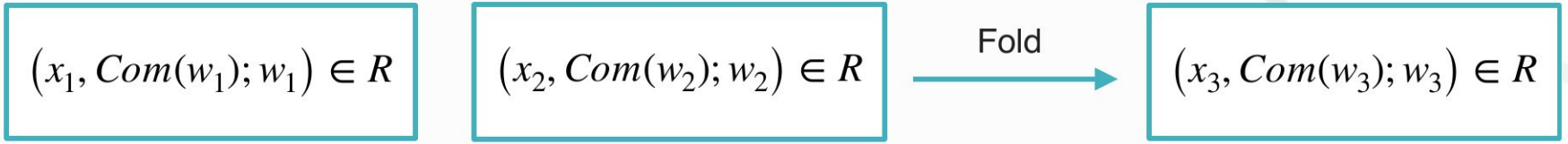

- How about folding and using STARKs?
- STARK: Loosely, a SNARK using code-based and Merkle-tree based commitments.
- E.g. ethSTARK, Plonky2/3, Boojum, Risc0, etc.

# How about folding and using STARKs?



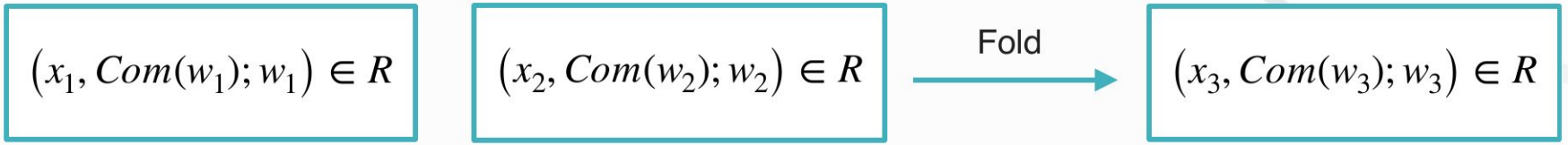

- How about folding and using STARKs?
- STARK: Loosely, a SNARK using code-based and Merkle-tree based commitments.
- E.g. ethSTARK, Plonky2/3, Boojum, Risc0, etc.
- Configured on small fields (Goldilocks 64bits, BabyBear 32bits, M31 32 bits).

# How about folding and using STARKs?



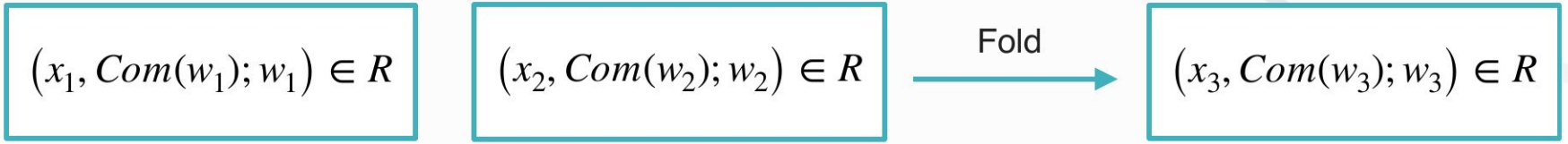

- How about folding and using STARKs?
- STARK: Loosely, a SNARK using code-based and Merkle-tree based commitments.
- E.g. ethSTARK, Plonky2/3, Boojum, Risc0, etc.
- Configured on small fields (Goldilocks 64bits, BabyBear 32bits, M31 32 bits).
    - Smaller arithmetizations due to specific properties of the primes

# How about folding and using STARKs?



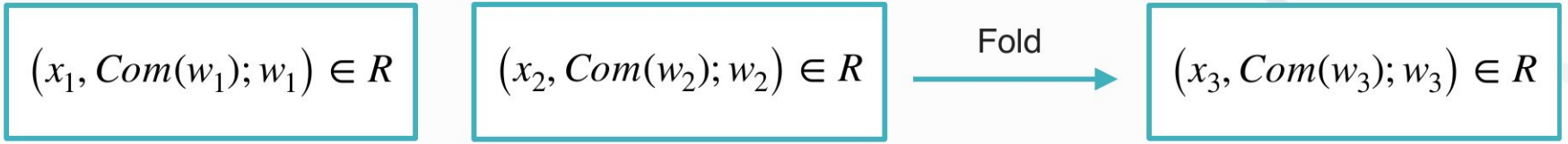

- How about folding and using STARKs?
- STARK: Loosely, a SNARK using code-based and Merkle-tree based commitments.
- E.g. ethSTARK, Plonky2/3, Boojum, Risc0, etc.
- Configured on small fields (Goldilocks 64bits, BabyBear 32bits, M31 32 bits).
    - Smaller arithmetizations due to specific properties of the primes
    - Cheaper computations, since field elements are never very large.

# How about folding and using STARKs?



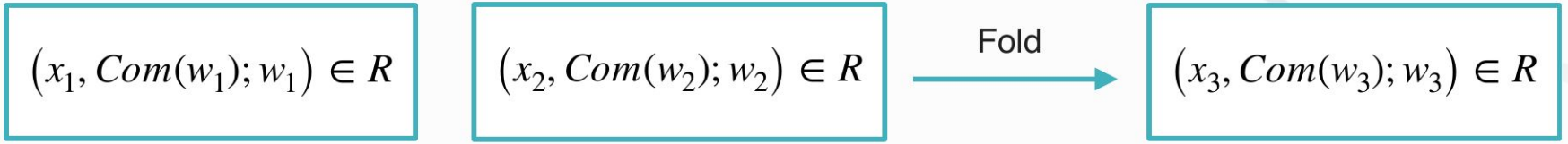

- How about folding and using STARKs?
- STARK: Loosely, a SNARK using code-based and Merkle-tree based commitments.
- E.g. ethSTARK, Plonky2/3, Boojum, Risc0, etc.
- Configured on small fields (Goldilocks 64bits, BabyBear 32bits, M31 32 bits).
  - Smaller arithmetizations due to specific properties of the primes
  - Cheaper computations, since field elements are never very large.
- Problems:

# How about folding and using STARKs?



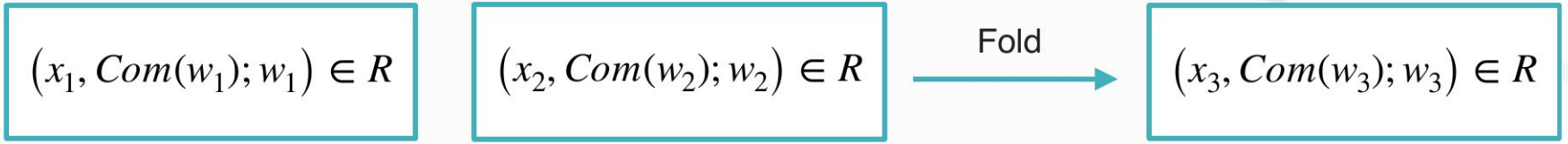

- How about folding and using STARKs?
- STARK: Loosely, a SNARK using code-based and Merkle-tree based commitments.
- E.g. ethSTARK, Plonky2/3, Boojum, Risc0, etc.
- Configured on small fields (Goldilocks 64bits, BabyBear 32bits, M31 32 bits).
  - Smaller arithmetizations due to specific properties of the primes
  - Cheaper computations, since field elements are never very large.
- Problems:
  - Merkle trees are not homomorphic

# How about folding and using STARKs?




- How about folding and using STARKs?
- STARK: Loosely, a SNARK using code-based and Merkle-tree based commitments.
- E.g. ethSTARK, Plonky2/3, Boojum, Risc0, etc.
- Configured on small fields (Goldilocks 64bits, BabyBear 32bits, M31 32 bits).
  - Smaller arithmetizations due to specific properties of the primes
  - Cheaper computations, since field elements are never very large.
- Problems:
  - Merkle trees are not homomorphic
  - Small field means we cannot rely on elliptic curves

