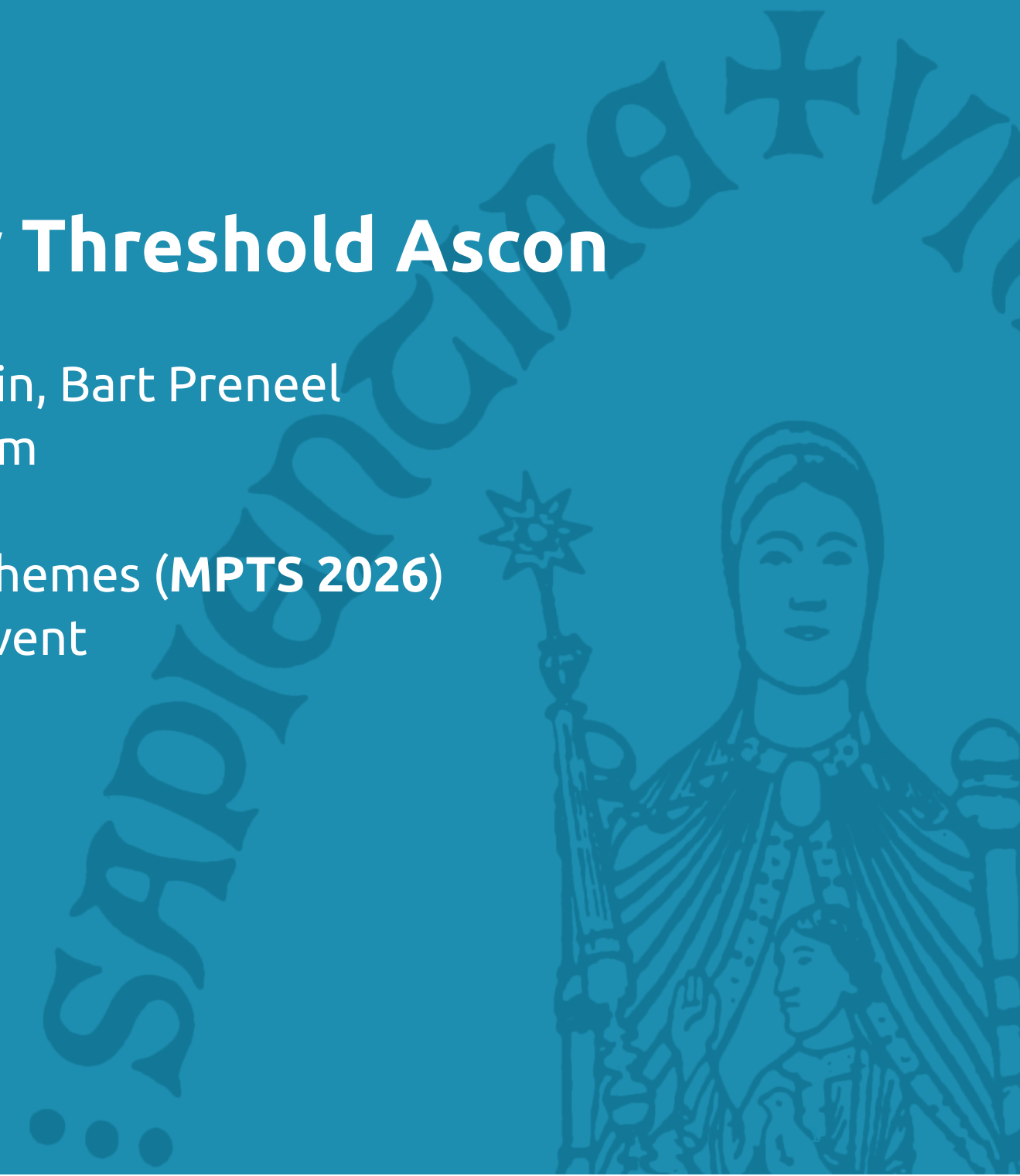


Towards an Efficient Multi-Party Threshold Ascon

Peter Schwarz, Erik Pohle, Aysajan Abidin, Bart Preneel
COSIC, KU Leuven, Belgium

NIST Workshop on Multi-Party Threshold Schemes (**MPTS 2026**)
January 28, 2026 - Virtual Event



Overview

- **Introduction**
- Ascon
- Baseline & Optimization
- Results & Discussion

Introduction - What is this talk about?

Introduction - What is this talk about?

- NIST **Lightweight** symmetric cryptography **standard Ascon...**

Introduction - What is this talk about?

- NIST **Lightweight** symmetric cryptography **standard Ascon...**
- ...evaluated in Secure **Multi-Party** Computation...

Introduction - What is this talk about?

- NIST **Lightweight** symmetric cryptography **standard Ascon...**
- ...evaluated in Secure **Multi-Party** Computation...
- **efficiently** - in **online** communication costs in bits

Introduction - What is this talk about?

- NIST **Lightweight** symmetric cryptography **standard Ascon...**
- ...evaluated in Secure **Multi-Party** Computation...
- **efficiently** - in **online** communication costs in bits
- Why? e.g.:

Introduction - What is this talk about?

- NIST **Lightweight** symmetric cryptography **standard Ascon...**
- ...evaluated in Secure **Multi-Party** Computation...
- **efficiently** - in **online** communication costs in bits
- Why? e.g.:
 - Oblivious en-/decryption for key security

Introduction - What is this talk about?

- NIST **Lightweight** symmetric cryptography **standard Ascon...**
- ...evaluated in Secure **Multi-Party** Computation...
- **efficiently** - in **online** communication costs in bits
- Why? e.g.:
 - Oblivious en-/decryption for key security
 - Enabling flexible computing on encrypted data (**COED**) usecases

Introduction - Secure Multi-Party Computation (MPC):

Introduction - Secure Multi-Party Computation (MPC):

- Multiple mutually distrusting parties can **calculate** a **function** on the parties' **private inputs**

Introduction - Secure Multi-Party Computation (MPC):

- Multiple mutually distrusting parties can **calculate** a **function** on the parties' **private inputs**
- **Threshold Secret-Sharing** based:

Introduction - Secure Multi-Party Computation (MPC):

- Multiple mutually distrusting parties can **calculate** a **function** on the parties' **private inputs**
- **Threshold Secret-Sharing** based:
 - Input **values** are **split** across parties to preserve privacy

Introduction - Secure Multi-Party Computation (MPC):

- Multiple mutually distrusting parties can **calculate** a **function** on the parties' **private inputs**
- **Threshold Secret-Sharing** based:
 - Input **values** are **split** across parties to preserve privacy
- **Non-Linear** operations require **communication** → bottleneck

Introduction - Secure Multi-Party Computation (MPC):

- Multiple mutually distrusting parties can **calculate** a **function** on the parties' **private inputs**
- **Threshold Secret-Sharing** based:
 - Input **values** are **split** across parties to preserve privacy
- **Non-Linear** operations require **communication** → bottleneck
- **Linear** operations: **performed locally** on shares by parties

Introduction - Focus:

Introduction - Focus:

- **Online-Phase** - Offline phase: to be evaluated in future work

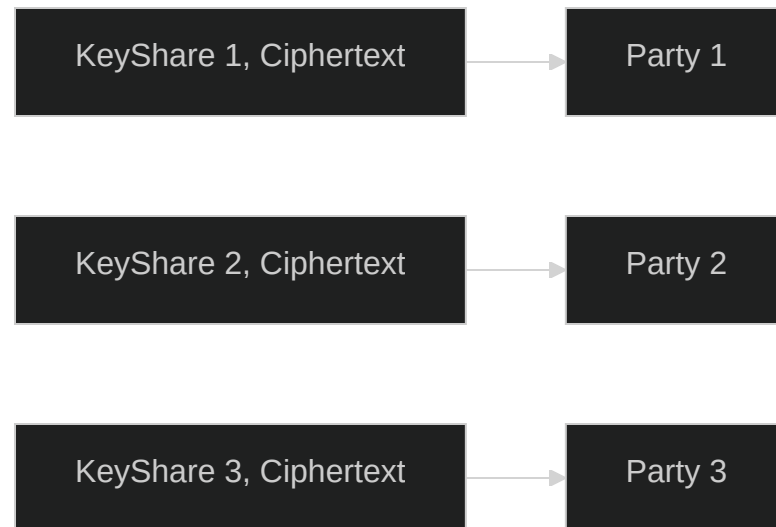
Introduction - Focus:

- **Online-Phase** - Offline phase: to be evaluated in future work
- Honest supermajority ($t < n/3$; t : num. **corrupted** parties, n : num. **total** parties), **Active security**

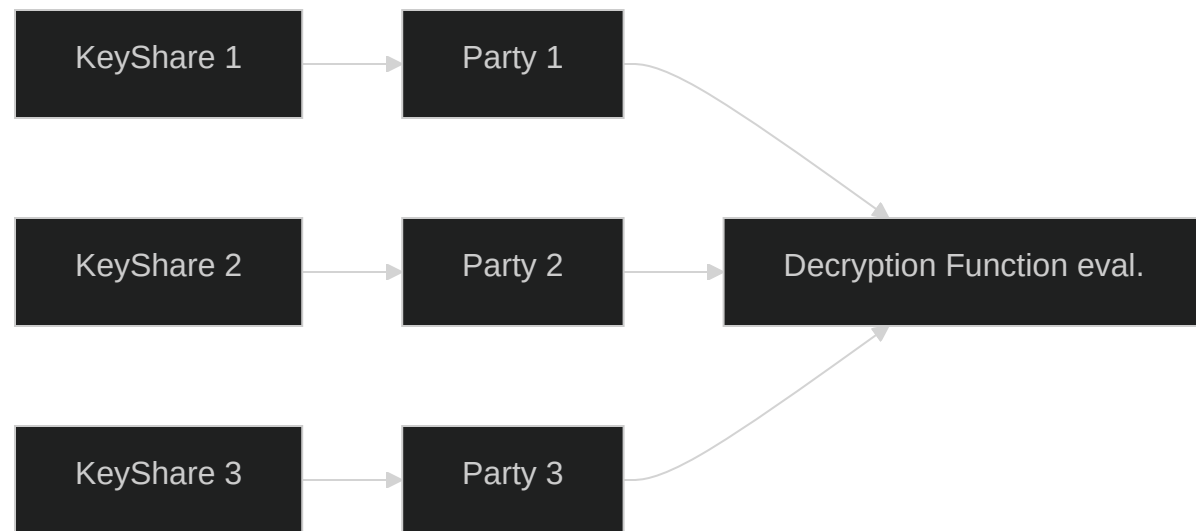
Introduction - Focus:

