On Implementing ChaCha on a Programmable Switch

EuroP4 Workshop 2022

Dec. 9, 2022

Yutaro Yoshinaka, Junji Takemasa, Yuki Koizumi, Toru Hasegawa
(Osaka University)

Motivation and Existing Approaches

- Motivation: practical cryptographic primitive on hardware programmable switches
  - Application:
    - Privacy and anti-censorship (PINOT [1], PHI [2])
    - Countermeasure for traffic analysis (ditto [3])
    - IoT and 5G security [4]
    - Onion Routing
  - Desirable properties:
    - **Security** – probabilistic encryption, sufficient key size (128 or 256 bits)
    - **Speed** – should reduce recirculations for throughput
    - **Applicability** – support for long message input

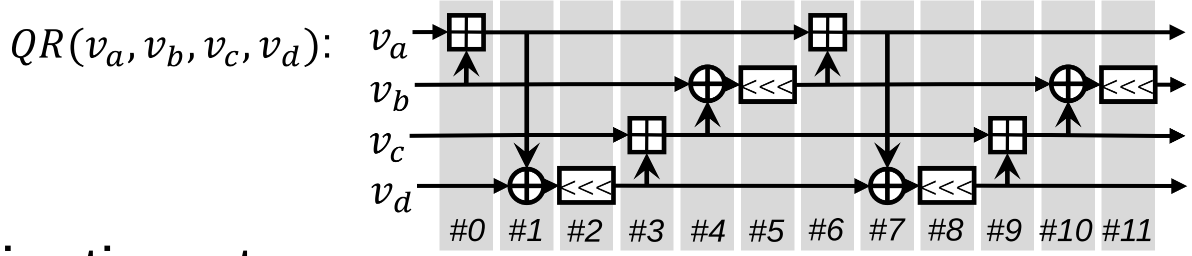
- Existing approaches
  - AES-Tofino [5]
    - Supporting only single-block (16B), deterministic encryption
    - Consumes 15% SRAM resources (57k entries) of switch for constructing LUT
  - Two-round Even-Mansour cipher [1], OTP with HalfSipHash [2,6]
    - Short key size (64 bits)
    - Considers short message only (~8B)

Our Approach

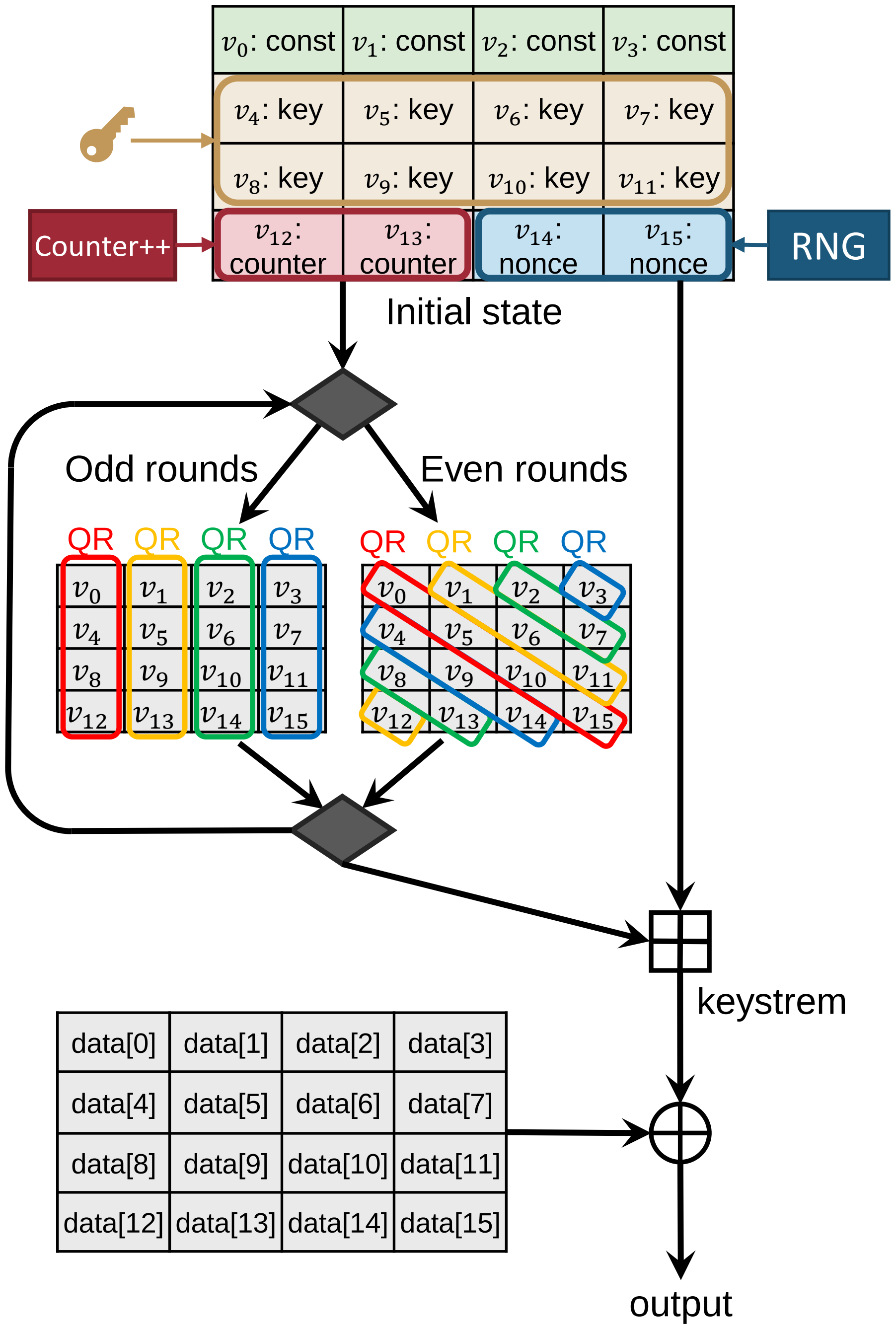
- ChaCha [7]: stream cipher
  - Highly portable for pipelined architecture, especially programmable switches
    - Operation for encryption and decryption are identical
    - **ARX-based** cipher (consists by only addition, rotation, and XOR)
    - Free from LUT and S-box, and requires **small memory footprint**
    - Keeps **small internal state** and is **in-place** algorithm
    - Natively supports **probabilistic** and **multi-block** encryption/decryption
    - No key schedule
  - Sufficient key size (256 bits), and no attacks found for 8+ rounds
  - Adopted in famous protocols and applications (e.g., TLS 1.3, QUIC, OpenSSH, WireGuard, Adiantum)