# How about folding and using STARKs?




- How about folding and using STARKs?
- STARK: Loosely, a SNARK using code-based and Merkle-tree based commitments.
- E.g. ethSTARK, Plonky2/3, Boojum, Risc0, etc.
- Configured on small fields (Goldilocks 64bits, BabyBear 32bits, M31 32 bits).
  - Smaller arithmetizations due to specific properties of the primes
  - Cheaper computations, since field elements are never very large.
- Problems:
  - Merkle trees are not homomorphic
  - Small field means we cannot rely on elliptic curves
  - Depending on how it is done, folding may not be worth it

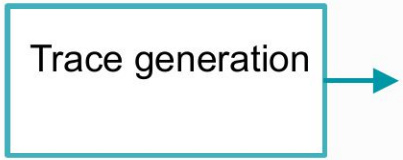
# STARK proof cost breakdown

# STARK proof cost breakdown

- Anatomy of a FRI-based STARK:

# STARK proof cost breakdown

- Anatomy of a FRI-based STARK:
    1. Compute the trace (i.e. the circuit values)

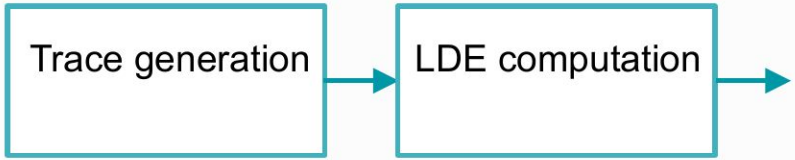





NETHERMIND RESEARCH

# STARK proof cost breakdown

- Anatomy of a FRI-based STARK:
    1. Compute the trace (i.e. the circuit values)
    2. Encode the trace: i.e. i) interpolate it (iFFT), ii) evaluate on larger domain (FFT). AKA compute the Low Degree Extension (LDE) of the trace.






NETHERMIND RESEARCH

# STARK proof cost breakdown

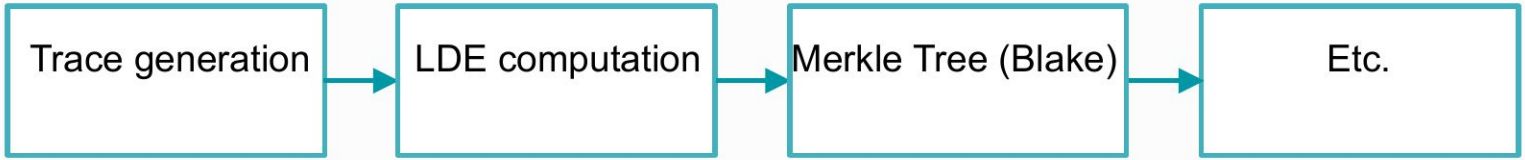
- Anatomy of a FRI-based STARK:
    1. Compute the trace (i.e. the circuit values)
    2. Encode the trace: i.e. i) interpolate it (iFFT), ii) evaluate on larger domain (FFT).
       AKA compute the Low Degree Extension (LDE) of the trace.
    3. Commit to the LDE with a Merkle tree.




NETHERMIND RESEARCH

# STARK proof cost breakdown

- Anatomy of a FRI-based STARK:
    1. Compute the trace (i.e. the circuit values)
    2. Encode the trace: i.e. i) interpolate it (iFFT), ii) evaluate on larger domain (FFT). AKA compute the Low Degree Extension (LDE) of the trace.
    3. Commit to the LDE with a Merkle tree.
    4. Etc.

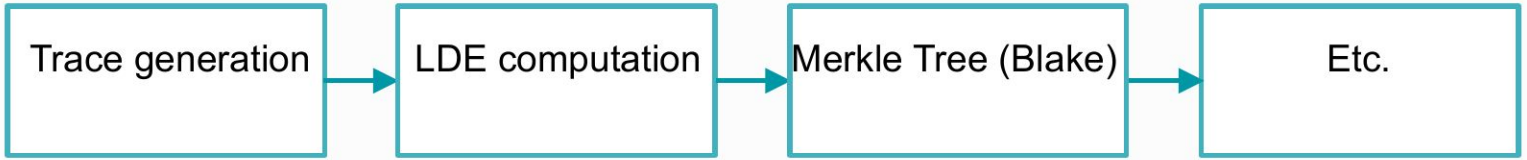





NETHERMIND RESEARCH

# STARK proof cost breakdown

- Anatomy of a FRI-based STARK:
    1. Compute the trace (i.e. the circuit values)
    2. Encode the trace: i.e. i) interpolate it (iFFT), ii) evaluate on larger domain (FFT). AKA compute the Low Degree Extension (LDE) of the trace.
    3. Commit to the LDE with a Merkle tree.
    4. Etc.
- STWO's prover cost breakdown, when proving a Blake hash computation (Source: "State of Stwo" by Eli Ben-Sasson at SBC 2024)

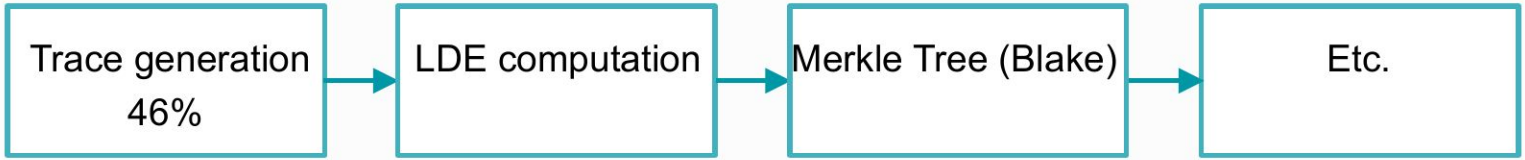





NETHERMIND RESEARCH

# STARK proof cost breakdown

- Anatomy of a FRI-based STARK:
    1. Compute the trace (i.e. the circuit values)
    2. Encode the trace: i.e. i) interpolate it (iFFT), ii) evaluate on larger domain (FFT). AKA compute the Low Degree Extension (LDE) of the trace.
    3. Commit to the LDE with a Merkle tree.
    4. Etc.
- STWO's prover cost breakdown, when proving a Blake hash computation (Source: "State of Stwo" by Eli Ben-Sasson at SBC 2024)

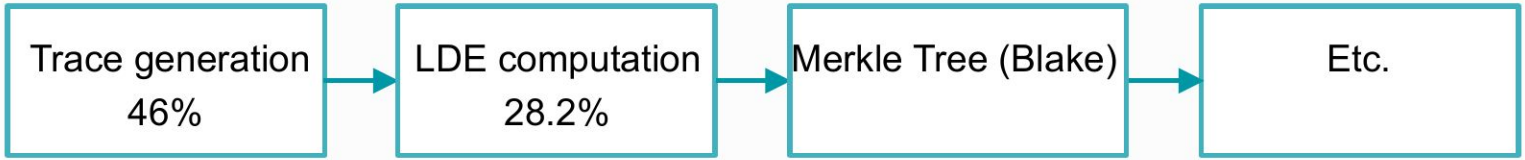





NETHERMIND RESEARCH

# STARK proof cost breakdown

- Anatomy of a FRI-based STARK:
    1. Compute the trace (i.e. the circuit values)
    2. Encode the trace: i.e. i) interpolate it (iFFT), ii) evaluate on larger domain (FFT). AKA compute the Low Degree Extension (LDE) of the trace.
    3. Commit to the LDE with a Merkle tree.
    4. Etc.
- STWO's prover cost breakdown, when proving a Blake hash computation (Source: "State of Stwo" by Eli Ben-Sasson at SBC 2024)