- **Online-Phase** - Offline phase: to be evaluated in future work
- Honest supermajority ($t < n/3$; t : num. **corrupted** parties, n : num. **total** parties), **Active security**
- Ascon-AEAD128

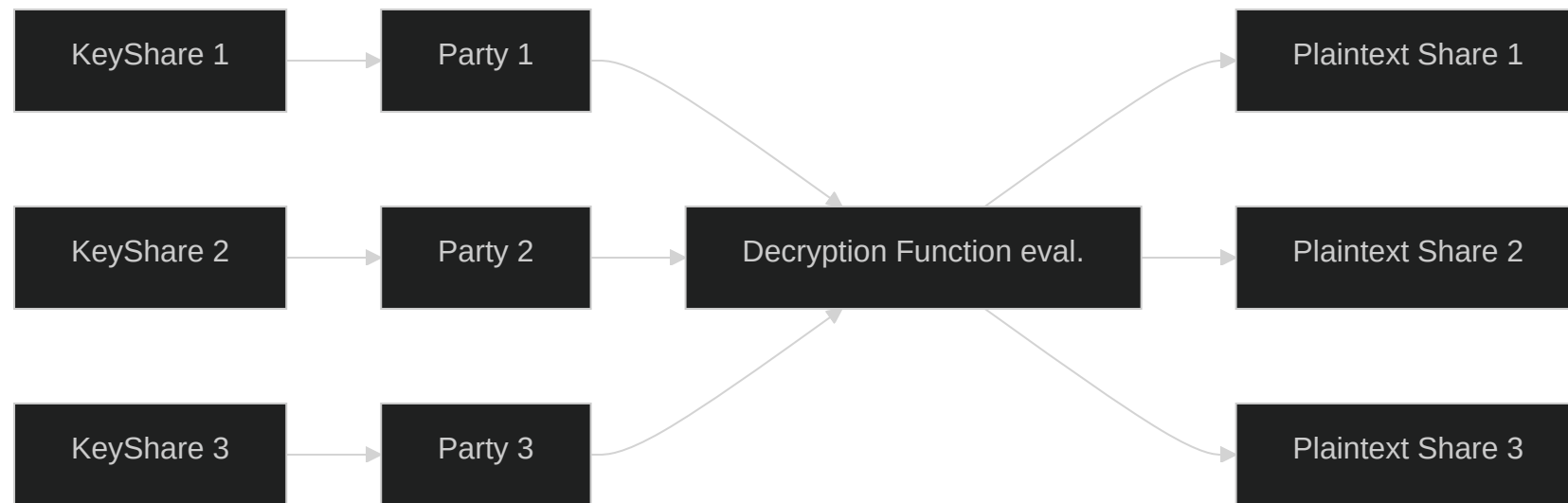
Introduction - Oblivious Decryption in MPC:



Introduction - Oblivious Decryption in MPC:



Introduction - Oblivious Decryption in MPC:

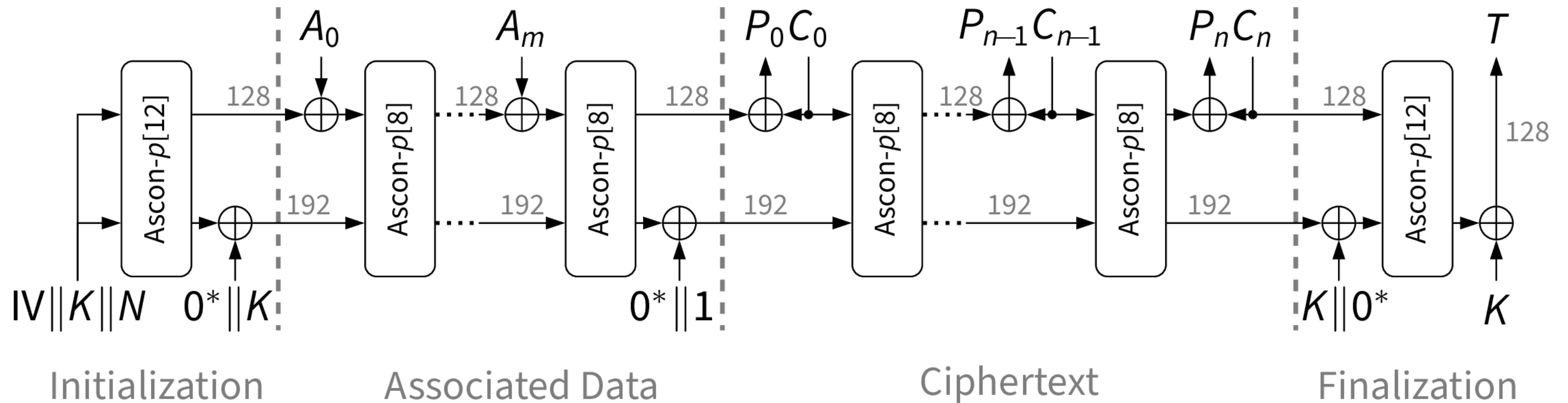


Overview

- ~~Introduction~~
- **Ascon**
- Baseline & Optimization
- Results & Discussion

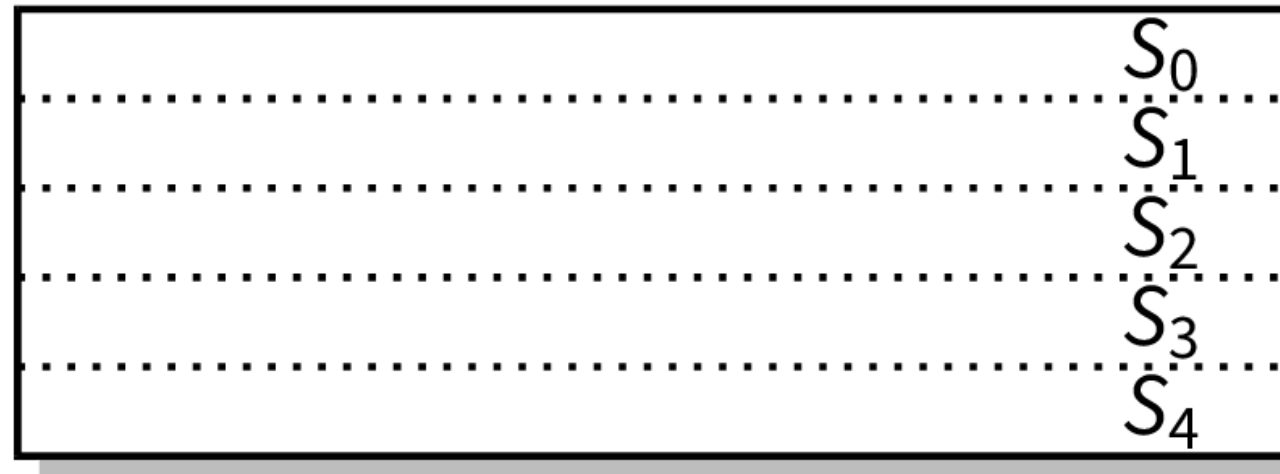
Background - Ascon Structure

Ascon-AEAD128 authenticated decryption mode:



Background - Ascon Structure

320-bit state: Five 64-bit words S_0, S_1, S_2, S_3, S_4



Background - Ascon Structure

Background - Ascon Structure

- **Permutation** $p = p_L \circ p_S \circ p_C$:

Background - Ascon Structure

- **Permutation** $p = p_L \circ p_S \circ p_C$:
 - p_C : Add round constants, linear