ChaCha

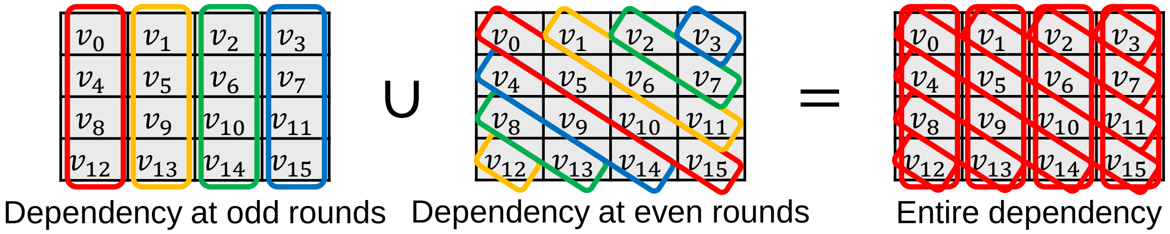
- **Initialization step**
  - Generate 128-bit nonce (for primary block)
  - Increment counter (for non-primary block)
  - Initialize 512-bit state with key, counter, and nonce
- **Round step**
  - Shuffle the state by performing four Quarter Rounds
  - Repeat 20, 12, or 8 rounds
    - Odd round: apply four QRs on columns
    - Even round: apply four QRs on diagonals



- **Finalization step**
  - Obtain keystream by adding initial state and shuffled state
- **Encryption/decryption step**
  - Take XOR of message and keystream