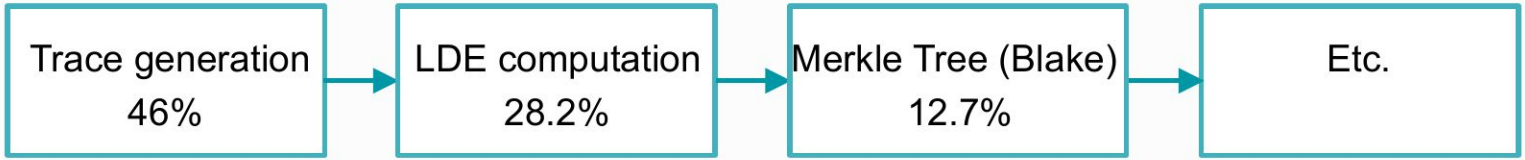





NETHERMIND RESEARCH

# STARK proof cost breakdown

- Anatomy of a FRI-based STARK:
    1. Compute the trace (i.e. the circuit values)
    2. Encode the trace: i.e. i) interpolate it (iFFT), ii) evaluate on larger domain (FFT). AKA compute the Low Degree Extension (LDE) of the trace.
    3. Commit to the LDE with a Merkle tree.
    4. Etc.
- STWO's prover cost breakdown, when proving a Blake hash computation (Source: "State of Stwo" by Eli Ben-Sasson at SBC 2024)

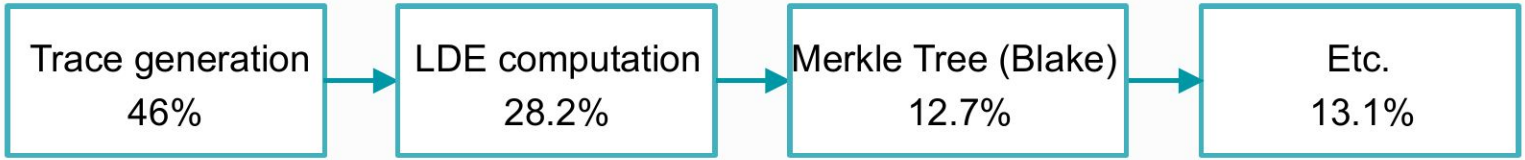





NETHERMIND RESEARCH

# STARK proof cost breakdown

- Anatomy of a FRI-based STARK:
    1. Compute the trace (i.e. the circuit values)
    2. Encode the trace: i.e. i) interpolate it (iFFT), ii) evaluate on larger domain (FFT). AKA compute the Low Degree Extension (LDE) of the trace.
    3. Commit to the LDE with a Merkle tree.
    4. Etc.
- STWO's prover cost breakdown, when proving a Blake hash computation (Source: "State of Stwo" by Eli Ben-Sasson at SBC 2024)



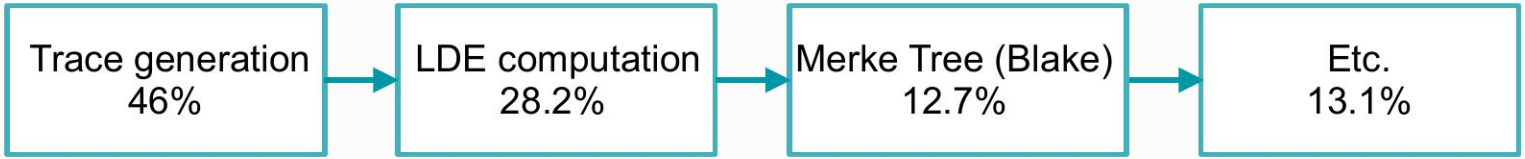



NETHERMIND RESEARCH

# STARK proof cost breakdown

# STARK proof cost breakdown

- STWO's prover cost breakdown, when proving a Blake hash computation

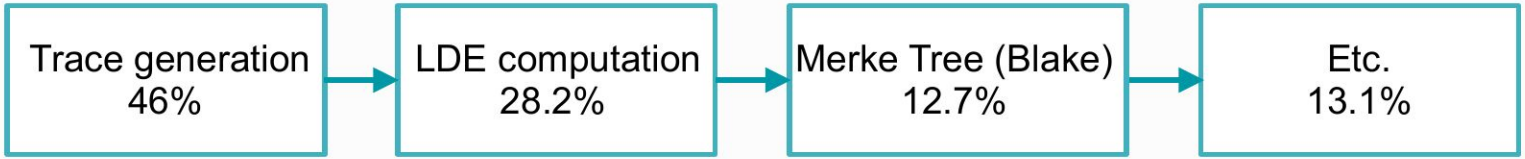





NETHERMIND RESEARCH

# STARK proof cost breakdown

- STWO's prover cost breakdown, when proving a Blake hash computation



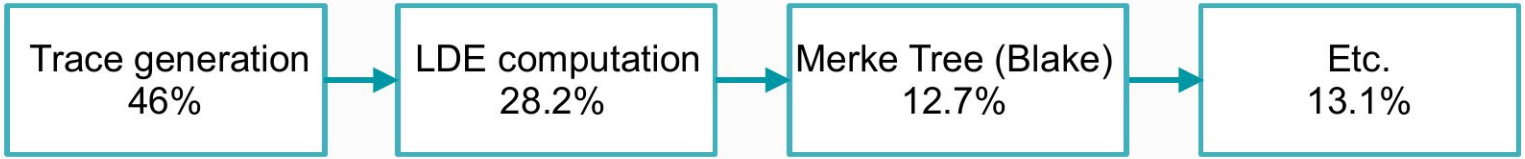

- Remark: If MT uses an algebraic hash then Step 3 is much more expensive.



NETHERMIND RESEARCH

# STARK proof cost breakdown

- STWO's prover cost breakdown, when proving a Blake hash computation



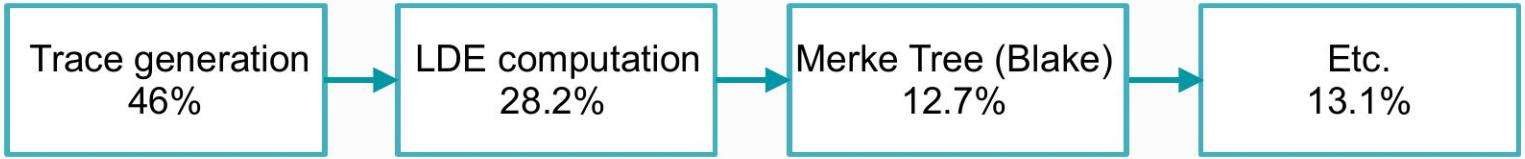

- Remark: If MT uses an algebraic hash then Step 3 is much more expensive.
- Important: Even if MT's were homomorphic, folding using the MT commitments to LDE would only save <<13% of the cost of the actual proof.



NETHERMIND RESEARCH

# STARK proof cost breakdown

- STWO's prover cost breakdown, when proving a Blake hash computation



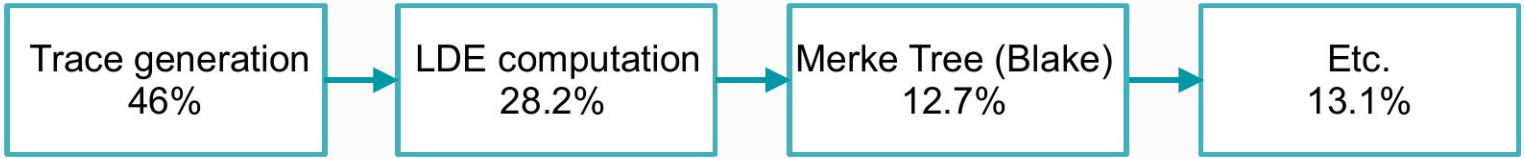

- Remark: If MT uses an algebraic hash then Step 3 is much more expensive.
- Important: Even if MT's were homomorphic, folding using the MT commitments to LDE would only save <<13% of the cost of the actual proof.
- Hence we aim to commit to the trace, instead of its LDE.