Background - Ascon Structure

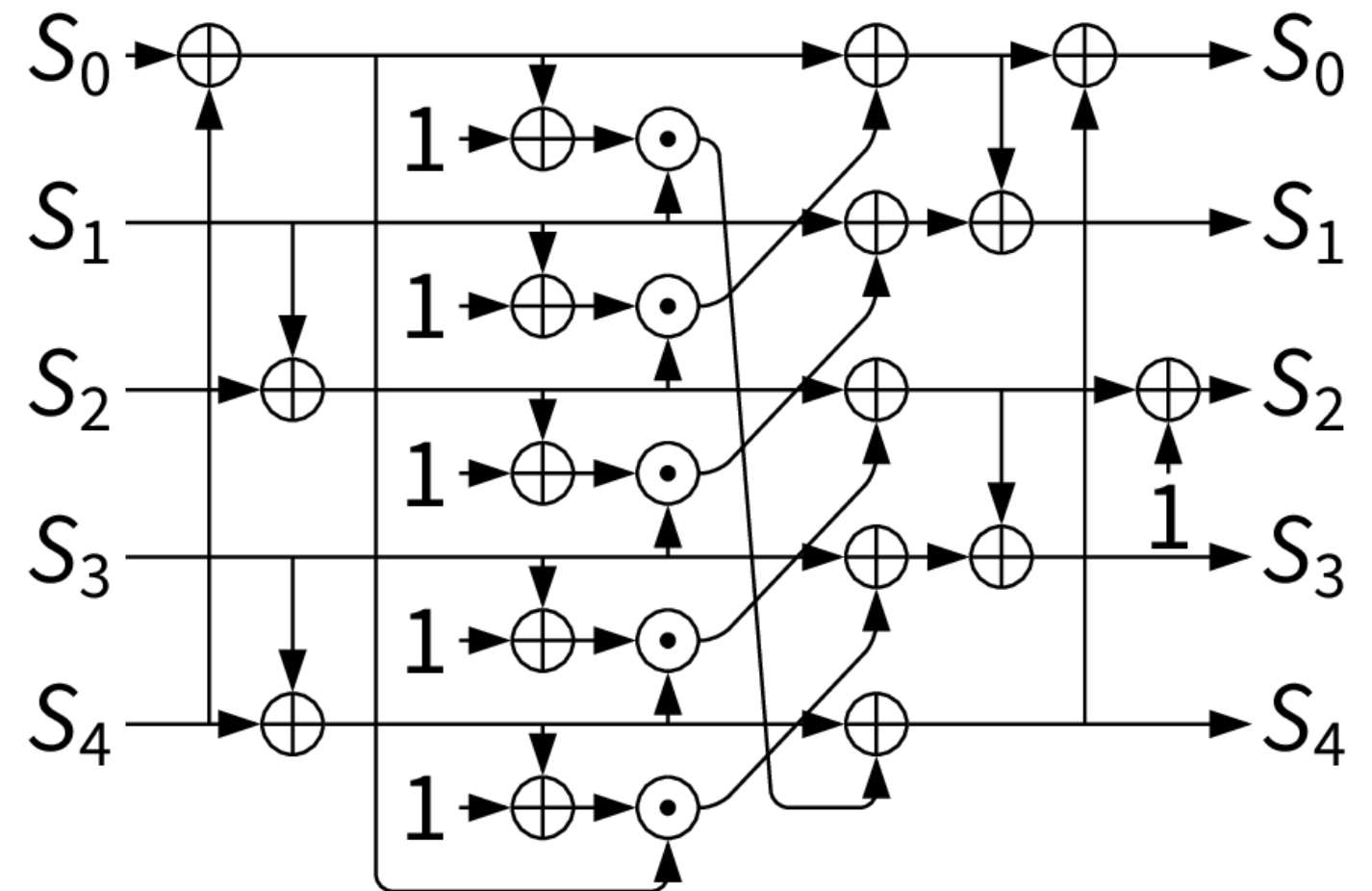
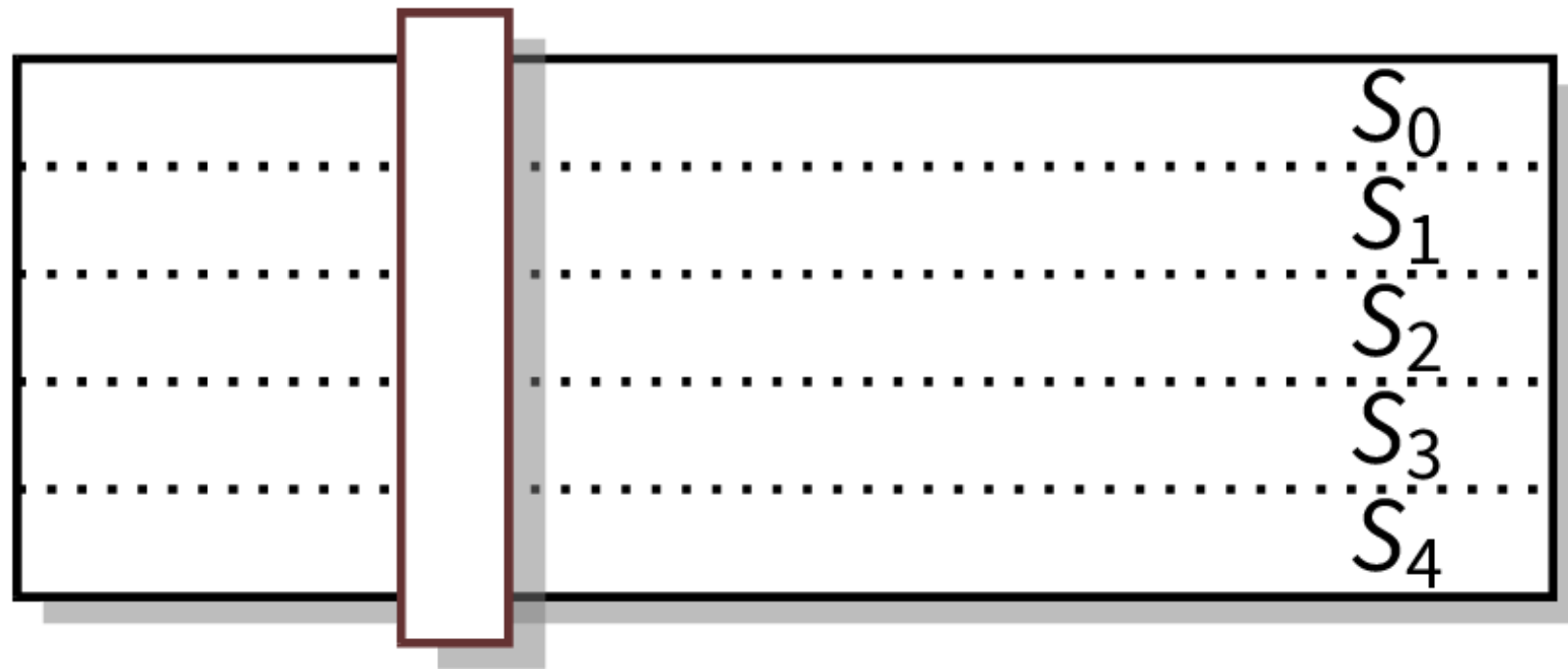
- **Permutation** $p = p_L \circ p_S \circ p_C$:
 - p_C : Add round constants, linear
 - p_S : 5-bit S-box applied to 64 vertical slices

Background - Ascon Structure

- **Permutation** $p = p_L \circ p_S \circ p_C$:
 - p_C : Add round constants, linear
 - p_S : 5-bit S-box applied to 64 vertical slices
 - p_L : Bit rotations within each word (linear in naive approach)

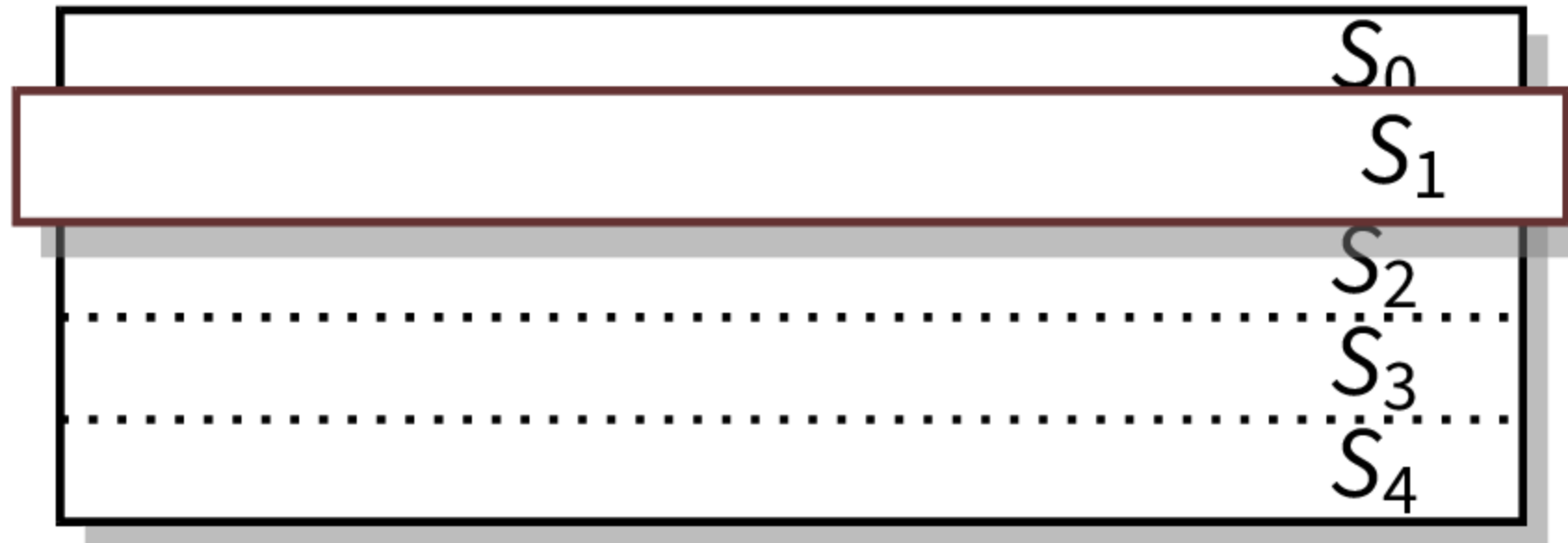
Background - Ascon Structure

Substitution layer, state structure and function:



Background - Ascon Structure

Linear Layer, state structure and function:



$$\Sigma_0(S_0) = S_0 \oplus (S_0 \ggg 19) \oplus (S_0 \ggg 28)$$

·
·
·
·

Overview

- ~~Introduction~~
- ~~Ascon~~
- **Baseline & Optimization**
- Results & Discussion

Baseline & Bottleneck

Baseline & Bottleneck

- **Ascon as a Boolean circuit:**

Baseline & Bottleneck

- **Ascon as a Boolean circuit:**
 - **Secret-share** each **bit** individually

Baseline & Bottleneck

- **Ascon as a Boolean circuit:**
 - **Secret-share** each **bit** individually
 - **XOR** operations (concatenations, **rotations**): **local** (no communication)

Baseline & Bottleneck

- **Ascon as a Boolean circuit:**
 - **Secret-share** each **bit** individually
 - **XOR** operations (concatenations, **rotations**): **local** (no communication)
 - **AND** operations: **require communication** ← bottleneck

Baseline & Bottleneck

- **Ascon as a Boolean circuit:**
 - **Secret-share** each **bit** individually
 - **XOR** operations (concatenations, **rotations**): **local** (no communication)
 - **AND** operations: **require communication** ← bottleneck
- **Cost** impacts:

Baseline & Bottleneck

- **Ascon as a Boolean circuit:**
 - **Secret-share** each **bit** individually
 - **XOR** operations (concatenations, **rotations**): **local** (no communication)
 - **AND** operations: **require communication** ← bottleneck
- **Cost** impacts:
 - 320 AND gates

Baseline & Bottleneck

- **Ascon as a Boolean circuit:**
 - **Secret-share** each **bit** individually
 - **XOR** operations (concatenations, **rotations**): **local** (no communication)
 - **AND** operations: **require communication** ← bottleneck
- **Cost** impacts:
 - 320 AND gates
 - Extension fields & share size blow-up for security (e.g. 5 bits transferred per computational bit)

Baseline & Bottleneck

- **Ascon as a Boolean circuit:**
 - **Secret-share** each **bit** individually
 - **XOR** operations (concatenations, **rotations**): **local** (no communication)
 - **AND** operations: **require communication** ← bottleneck
- **Cost** impacts:
 - 320 AND gates
 - Extension fields & share size blow-up for security (e.g. 5 bits transferred per computational bit)
- **Question:** Can we reduce num. of AND gates or amortize their cost?