NETHERMIND RESEARCH

# STARK proof cost breakdown

- STWO's prover cost breakdown, when proving a Blake hash computation




- Remark: If MT uses an algebraic hash then Step 3 is much more expensive.
- Important: Even if MT's were homomorphic, folding using the MT commitments to LDE would only save <<13% of the cost of the actual proof.
- Hence we aim to commit to the trace, instead of its LDE.

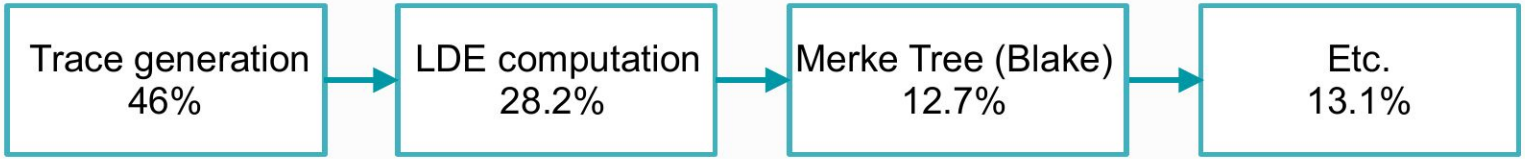
  Besides removing LDE costs, this reduces MT commitment time by a $\approx \rho$ factor ($\rho$ = rate ≤ 1/2). Saving of $\geq 50\,\%$



NETHERMIND RESEARCH

# STARK proof cost breakdown

- STWO's prover cost breakdown, when proving a Blake hash computation




- Remark: If MT uses an algebraic hash then Step 3 is much more expensive.
- Important: Even if MT's were homomorphic, folding using the MT commitments to LDE would only save <<13% of the cost of the actual proof.
- Hence we aim to commit to the trace, instead of its LDE.

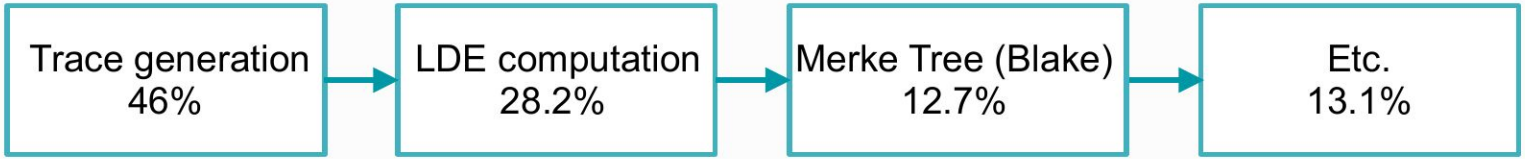
  Besides removing LDE costs, this reduces MT commitment time by a $\approx \rho$ factor ($\rho$ = rate ≤ 1/2). Saving of $\geq 50\,\%$
- Key difference with "Accumulation without Homomorphism" and "Arc" (Bünz, Mishra,



NETHERMIND RESEARCH

# STARK proof cost breakdown

- STWO's prover cost breakdown, when proving a Blake hash computation




- Remark: If MT uses an algebraic hash then Step 3 is much more expensive.
- Important: Even if MT's were homomorphic, folding using the MT commitments to LDE would only save <<13% of the cost of the actual proof.
- Hence we aim to commit to the trace, instead of its LDE.

  Besides removing LDE costs, this reduces MT commitment time by a $\approx \rho$ factor ($\rho$ = rate ≤ 1/2). Saving of $\geq 50\,\%$
- Key difference with "Accumulation without Homomorphism" and "Arc" (Bünz, Mishra, Nguyen, Wang, 2024)

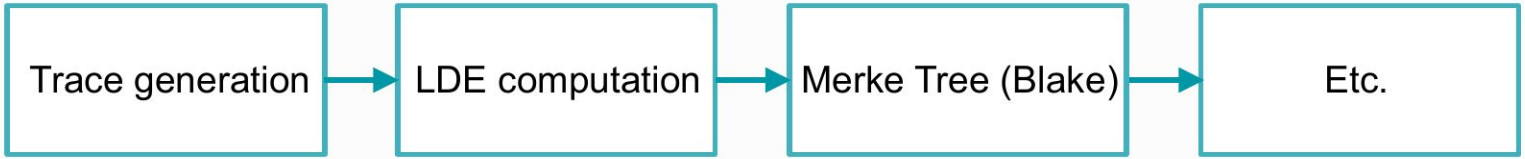


NETHERMIND RESEARCH

# Folding and STARKs

# Folding and STARKs

- Let's recap:

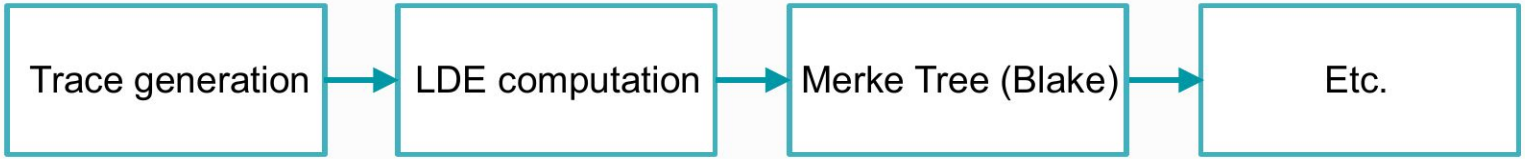





NETHERMIND RESEARCH

# Folding and STARKs

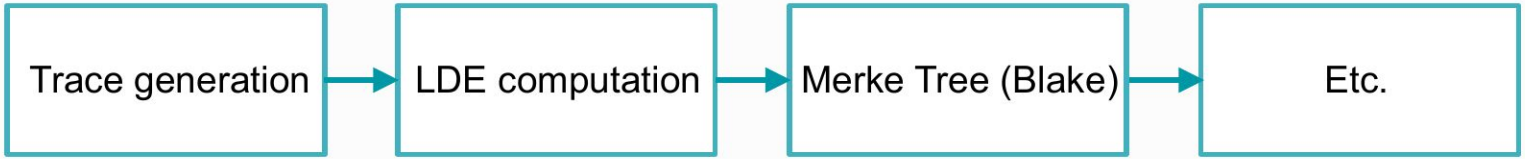
- Let's recap:




- We want to commit to the trace, rather than the LDE

# Folding and STARKs

- Let's recap:



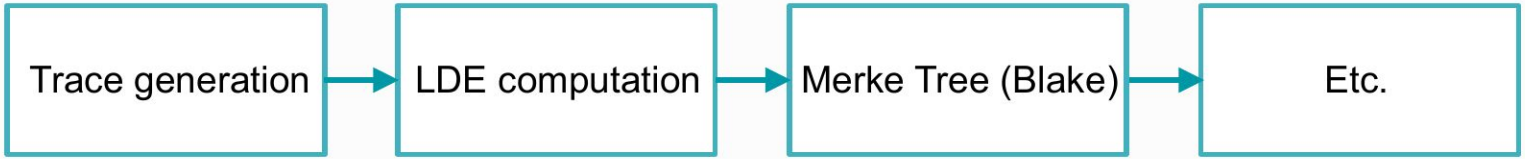

- We want to commit to the trace, rather than the LDE
- We want the scheme to be compatible with STARKs:



NETHERMIND RESEARCH

# Folding and STARKs

- Let's recap:



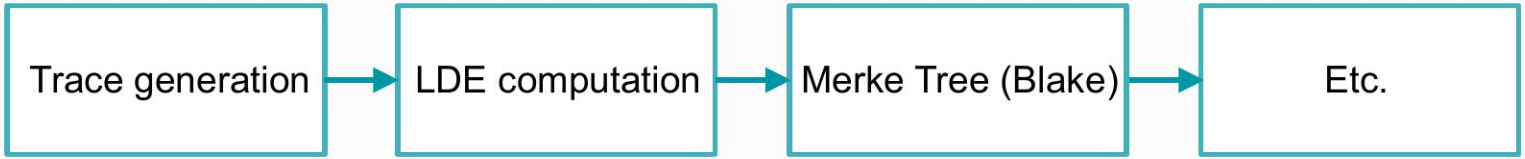

- We want to commit to the trace, rather than the LDE
- We want the scheme to be compatible with STARKs:
    - Small field



NETHERMIND RESEARCH

# Folding and STARKs

- Let's recap:



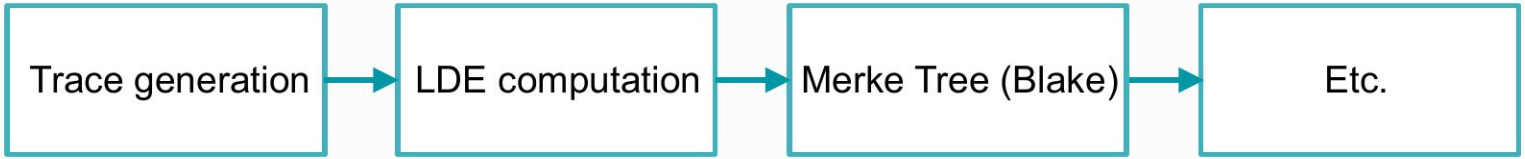

- We want to commit to the trace, rather than the LDE
- We want the scheme to be compatible with STARKs:
  - Small field
  - Folded instance to be efficiently provable with your favorite STARK



NETHERMIND RESEARCH

# Folding and STARKs

- Let's recap:



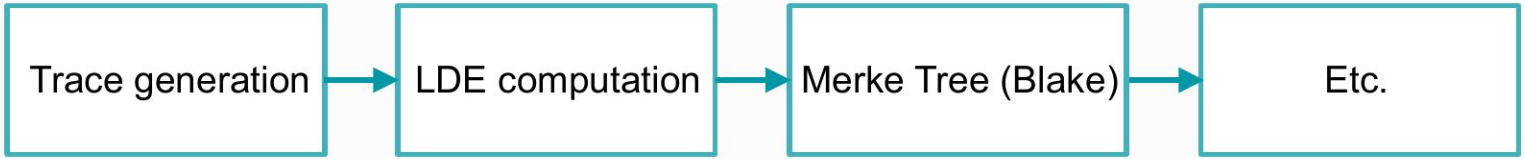

- We want to commit to the trace, rather than the LDE
- We want the scheme to be compatible with STARKs:
  - Small field
  - Folded instance to be efficiently provable with your favorite STARK
  - An instance is an AIR instance or a Plonkish instance



NETHERMIND RESEARCH

# Folding and STARKs

- Let's recap:




- We want to commit to the trace, rather than the LDE
- We want the scheme to be compatible with STARKs:
  - Small field
  - Folded instance to be efficiently provable with your favorite STARK
  - An instance is an AIR instance or a Plonkish instance
  - Using the CCS paper (Setty, Thaler, Wahby), we look at these as a CCS.

# Framework for folding STARKs

# Framework for folding STARKs

Let $R$ be the AIR or Plonkish relation. Let $(x_1; w_1) \in R, \quad (x_2; w_2) \in R.$

# Framework for folding STARKs

Let $R$ be the AIR or Plonkish relation. Let $(x_1; w_1) \in R$, $(x_2; w_2) \in R$.

Framework:

# Framework for folding STARKs

Let $R$ be the AIR or Plonkish relation. Let $(x_1; w_1) \in R$, $(x_2; w_2) \in R$.

Framework:

1. P commits to the trace (i.e. the witness) with a homomorphic scheme $Com$:

# Framework for folding STARKs

Let $R$ be the AIR or Plonkish relation. Let $(x_1; w_1) \in R$, $(x_2; w_2) \in R$.

Framework:

1. P commits to the trace (i.e. the witness) with a homomorphic scheme $Com$:

$$(x_1, Com(w_1); w_1) \in R, \qquad (x_2, Com(w_2); w_2) \in R$$

# Framework for folding STARKs

Let $R$ be the AIR or Plonkish relation. Let $(x_1; w_1) \in R$, $(x_2; w_2) \in R$.

Framework:

1. P commits to the trace (i.e. the witness) with a homomorphic scheme $Com$:

$$(x_1, Com(w_1); w_1) \in R, \qquad (x_2, Com(w_2); w_2) \in R$$

2. Fold (somehow) in a way that the folded instance belongs to $R$ (or similar):

# Framework for folding STARKs

Let $R$ be the AIR or Plonkish relation. Let $(x_1; w_1) \in R$, $(x_2; w_2) \in R$.

Framework:

1. P commits to the trace (i.e. the witness) with a homomorphic scheme $Com$:

$$(x_1, Com(w_1); w_1) \in R, \qquad (x_2, Com(w_2); w_2) \in R$$

2. Fold (somehow) in a way that the folded instance belongs to $R$ (or similar):

$$(x_3, Com(w_3); w_3) \in R$$

# Framework for folding STARKs

Let $R$ be the AIR or Plonkish relation. Let $(x_1; w_1) \in R$, $(x_2; w_2) \in R$.

Framework:

1. P commits to the trace (i.e. the witness) with a homomorphic scheme $Com$:

$$(x_1, Com(w_1); w_1) \in R, \qquad (x_2, Com(w_2); w_2) \in R$$

2. Fold (somehow) in a way that the folded instance belongs to $R$ (or similar):

$$(x_3, Com(w_3); w_3) \in R$$

3. When proving a folded instance, say $(x_3, Com(w_3); w_3)$,

# Framework for folding STARKs

Let $R$ be the AIR or Plonkish relation. Let $(x_1; w_1) \in R$, $(x_2; w_2) \in R$.

Framework:

1. P commits to the trace (i.e. the witness) with a homomorphic scheme $Com$:

$$(x_1, Com(w_1); w_1) \in R, \qquad (x_2, Com(w_2); w_2) \in R$$

2. Fold (somehow) in a way that the folded instance belongs to $R$ (or similar):

$$(x_3, Com(w_3); w_3) \in R$$

3. When proving a folded instance, say $(x_3, Com(w_3); w_3)$,
    1. Compute the LDE of $w_3$ and commit to it with a Merkle tree: $Mer(LDE(w_3)$

# Framework for folding STARKs

Let $R$ be the AIR or Plonkish relation. Let $(x_1; w_1) \in R$, $(x_2; w_2) \in R$.

Framework:

1. P commits to the trace (i.e. the witness) with a homomorphic scheme $Com$:

$$(x_1, Com(w_1); w_1) \in R, \qquad (x_2, Com(w_2); w_2) \in R$$

2. Fold (somehow) in a way that the folded instance belongs to $R$ (or similar):

$$(x_3, Com(w_3); w_3) \in R$$

3. When proving a folded instance, say $(x_3, Com(w_3); w_3)$,
    1. Compute the LDE of $w_3$ and commit to it with a Merkle tree: $Mer(LDE(w_3)$
    2. Prove that $Com(w_3)$ and $Mer(LDE(w_3)$ come from the same witness $w_3$.

# Framework for folding STARKs

Let $R$ be the AIR or Plonkish relation. Let $(x_1; w_1) \in R$, $(x_2; w_2) \in R$.