Baseline & Bottleneck

- **Ascon as a Boolean circuit:**
 - **Secret-share** each **bit** individually
 - **XOR** operations (concatenations, **rotations**): **local** (no communication)
 - **AND** operations: **require communication** ← bottleneck
- **Cost** impacts:
 - 320 AND gates
 - Extension fields & share size blow-up for security (e.g. 5 bits transferred per computational bit)
- **Question:** Can we reduce num. of AND gates or amortize their cost?
 - What about those extension fields?

Optimizing Communication: Packing Strategy

Optimizing Communication: Packing Strategy

- **Core observation:** Ascon was designed for **SIMD** (single instruction, multiple data)

Optimizing Communication: Packing Strategy

- **Core observation:** Ascon was designed for **SIMD** (single instruction, multiple data)
- **Our approach: Pack** multiple bits into field elements

Optimizing Communication: Packing Strategy

- **Core observation:** Ascon was designed for **SIMD** (single instruction, multiple data)
- **Our approach: Pack** multiple bits into field elements
 - Instead of 320 separate AND operations...

Optimizing Communication: Packing Strategy

- **Core observation:** Ascon was designed for **SIMD** (single instruction, multiple data)
- **Our approach: Pack** multiple bits into field elements
 - Instead of 320 separate AND operations...
 - ...**process packed** data with fewer multiplications

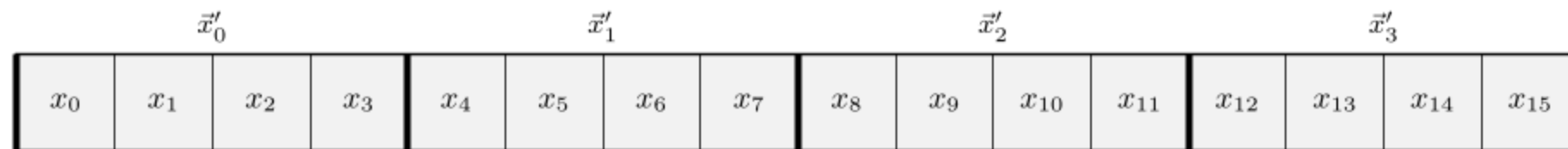
Optimizing Communication: Packing Strategy

- **Core observation:** Ascon was designed for **SIMD** (single instruction, multiple data)
- **Our approach: Pack** multiple bits into field elements
 - Instead of 320 separate AND operations...
 - ...**process packed** data with fewer multiplications
 - Amortize communication over multiple AND gates

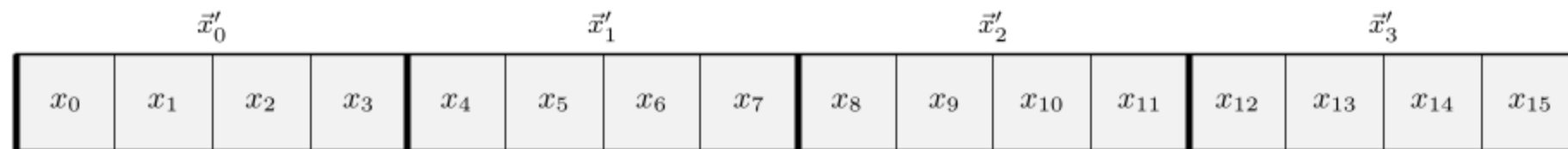
Optimizing Communication: Packing Strategy

- **Core observation:** Ascon was designed for **SIMD** (single instruction, multiple data)
- **Our approach: Pack** multiple bits into field elements
 - Instead of 320 separate AND operations...
 - ...**process packed** data with fewer multiplications
 - Amortize communication over multiple AND gates
- **Tool:** Reverse Multiplication-Friendly Embeddings (RMFEs)

Optimization - Idea: Packing



Optimization - Idea: Packing



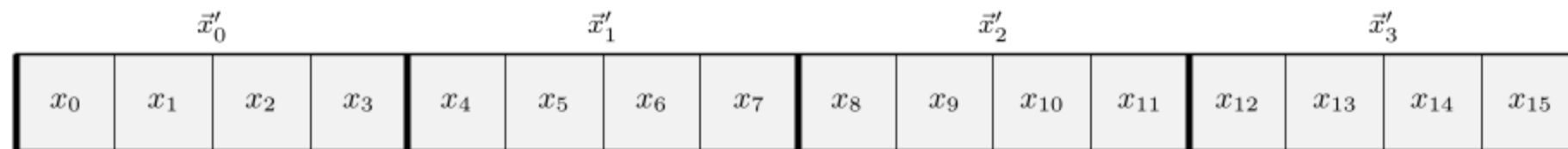
- **Pack** k bits ($x_{0..3}$) \rightarrow single **field element** in \mathbb{F}_{2^m}

Optimization - Idea: Packing



- **Pack** k bits ($x_{0..3}$) \rightarrow single **field element** in \mathbb{F}_{2^m}
 - Compute on field element (packed values)

Optimization - Idea: Packing



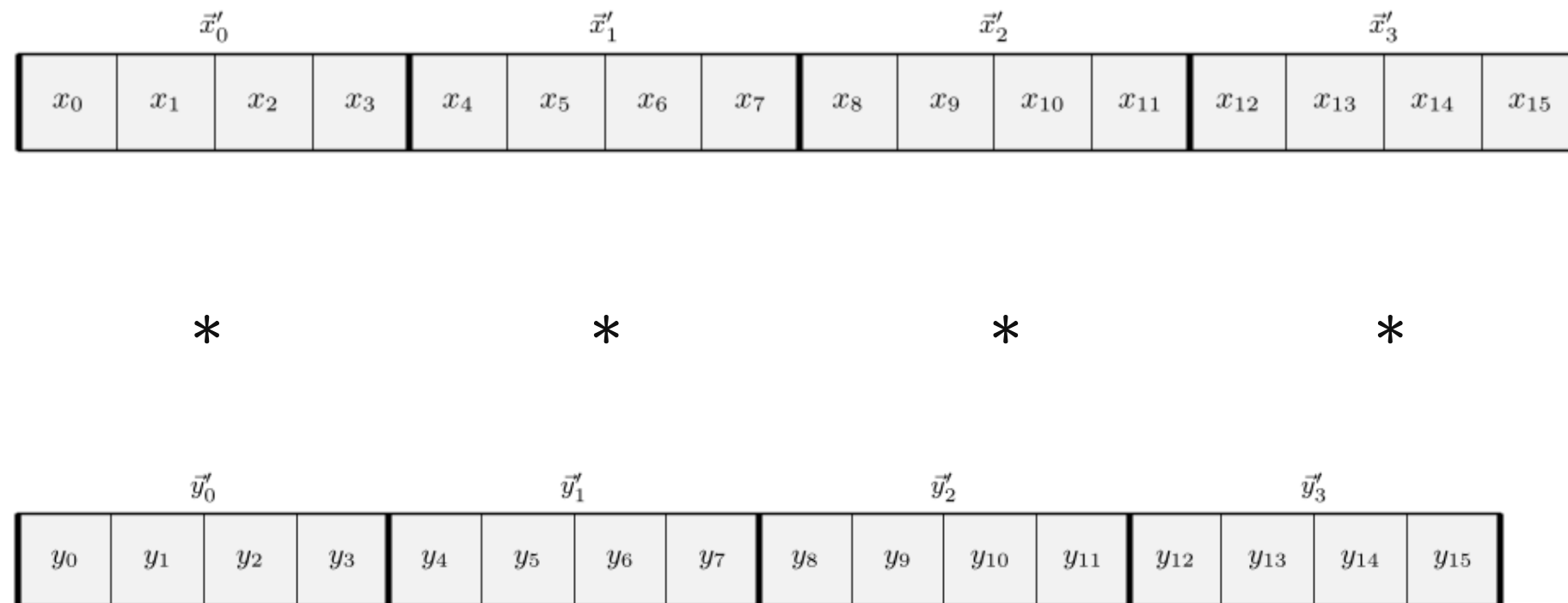
- **Pack** k bits ($x_{0..3}$) \rightarrow single **field element** in \mathbb{F}_{2^m}
 - Compute on field element (packed values)
 - Enable **SIMD**-style computation, as Ascon was designed for

Optimization - Idea: Packing



- **Pack** k bits ($x_{0..3}$) \rightarrow single **field element** in \mathbb{F}_{2^m}
 - Compute on field element (packed values)
 - Enable **SIMD**-style computation, as Ascon was designed for
Advantage: Amortize communication over packed values

Optimization - SIMD Calculation



Optimization - SIMD Calculation

=

$\vec{x}'_0 * \vec{y}'_0$				$\vec{x}'_1 * \vec{y}'_1$				$\vec{x}'_2 * \vec{y}'_2$				$\vec{x}'_3 * \vec{y}'_3$			
$x_0 \cdot y_0$	$x_1 \cdot y_1$	$x_2 \cdot y_2$	$x_3 \cdot y_3$	$x_4 \cdot y_4$	$x_5 \cdot y_5$	$x_6 \cdot y_6$	$x_7 \cdot y_7$	$x_8 \cdot y_8$	$x_9 \cdot y_9$	$x_{10} \cdot y_{10}$	$x_{11} \cdot y_{11}$	$x_{12} \cdot y_{12}$	$x_{13} \cdot y_{13}$	$x_{14} \cdot y_{14}$	$x_{15} \cdot y_{15}$

Optimization - RMFEs

Reverse Multiplication-Friendly Embeddings:

Optimization - RMFEs

Reverse Multiplication-Friendly Embeddings:

- Introduced by Cascudo et al. in CRYPTO'18 to **amortize multiple executions** of a single circuit

Optimization - RMFEs

Reverse Multiplication-Friendly Embeddings:

- Introduced by Cascudo et al. in CRYPTO'18 to **amortize multiple executions** of a single circuit
- **Leverage** sizeable **extension fields** necessary in Boolean Shamir Secret-Sharing

Optimization - RMFEs

Reverse Multiplication-Friendly Embeddings:

- Introduced by Cascudo et al. in CRYPTO'18 to **amortize multiple executions** of a single circuit
- **Leverage** sizeable **extension fields** necessary in Boolean Shamir Secret-Sharing
- **Imitate** a **homomorphism** between \mathbb{F}_2^k and \mathbb{F}_{2^k} using mappings ϕ, ψ

Optimization - RMFEs

Reverse Multiplication-Friendly Embeddings:

- Introduced by Cascudo et al. in CRYPTO'18 to **amortize multiple executions** of a single circuit
- **Leverage** sizeable **extension fields** necessary in Boolean Shamir Secret-Sharing
- **Imitate** a **homomorphism** between \mathbb{F}_2^k and \mathbb{F}_{2^k} using mappings ϕ, ψ
 - Real world: packing to \mathbb{F}_{2^m} with $m = O(k)$

Optimization - RMFEs

Reverse Multiplication-Friendly Embeddings:

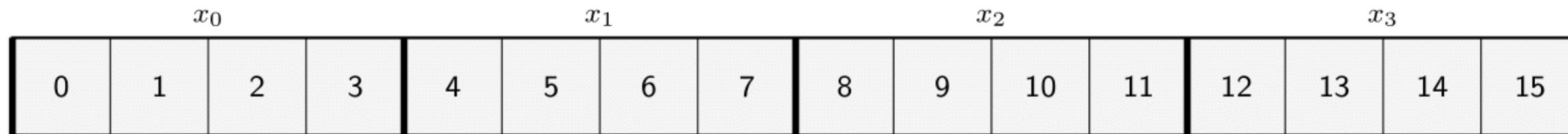
- Introduced by Cascudo et al. in CRYPTO'18 to **amortize multiple executions** of a single circuit
- **Leverage** sizeable **extension fields** necessary in Boolean Shamir Secret-Sharing
- **Imitate** a **homomorphism** between \mathbb{F}_2^k and \mathbb{F}_{2^k} using mappings ϕ, ψ
 - Real world: packing to \mathbb{F}_{2^m} with $m = O(k)$
- If maximizing packing, instead of **320** bit AND gates, process **5 AND** gates for Ascon S-Box

Optimization - Bit Rotations

$$S_0 \ggg 5$$

Optimization - Bit Rotations

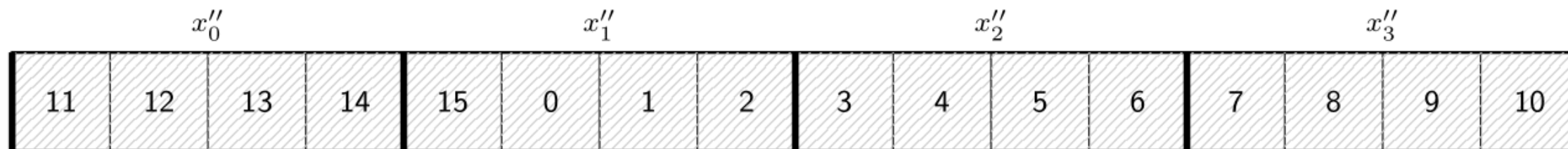
$$S_0 \ggg 5$$



- In Literature: General Purpose **Network Routing**

Optimization - Bit Rotations

$$S_0 \ggg 5$$



- In Literature: General Purpose **Network Routing**
- Our Approach: Inter-Packing, simpler, **less communication** required due to **specificity**

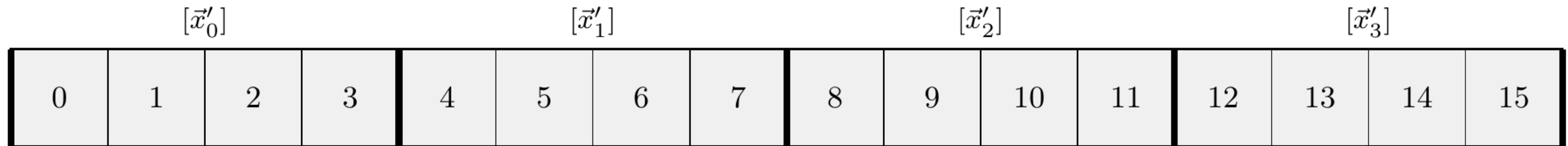
Technical Challenge – Bit Rotations

$$S_0 \ggg 5$$

Technical Challenge – Bit Rotations

$$S_0 \ggg 5$$

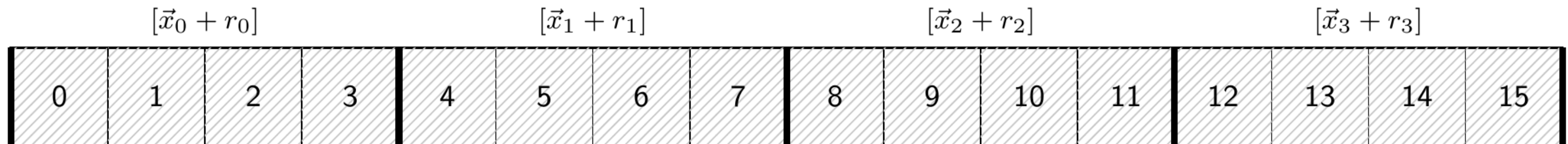
Original state-word (secret-shared):



Technical Challenge – Bit Rotations

$$S_0 \ggg 5$$

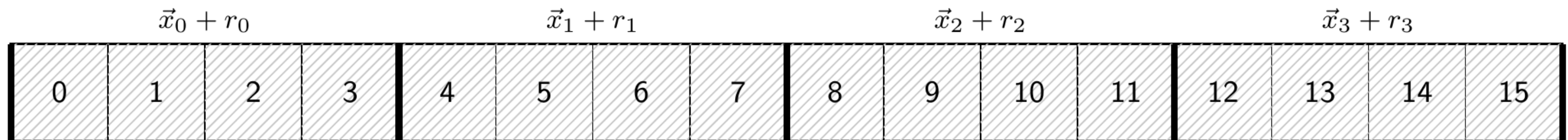
Blinded (secret-shared):



Technical Challenge – Bit Rotations

$$S_0 \ggg 5$$

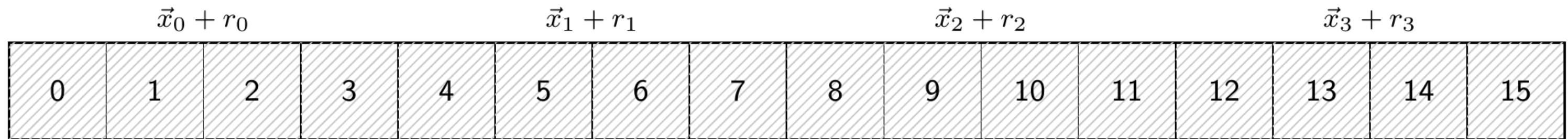
Reconstructed (blinded, not secret-shared):



Technical Challenge – Bit Rotations

$$S_0 \ggg 5$$

Unpacked (blinded, not secret-shared):



Technical Challenge – Bit Rotations

$$S_0 \ggg 5$$

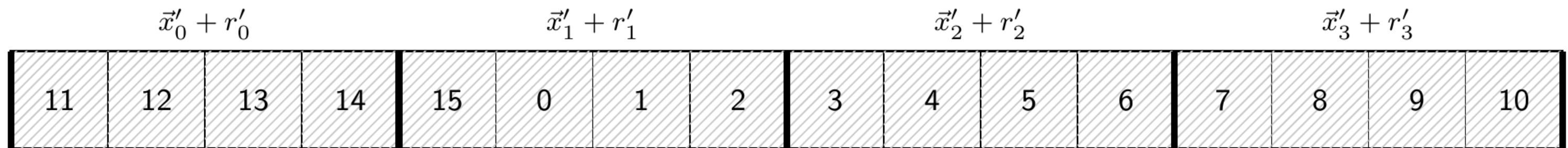
Rotated (unpacked, blinded, not secret-shared):

11	12	13	14	15	0	1	2	3	4	5	6	7	8	9	10
----	----	----	----	----	---	---	---	---	---	---	---	---	---	---	----

Technical Challenge – Bit Rotations

$$S_0 \ggg 5$$

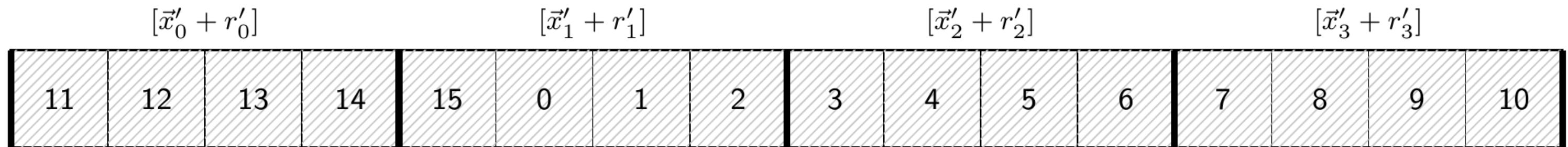
Repacked (rotated, blinded, not secret-shared):



Technical Challenge – Bit Rotations

$$S_0 \ggg 5$$

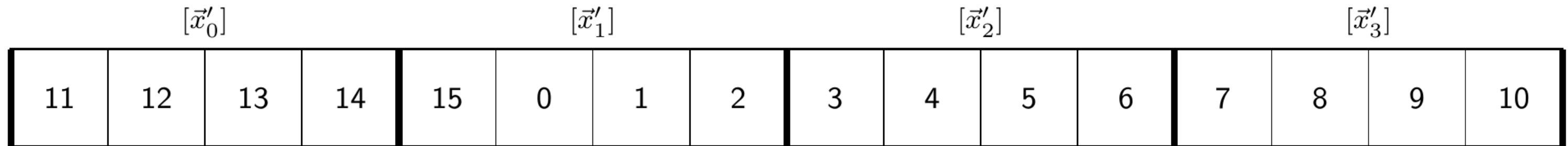
Secret-shared (rotated, blinded):