Framework:

1. P commits to the trace (i.e. the witness) with a homomorphic scheme $Com$:

$$(x_1, Com(w_1); w_1) \in R, \qquad (x_2, Com(w_2); w_2) \in R$$

2. Fold (somehow) in a way that the folded instance belongs to $R$ (or similar):

$$(x_3, Com(w_3); w_3) \in R$$

3. When proving a folded instance, say $(x_3, Com(w_3); w_3)$,
    1. Compute the LDE of $w_3$ and commit to it with a Merkle tree: $Mer(LDE(w_3)$
    2. Prove that $Com(w_3)$ and $Mer(LDE(w_3)$ come from the same witness $w_3$.
    3. Finish the proof for $(x_3; w_3) \in R$ with your favorite STARK

# Instantiating the framework

# Instantiating the framework

We need a commitment scheme $Com$ that is:

# Instantiating the framework

We need a commitment scheme $Com$ that is:

- Homomorphic

# Instantiating the framework

We need a commitment scheme $Com$ that is:

- Homomorphic
- Compatible with a STARK field

# Instantiating the framework

We need a commitment scheme $Com$ that is:

- Homomorphic
- Compatible with a STARK field

Candidate: Ajtai commitment (following Latticefold)

# Instantiating the framework

We need a commitment scheme $Com$ that is:

- Homomorphic
- Compatible with a STARK field

Candidate: Ajtai commitment (following Latticefold)

- We need a folding scheme that:

# Instantiating the framework

We need a commitment scheme $Com$ that is:

- Homomorphic
- Compatible with a STARK field

Candidate: Ajtai commitment (following Latticefold)

- We need a folding scheme that:
  - Folds pairs of AIR/Plonkish instances into (roughly) an AIR/Plonkish instance.

# Instantiating the framework

We need a commitment scheme $Com$ that is:

- Homomorphic
- Compatible with a STARK field

Candidate: Ajtai commitment (following Latticefold)

- We need a folding scheme that:
  - Folds pairs of AIR/Plonkish instances into (roughly) an AIR/Plonkish instance.

Candidate approach:

# Instantiating the framework

We need a commitment scheme $Com$ that is:

- Homomorphic
- Compatible with a STARK field

Candidate: Ajtai commitment (following Latticefold)

- We need a folding scheme that:
  - Folds pairs of AIR/Plonkish instances into (roughly) an AIR/Plonkish instance.

Candidate approach:

- Looking at AIR/Plonkish as CCS.

# Instantiating the framework

We need a commitment scheme $Com$ that is:

- Homomorphic
- Compatible with a STARK field

Candidate: Ajtai commitment (following Latticefold)

- We need a folding scheme that:
  - Folds pairs of AIR/Plonkish instances into (roughly) an AIR/Plonkish instance.

Candidate approach:

- Looking at AIR/Plonkish as CCS.
- Restrict to degree 2. Then AIR/Plonkish is roughly relaxed R1CS

# Instantiating the framework

We need a commitment scheme $Com$ that is:

- Homomorphic
- Compatible with a STARK field

Candidate: Ajtai commitment (following Latticefold)

- We need a folding scheme that:
  - Folds pairs of AIR/Plonkish instances into (roughly) an AIR/Plonkish instance.

Candidate approach:

- Looking at AIR/Plonkish as CCS.
- Restrict to degree 2. Then AIR/Plonkish is roughly relaxed R1CS
- Nova folds relaxed R1CS into relaxed R1CS. So we choose it.

# Instantiating the framework

We need a commitment scheme $Com$ that is:

- Homomorphic
- Compatible with a STARK field

Candidate: Ajtai commitment (following Latticefold)

- We need a folding scheme that:
  - Folds pairs of AIR/Plonkish instances into (roughly) an AIR/Plonkish instance.

Candidate approach:

- Looking at AIR/Plonkish as CCS.
- Restrict to degree 2. Then AIR/Plonkish is roughly relaxed R1CS
- Nova folds relaxed R1CS into relaxed R1CS. So we choose it.

Problem: There is no Nova-type folding scheme over lattices.

# Instantiating the framework

We need a commitment scheme $Com$ that is:

- Homomorphic
- Compatible with a STARK field

Candidate: Ajtai commitment (following Latticefold)

- We need a folding scheme that:
  - Folds pairs of AIR/Plonkish instances into (roughly) an AIR/Plonkish instance.

Candidate approach:

- Looking at AIR/Plonkish as CCS.
- Restrict to degree 2. Then AIR/Plonkish is roughly relaxed R1CS
- Nova folds relaxed R1CS into relaxed R1CS. So we choose it.

Problem: There is no Nova-type folding scheme over lattices.

Our main result is designing such.

# Lattices and Latticefold: crash course

# Lattices and Latticefold: crash course

- A cyclotomic ring has the form

# Lattices and Latticefold: crash course

- A cyclotomic ring has the form

$$R = \mathbb{F}_q[X]/(f(X)) = \{\text{all polys of deg} < \text{d}\}$$

# Lattices and Latticefold: crash course

- A cyclotomic ring has the form

$$R = \mathbb{F}_q[X]/(f(X)) = \{\text{all polys of deg} < \text{d}\}$$

- This is like a field extension of $\mathbb{F}_q$, but $f(X)$ may not be irreducible, and then

# Lattices and Latticefold: crash course

- A cyclotomic ring has the form

$$R = \mathbb{F}_q[X]/(f(X)) = \{\text{all polys of deg} < \text{d}\}$$

- This is like a field extension of $\mathbb{F}_q$, but $f(X)$ may not be irreducible, and then

$$R \cong \mathbb{F}_{q^t} \times \ldots \times \mathbb{F}_{q^t}$$

# Lattices and Latticefold: crash course

- A cyclotomic ring has the form

$$R = \mathbb{F}_q[X]/(f(X)) = \{\text{all polys of deg} < \text{d}\}$$

- This is like a field extension of $\mathbb{F}_q$, but $f(X)$ may not be irreducible, and then

$$R \cong \mathbb{F}_{q^t} \times \ldots \times \mathbb{F}_{q^t}$$

- Lemma: It is possible to properly configure $R$ so that $\mathbb{F}_q$ is a STARK prime field (Goldilocks, Babybear, the big STARK prime — M31 is WIP)

# Lattices and Latticefold: crash course

- A cyclotomic ring has the form

$$R = \mathbb{F}_q[X]/(f(X)) = \{\text{all polys of deg} < \text{d}\}$$

- This is like a field extension of $\mathbb{F}_q$, but $f(X)$ may not be irreducible, and then

$$R \cong \mathbb{F}_{q^t} \times \ldots \times \mathbb{F}_{q^t}$$

- Lemma: It is possible to properly configure $R$ so that $\mathbb{F}_q$ is a STARK prime field (Goldilocks, Babybear, the big STARK prime — M31 is WIP)
- The Ajtai commitment scheme for vectors of length $m$ works as follows:

# Lattices and Latticefold: crash course

- A cyclotomic ring has the form

$$R = \mathbb{F}_q[X]/(f(X)) = \{\text{all polys of deg} < \text{d}\}$$

- This is like a field extension of $\mathbb{F}_q$, but $f(X)$ may not be irreducible, and then

$$R \cong \mathbb{F}_{q^t} \times \ldots \times \mathbb{F}_{q^t}$$

- Lemma: It is possible to properly configure $R$ so that $\mathbb{F}_q$ is a STARK prime field (Goldilocks, Babybear, the big STARK prime — M31 is WIP)
- The Ajtai commitment scheme for vectors of length $m$ works as follows:
  - Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.

# Lattices and Latticefold: crash course

- A cyclotomic ring has the form

$$R = \mathbb{F}_q[X]/(f(X)) = \{\text{all polys of deg} < \text{d}\}$$

- This is like a field extension of $\mathbb{F}_q$, but $f(X)$ may not be irreducible, and then

$$R \cong \mathbb{F}_{q^t} \times \ldots \times \mathbb{F}_{q^t}$$

- Lemma: It is possible to properly configure $R$ so that $\mathbb{F}_q$ is a STARK prime field (Goldilocks, Babybear, the big STARK prime — M31 is WIP)
- The Ajtai commitment scheme for vectors of length $m$ works as follows:
  - Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.
  - Input: A vector $\vec{v} \in R^m$.