Technical Challenge – Bit Rotations

$$S_0 \ggg 5$$

Unblinded Rotated state-word (secret-shared):

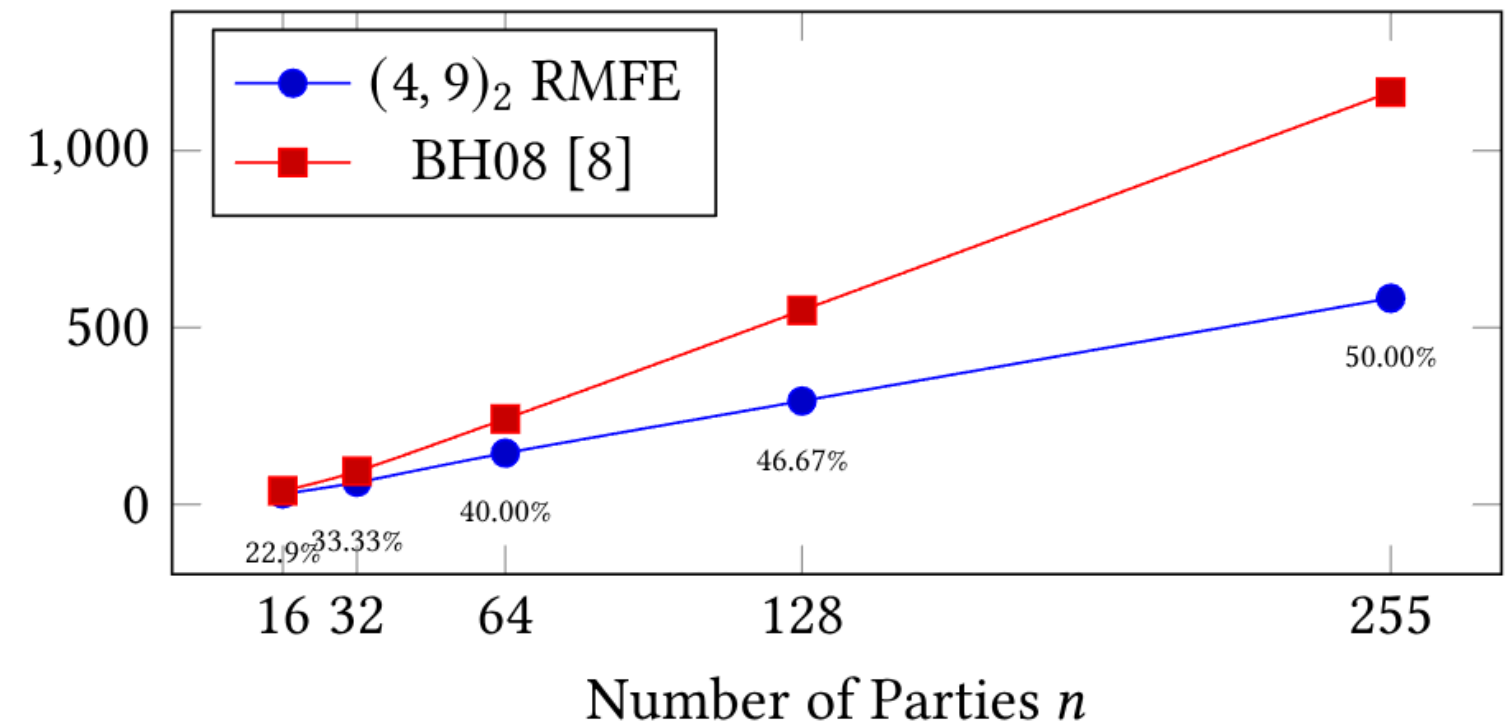
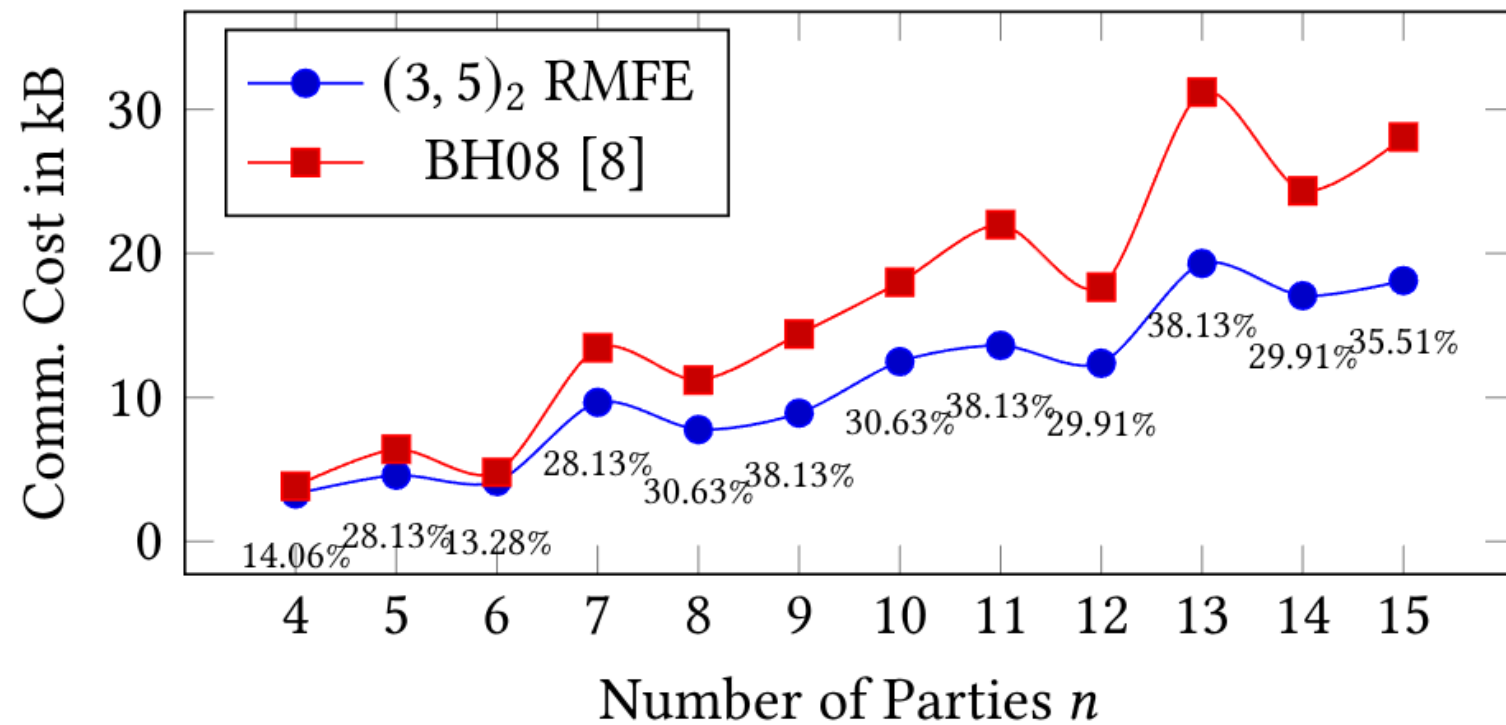


Overview

- ~~Introduction~~
- ~~Ascon~~
- ~~Baseline & Optimization~~
- **Results & Discussion**

Results: Ascon Permutation Online Communication Improvements

One Ascon round:



Discussion: Packing sizes

Larger **packing** sizes (e.g. **64-bit**) not worth it, due to:

Discussion: Packing sizes

Larger **packing** sizes (e.g. **64-bit**) not worth it, due to:

- Security setting and ensuing **reconstruction protocol**

Discussion: Packing sizes

Larger **packing** sizes (e.g. **64-bit**) not worth it, due to:

- Security setting and ensuing **reconstruction protocol**
 - Big **RMFEs "cannibalize" reconstruction-protocol amortization**

Discussion: Packing sizes

Larger **packing** sizes (e.g. **64-bit**) not worth it, due to:

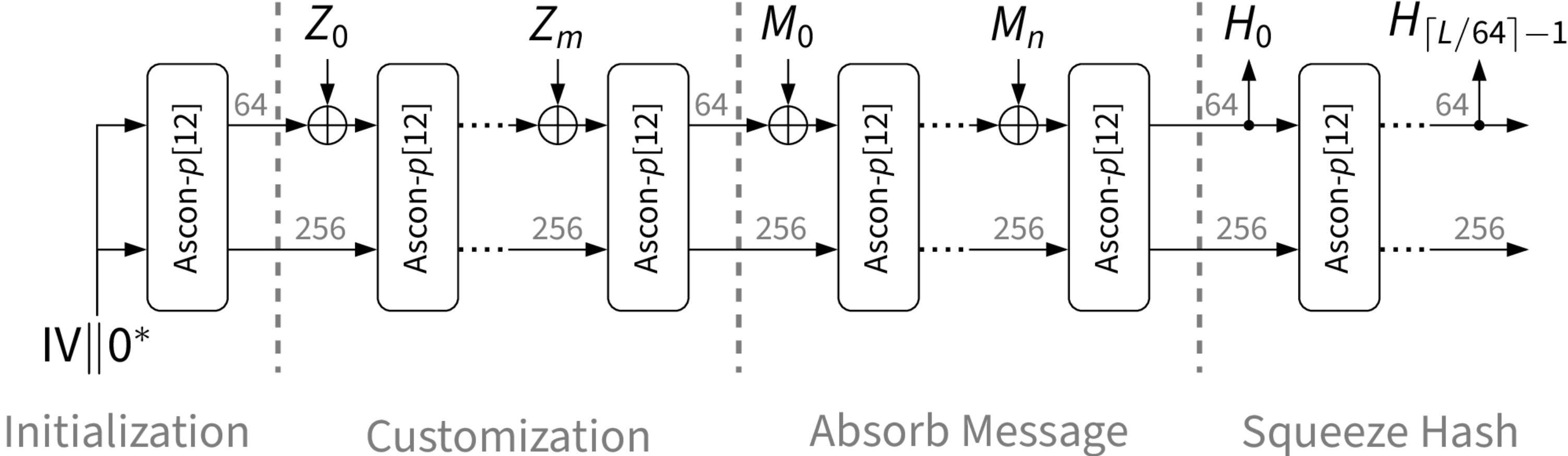
- Security setting and ensuing **reconstruction protocol**
 - Big **RMFEs "cannibalize" reconstruction-protocol amortization**
- **Worse packing rate/Extension field mismatch** (1.66 for $(3, 5)_2$ vs 3.36 for $(64, 215)_2$)

Discussion: Packing sizes

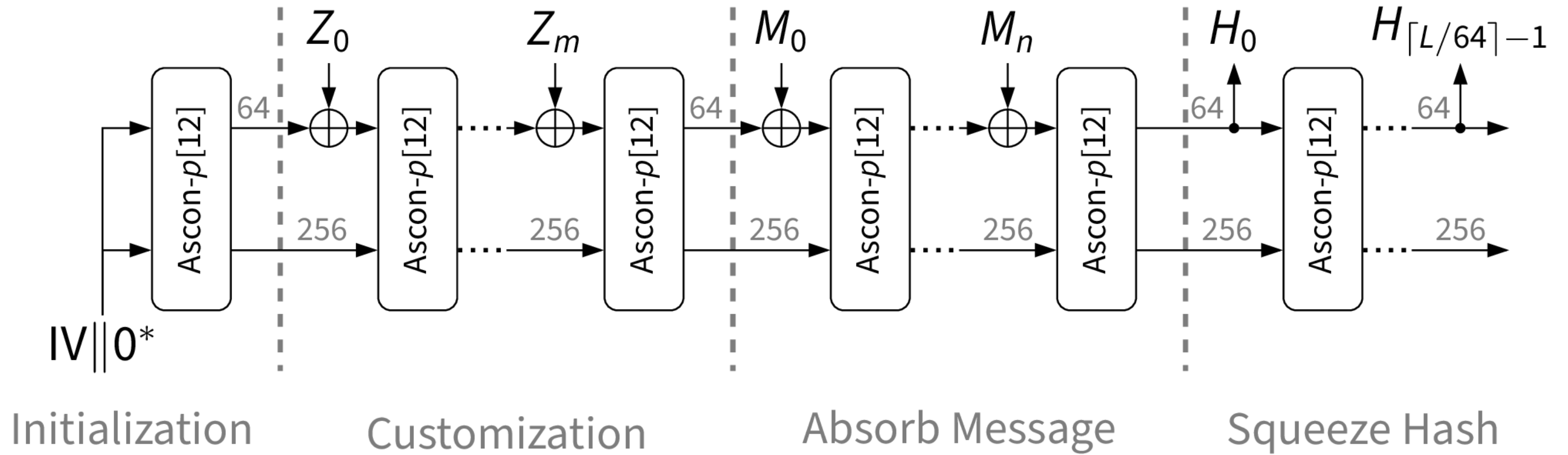
Larger **packing** sizes (e.g. **64-bit**) not worth it, due to:

- Security setting and ensuing **reconstruction protocol**
 - Big **RMFEs "cannibalize" reconstruction-protocol amortization**
- **Worse packing rate**/Extension field mismatch (1.66 for $(3, 5)_2$ vs 3.36 for $(64, 215)_2$)
- Linear Layer overhead

Results: Ascon-Hash, Ascon[C]XOF

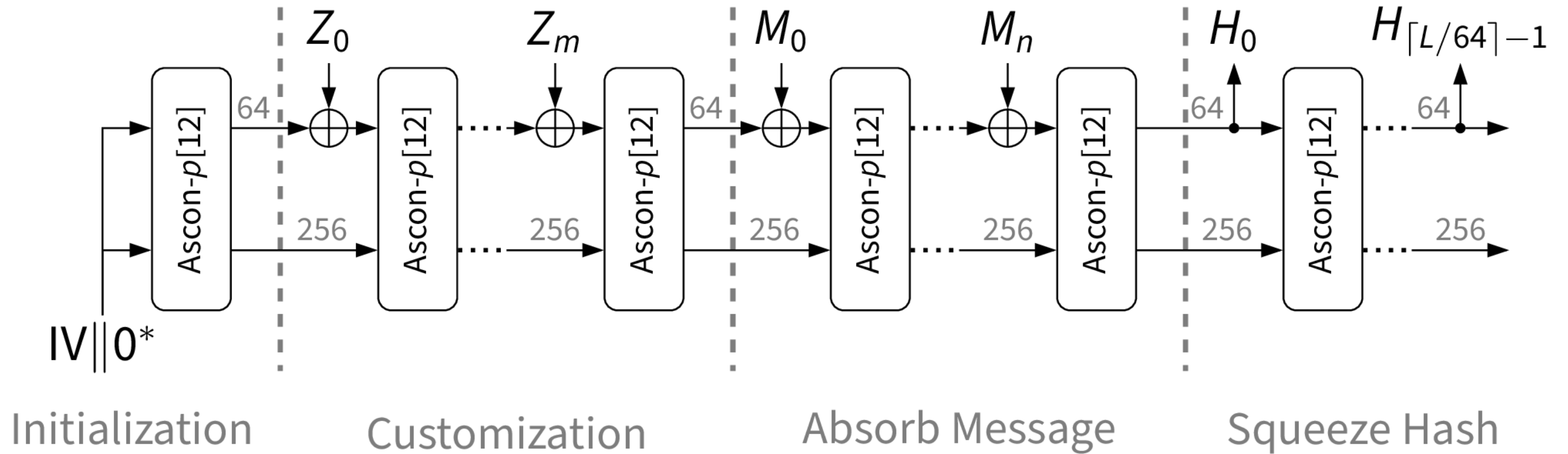


Results: Ascon-Hash, Ascon[C]XOF



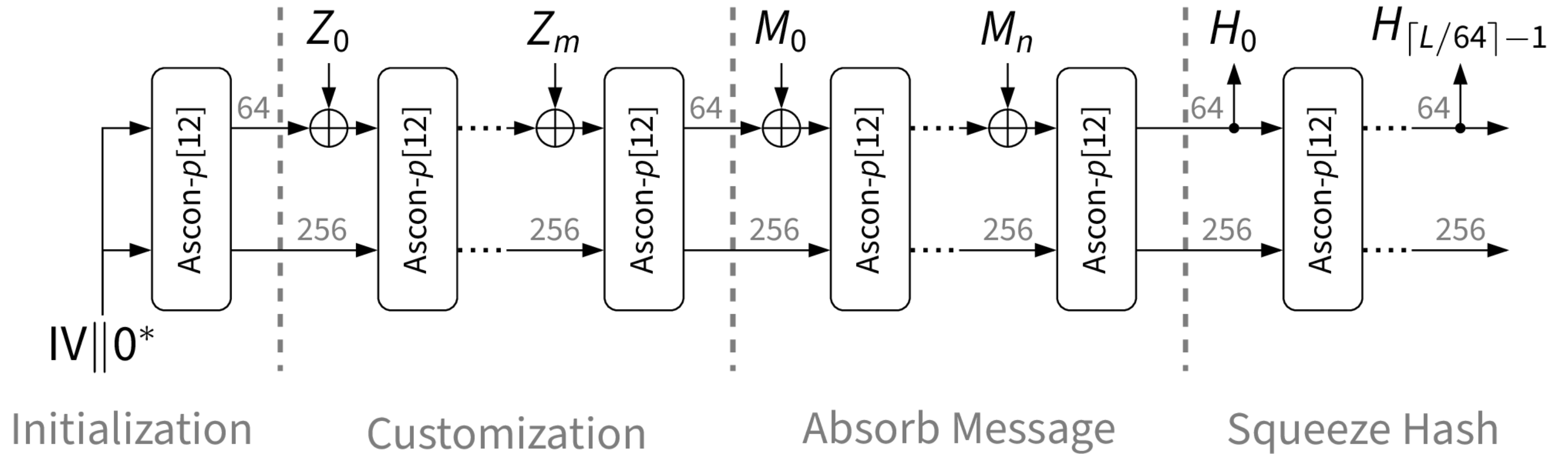
- **Differences:**

Results: Ascon-Hash, Ascon[C]XOF



- **Differences:**
 - Rate: **64** bit (Hash) vs. **128** bit/block (AEAD)

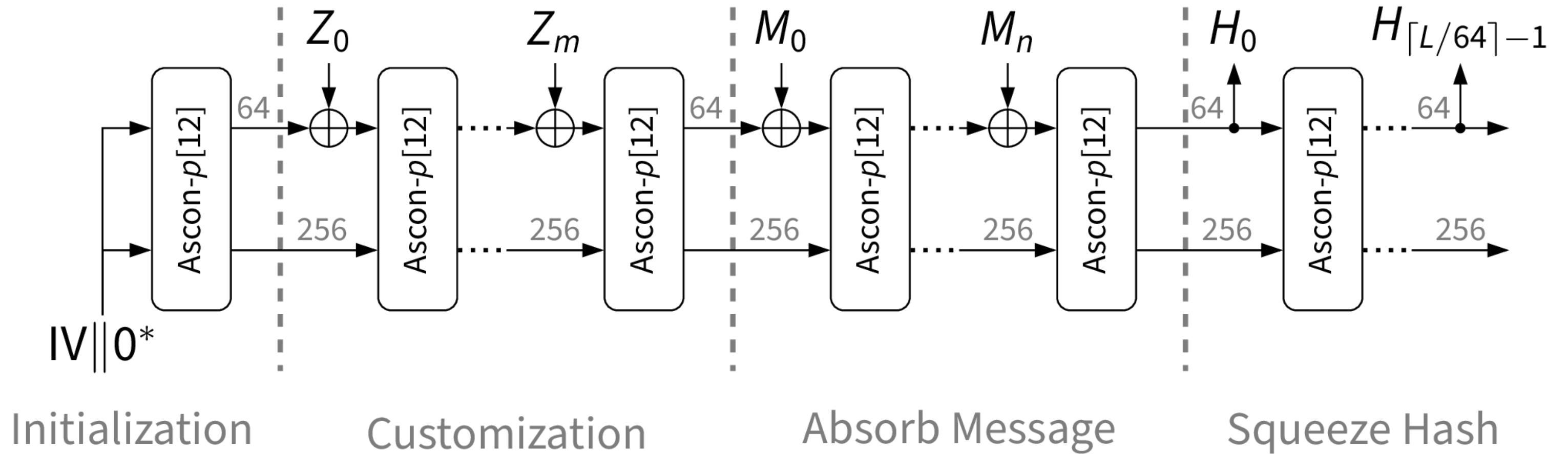
Results: Ascon-Hash, Ascon[C]XOF



- **Differences:**

- Rate: **64** bit (Hash) vs. **128** bit/block (AEAD)
- Rounds per block: **12** (Hash) vs. **8** (AEAD)