# Lattices and Latticefold: crash course

- A cyclotomic ring has the form

$$R = \mathbb{F}_q[X]/(f(X)) = \{\text{all polys of deg} < \text{d}\}$$

- This is like a field extension of $\mathbb{F}_q$, but $f(X)$ may not be irreducible, and then

$$R \cong \mathbb{F}_{q^t} \times \ldots \times \mathbb{F}_{q^t}$$

- Lemma: It is possible to properly configure $R$ so that $\mathbb{F}_q$ is a STARK prime field (Goldilocks, Babybear, the big STARK prime — M31 is WIP)
- The Ajtai commitment scheme for vectors of length $m$ works as follows:
  - Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.
  - Input: A vector $\vec{v} \in R^m$.
  - Output: The commitment $A \cdot \vec{v}^T \in R^\kappa$

# Efficiency of Ajtai

# Efficiency of Ajtai

- Set $\mathbb{F}_q$ as the Goldilocks field (64 bits)

# Efficiency of Ajtai

- Set $\mathbb{F}_q$ as the Goldilocks field (64 bits)
- Configure $R$ so that $R = \mathbb{F}_{q^3} \times \overset{8}{\ldots} \times \mathbb{F}_{q^3}$ (using 72-th cyclotomic polynomial)

# Efficiency of Ajtai

- Set $\mathbb{F}_q$ as the Goldilocks field (64 bits)
- Configure $R$ so that $R = \mathbb{F}_{q^3} \times \overset{8}{\ldots} \times \mathbb{F}_{q^3}$ (using 72-th cyclotomic polynomial)
- A vector of size $2^n$ can potentially store $8 \cdot 2^n = 2^{n+3}$ trace cells

# Efficiency of Ajtai

- Set $\mathbb{F}_q$ as the Goldilocks field (64 bits)
- Configure $R$ so that $R = \mathbb{F}_{q^3} \times \overset{8}{\ldots} \times \mathbb{F}_{q^3}$ (using 72-th cyclotomic polynomial)
- A vector of size $2^n$ can potentially store $8 \cdot 2^n = 2^{n+3}$ trace cells

| | \|v\| | Field |
|---|---|---|
| **Ajtai** | 2^16 (2^19 capacity) | 65ms |
| **Merkle (Blake)** | 2^16 | 14ms |
| **Merkle (Blake)** | 2^19 | 50ms |
| **Merkle (Blake)** | 2^20 | 100ms |
| **Merkle (Poseidon)** | 2^16 | 2.5s |



Ajtai: AMD Ryzen 7 3700X 8-Core - 64 Gb RAM

Merkles: i7 12th 12700F 8-core - 32Gb RAM

# Lattices and Latticefold: crash course

# Lattices and Latticefold: crash course

Ajtai commitment:

# Lattices and Latticefold: crash course

Ajtai commitment:

- Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.

# Lattices and Latticefold: crash course

Ajtai commitment:

- Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.
- Input: A vector $\vec{v} \in R^{m}$.

# Lattices and Latticefold: crash course

Ajtai commitment:

- Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.
- Input: A vector $\vec{v} \in R^m$.
- Output: The commitment $A \cdot \vec{v}^T \in R^\kappa$

# Lattices and Latticefold: crash course

Ajtai commitment:

- Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.
- Input: A vector $\vec{v} \in R^m$.
- Output: The commitment $A \cdot \vec{v}^T \in R^\kappa$

- Big caveat: It is only binding when $\vec{v}$ has small norm.

# Lattices and Latticefold: crash course

Ajtai commitment:

- Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.
- Input: A vector $\vec{v} \in R^m$.
- Output: The commitment $A \cdot \vec{v}^T \in R^\kappa$

- Big caveat: It is only binding when $\vec{v}$ has small norm.
- Norm = largest coeff. of an entry $v_i$ in $\vec{v}$, seeing $v_i$ as a polynomial in $R = \mathbb{F}_q[X]/(f(X))$

# Lattices and Latticefold: crash course

Ajtai commitment:

- Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.
- Input: A vector $\vec{v} \in R^m$.
- Output: The commitment $A \cdot \vec{v}^T \in R^\kappa$

- Big caveat: It is only binding when $\vec{v}$ has small norm.
- Norm = largest coeff. of an entry $v_i$ in $\vec{v}$, seeing $v_i$ as a polynomial in $R = \mathbb{F}_q[X]/(f(X))$
- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$

# Lattices and Latticefold: crash course

Ajtai commitment:

- Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.
- Input: A vector $\vec{v} \in R^m$.
- Output: The commitment $A \cdot \vec{v}^T \in R^\kappa$

- Big caveat: It is only binding when $\vec{v}$ has small norm.
- Norm = largest coeff. of an entry $v_i$ in $\vec{v}$, seeing $v_i$ as a polynomial in $R = \mathbb{F}_q[X]/(f(X))$
- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$
- Two issues:

# Lattices and Latticefold: crash course

Ajtai commitment:

- Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.
- Input: A vector $\vec{v} \in R^m$.
- Output: The commitment $A \cdot \vec{v}^T \in R^\kappa$

- Big caveat: It is only binding when $\vec{v}$ has small norm.
- Norm = largest coeff. of an entry $v_i$ in $\vec{v}$, seeing $v_i$ as a polynomial in $R = \mathbb{F}_q[X]/(f(X))$
- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$
- Two issues:
    1. $w_3$ may have large norm. Then $Com(w_3)$ is not binding.

# Lattices and Latticefold: crash course

Ajtai commitment:

- Parameters: A matrix $A \in R^{\kappa \times m}$ sampled uniformly at random.
- Input: A vector $\vec{v} \in R^m$.
- Output: The commitment $A \cdot \vec{v}^T \in R^\kappa$

- Big caveat: It is only binding when $\vec{v}$ has small norm.
- Norm = largest coeff. of an entry $v_i$ in $\vec{v}$, seeing $v_i$ as a polynomial in $R = \mathbb{F}_q[X]/(f(X))$
- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$
- Two issues:
  1. $w_3$ may have large norm. Then $Com(w_3)$ is not binding.
  2. Even if 1 does not happen, extractor should extract $w_1, w_2$ with small norm.

# The main obstacles

# The main obstacles

- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$

# The main obstacles

- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$
- Two issues:

# The main obstacles

- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$
- Two issues:
    1. $w_3$ may have large norm. Then $Com(w_3)$ is not binding.

# The main obstacles

- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$
- Two issues:
    1. $w_3$ may have large norm. Then $Com(w_3)$ is not binding.
    2. Even if 1 does not happen, extractor should extract $w_1, w_2$ with small norm.

# The main obstacles

- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$
- Two issues:
    1. $w_3$ may have large norm. Then $Com(w_3)$ is not binding.
    2. Even if 1 does not happen, extractor should extract $w_1, w_2$ with small norm.
- Latticefold's approach to 1:

# The main obstacles

- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$
- Two issues:
  1. $w_3$ may have large norm. Then $Com(w_3)$ is not binding.
  2. Even if 1 does not happen, extractor should extract $w_1, w_2$ with small norm.
- Latticefold's approach to 1:
  - The CCS's are first linearized as in Hypernova

# The main obstacles

- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$
- Two issues:
  1. $w_3$ may have large norm. Then $Com(w_3)$ is not binding.
  2. Even if 1 does not happen, extractor should extract $w_1, w_2$ with small norm.
- Latticefold's approach to 1:
  - The CCS's are first linearized as in Hypernova
  - Then each $w_i$ is decomposed as $w_i = \sum_{j \in [k]} 2^j w_{ij}$ with $w_{ij}$ having tiny norm.