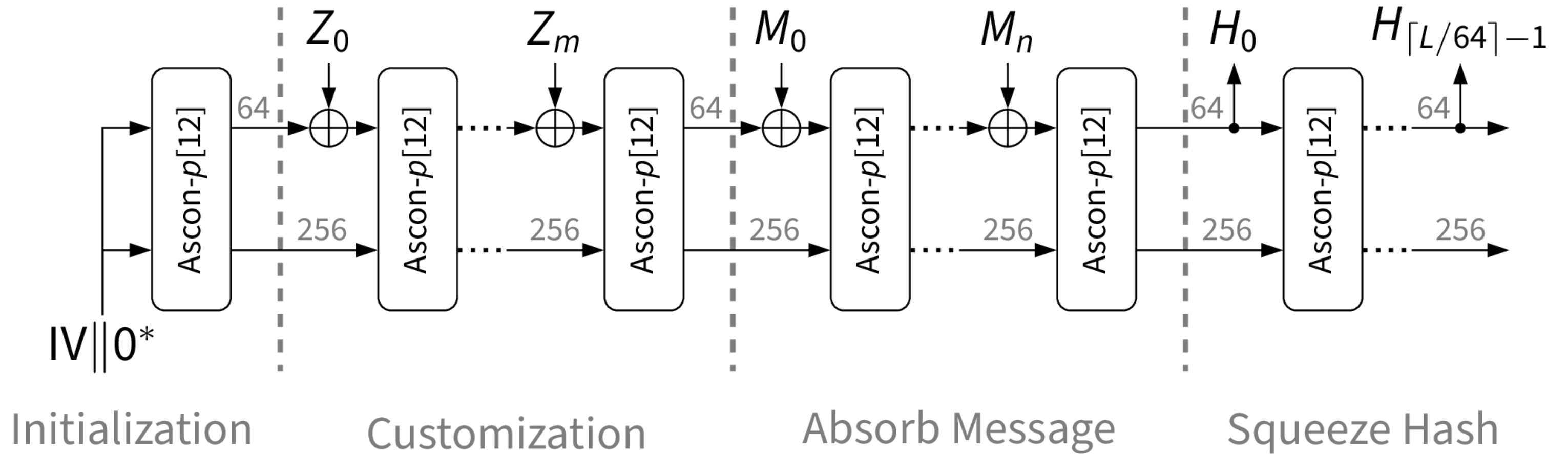
Results: Ascon-Hash, Ascon[C]XOF



- **Differences:**

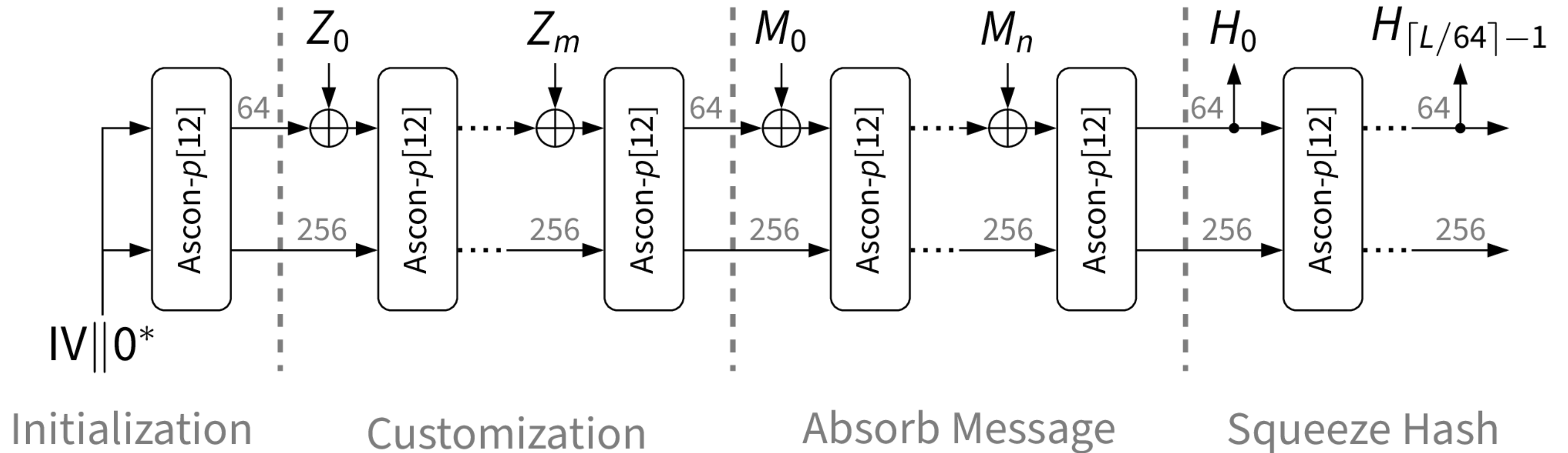
- Rate: **64** bit (Hash) vs. **128** bit/block (AEAD)
- Rounds per block: **12** (Hash) vs. **8** (AEAD)
- Worse throughput in bits (absolute) Hash vs. AEAD

Results: Ascon-Hash, Ascon[C]XOF



- **Differences:**
 - Rate: **64** bit (Hash) vs. **128** bit/block (AEAD)
 - Rounds per block: **12** (Hash) vs. **8** (AEAD)
 - Worse throughput in bits (absolute) Hash vs. AEAD
- **Same: Improvement** over baseline

Results: Ascon-Hash, Ascon[C]XOF



- **Differences:**
 - Rate: **64** bit (Hash) vs. **128** bit/block (AEAD)
 - Rounds per block: **12** (Hash) vs. **8** (AEAD)
 - Worse throughput in bits (absolute) Hash vs. AEAD
- **Same: Improvement** over baseline
 - Reason: **optimizing** same **permutation**

Conclusion

Conclusion

- **First** optimization study for NIST lightweight standard in MPC

Conclusion

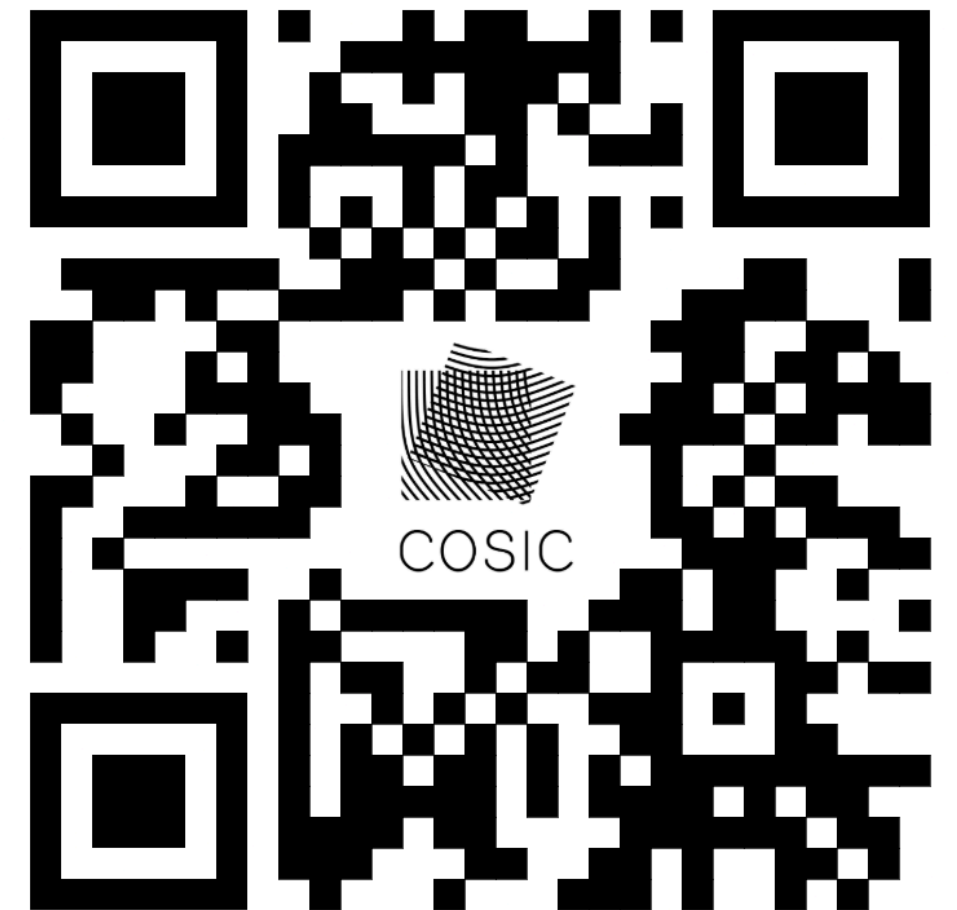
- **First** optimization study for NIST lightweight standard in MPC
- **Real-World Application** of RMFEs

Conclusion

- **First** optimization study for NIST lightweight standard in MPC
- **Real-World Application** of RMFEs
- Future Work: Preprocessing, extension to other ciphers

Conclusion

- **First** optimization study for NIST lightweight standard in MPC
 - **Real-World Application** of RMFEs
 - Future Work: Preprocessing, extension to other ciphers
-
- **Paper:** "Evaluating Ascon in Secure Multi-Party Computation using Reverse Multiplication-Friendly Embeddings"
 - **Contact:** peter.schwarz@esat.kuleuven.be
 - **Full Paper:** ia.cr/2025/1538



Questions?