# The main obstacles

- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$
- Two issues:
  1. $w_3$ may have large norm. Then $Com(w_3)$ is not binding.
  2. Even if 1 does not happen, extractor should extract $w_1, w_2$ with small norm.
- Latticefold's approach to 1:
  - The CCS's are first linearized as in Hypernova
  - Then each $w_i$ is decomposed as $w_i = \sum_{j \in [k]} 2^j w_{ij}$ with $w_{ij}$ having tiny norm.
  - Each $w_{ij}$ is a witness to a linearized CCS instance, $i = 1,2, \quad j \in [k]$

# The main obstacles

- Recall: the folded witness $w_3$ is $w_3 = w_1 + \alpha w_2$
- Two issues:
  1. $w_3$ may have large norm. Then $Com(w_3)$ is not binding.
  2. Even if 1 does not happen, extractor should extract $w_1, w_2$ with small norm.
- Latticefold's approach to 1:
  - The CCS's are first linearized as in Hypernova
  - Then each $w_i$ is decomposed as $w_i = \sum_{j \in [k]} 2^j w_{ij}$ with $w_{ij}$ having tiny norm.
  - Each $w_{ij}$ is a witness to a linearized CCS instance, $i = 1,2, \quad j \in [k]$
  - Then the $2k$ linearized CCS's are folded together.

# Obstacle 1: $w_3$ could have large norm

# Obstacle 1: $w_3$ could have large norm

Latticefold's approach to 1:

# Obstacle 1: $w_3$ could have large norm

Latticefold's approach to 1:

- The CCS's are first linearized as in Hypernova

# Obstacle 1: $w_3$ could have large norm

Latticefold's approach to 1:

- The CCS's are first linearized as in Hypernova
- Then each $w_i$ is decomposed as $w_i = \sum_{j \in [k]} 2^j w_{ij}$ with $w_{ij}$ having tiny norm.

# Obstacle 1: $w_3$ could have large norm

Latticefold's approach to 1:

- The CCS's are first linearized as in Hypernova
- Then each $w_i$ is decomposed as $w_i = \sum_{j \in [k]} 2^j w_{ij}$ with $w_{ij}$ having tiny norm.
- Each $w_{ij}$ is a witness to a linearized CCS instance, $i = 1,2, \quad j \in [k]$

# Obstacle 1: $w_3$ could have large norm

Latticefold's approach to 1:

- The CCS's are first linearized as in Hypernova
- Then each $w_i$ is decomposed as $w_i = \sum_{j \in [k]} 2^j w_{ij}$ with $w_{ij}$ having tiny norm.
- Each $w_{ij}$ is a witness to a linearized CCS instance, $i = 1,2, \quad j \in [k]$
- Then the $2k$ linearized CCS's are folded together.

# Obstacle 1: $w_3$ could have large norm

Latticefold's approach to 1:

- The CCS's are first linearized as in Hypernova
- Then each $w_i$ is decomposed as $w_i = \sum_{j \in [k]} 2^j w_{ij}$ with $w_{ij}$ having tiny norm.
- Each $w_{ij}$ is a witness to a linearized CCS instance, $i = 1,2, \quad j \in [k]$
- Then the $2k$ linearized CCS's are folded together.

Our approach to 1:

# Obstacle 1: $w_3$ could have large norm

Latticefold's approach to 1:

- The CCS's are first linearized as in Hypernova
- Then each $w_i$ is decomposed as $w_i = \sum_{j \in [k]} 2^j w_{ij}$ with $w_{ij}$ having tiny norm.
- Each $w_{ij}$ is a witness to a linearized CCS instance, $i = 1,2, \quad j \in [k]$
- Then the $2k$ linearized CCS's are folded together.

Our approach to 1:

- We want to do Nova-style folding, so we can't linearize (requires sumcheck)

# Obstacle 1: $w_3$ could have large norm

Latticefold's approach to 1:

- The CCS's are first linearized as in Hypernova
- Then each $w_i$ is decomposed as $w_i = \sum_{j \in [k]} 2^j w_{ij}$ with $w_{ij}$ having tiny norm.
- Each $w_{ij}$ is a witness to a linearized CCS instance, $i = 1,2, \quad j \in [k]$
- Then the $2k$ linearized CCS's are folded together.

Our approach to 1:

- We want to do Nova-style folding, so we can't linearize (requires sumcheck)
- We decompose $w_i$ as above.

# Obstacle 1: $w_3$ could have large norm

Latticefold's approach to 1:

- The CCS's are first linearized as in Hypernova
- Then each $w_i$ is decomposed as $w_i = \sum_{j \in [k]} 2^j w_{ij}$ with $w_{ij}$ having tiny norm.
- Each $w_{ij}$ is a witness to a linearized CCS instance, $i = 1,2, \quad j \in [k]$
- Then the $2k$ linearized CCS's are folded together.

Our approach to 1:

- We want to do Nova-style folding, so we can't linearize (requires sumcheck)
- We decompose $w_i$ as above.
- The problem is that the norm in Nova's error term $E$ can't be controlled.

# Obstacle 1: $w_3$ could have large norm

Latticefold's approach to 1:

- The CCS's are first linearized as in Hypernova
- Then each $w_i$ is decomposed as $w_i = \sum_{j \in [k]} 2^j w_{ij}$ with $w_{ij}$ having tiny norm.
- Each $w_{ij}$ is a witness to a linearized CCS instance, $i = 1,2, \quad j \in [k]$
- Then the $2k$ linearized CCS's are folded together.

Our approach to 1:

- We want to do Nova-style folding, so we can't linearize (requires sumcheck)
- We decompose $w_i$ as above.
- The problem is that the norm in Nova's error term $E$ can't be controlled.
- We follow Mova, a variation of Nova where $E$ is not committed.

# Obstacle 2: Must extract $w_1, w_2$ with small norm

# Obstacle 2: Must extract $w_1, w_2$ with small norm

Issue 2: Latticefold's approach to 2:

# Obstacle 2: Must extract $w_1, w_2$ with small norm

Issue 2: Latticefold's approach to 2:

- At each folding step, write a constraint for the norm bound of $w_i$ as a grand product.

# Obstacle 2: Must extract $w_1, w_2$ with small norm

Issue 2: Latticefold's approach to 2:

- At each folding step, write a constraint for the norm bound of $w_i$ as a grand product.
- Use sumcheck to reduce the grand product to a polynomial evaluation claim.

# Obstacle 2: Must extract $w_1, w_2$ with small norm

Issue 2: Latticefold's approach to 2:

- At each folding step, write a constraint for the norm bound of $w_i$ as a grand product.
- Use sumcheck to reduce the grand product to a polynomial evaluation claim.

Our approach to 2:

# Obstacle 2: Must extract $w_1, w_2$ with small norm

Issue 2: Latticefold's approach to 2:

- At each folding step, write a constraint for the norm bound of $w_i$ as a grand product.
- Use sumcheck to reduce the grand product to a polynomial evaluation claim.

Our approach to 2:

- We prefer to avoid sumcheck.

# Obstacle 2: Must extract $w_1, w_2$ with small norm

Issue 2: Latticefold's approach to 2:

- At each folding step, write a constraint for the norm bound of $w_i$ as a grand product.
- Use sumcheck to reduce the grand product to a polynomial evaluation claim.

Our approach to 2:

- We prefer to avoid sumcheck.
- We decompose $w_i$ into a linear combination of $w_{ij}$ such that:

# Obstacle 2: Must extract $w_1, w_2$ with small norm

Issue 2: Latticefold's approach to 2:

- At each folding step, write a constraint for the norm bound of $w_i$ as a grand product.
- Use sumcheck to reduce the grand product to a polynomial evaluation claim.

Our approach to 2:

- We prefer to avoid sumcheck.
- We decompose $w_i$ into a linear combination of $w_{ij}$ such that:

  every entry in $w_{ij}$ has binary coefficients (*)

# Obstacle 2: Must extract $w_1, w_2$ with small norm

Issue 2: Latticefold's approach to 2:

- At each folding step, write a constraint for the norm bound of $w_i$ as a grand product.
- Use sumcheck to reduce the grand product to a polynomial evaluation claim.

Our approach to 2:

- We prefer to avoid sumcheck.
- We decompose $w_i$ into a linear combination of $w_{ij}$ such that:

  every entry in $w_{ij}$ has binary coefficients (*)

- We express the property (*) as an R1CS, and fold it along with the original R1CS

NETHERMIND RESEARCH

# Thanks