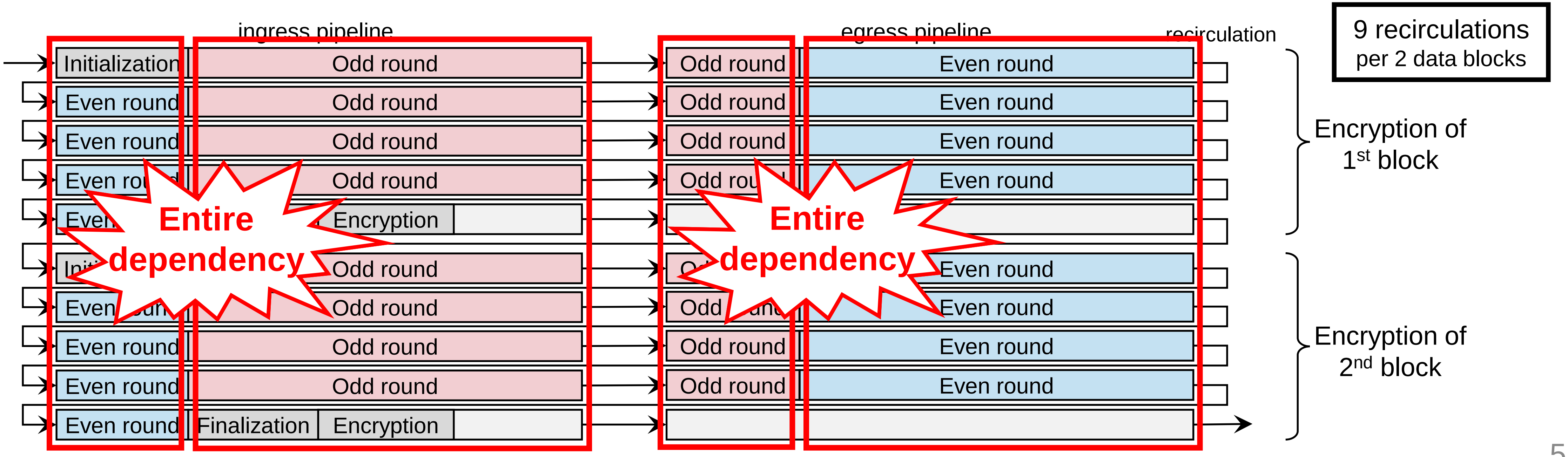


Challenge and Design



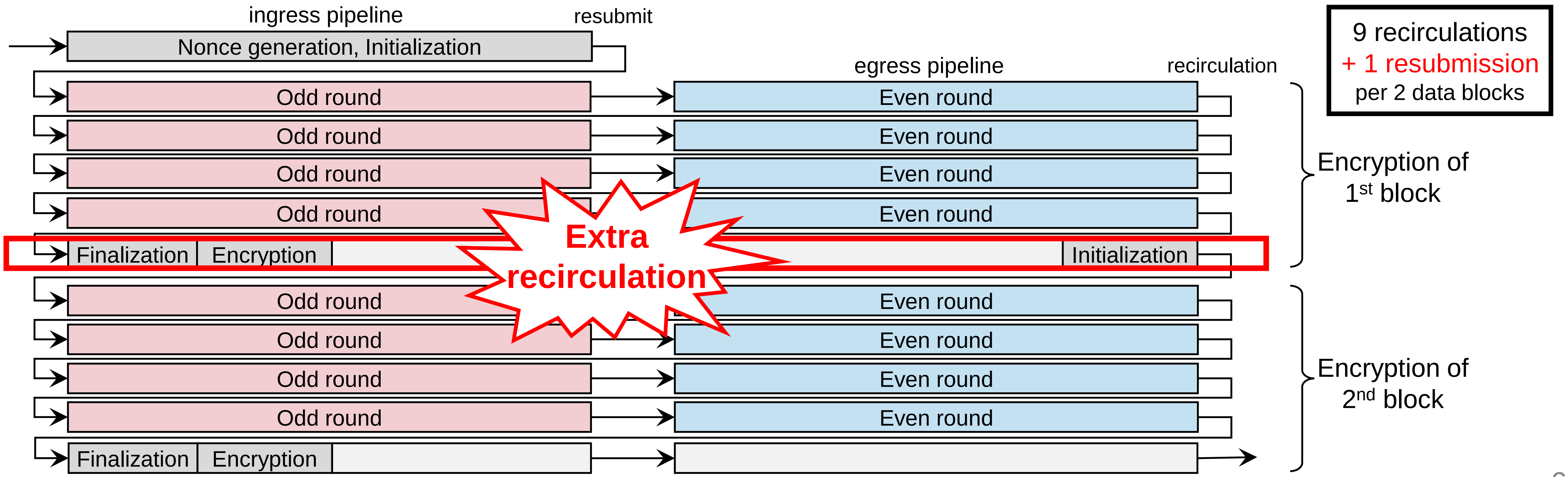
☐ Challenge: Dependency among QRs

- **Action dependencies** between consecutive rounds, not between four QRs in a round
  - ➢ Deploy round function to 12 stages by performing four QRs in parallel leveraging VLIW architecture
- **Variable dependencies** in QRs precludes implementing odd and even rounds on a single pipeline
  - ➢ Place odd and even rounds on ingress and egress pipeline exclusively
  - ➢ Generate nonce in ingress pipeline in the first pass, followed by resubmission and round operaions



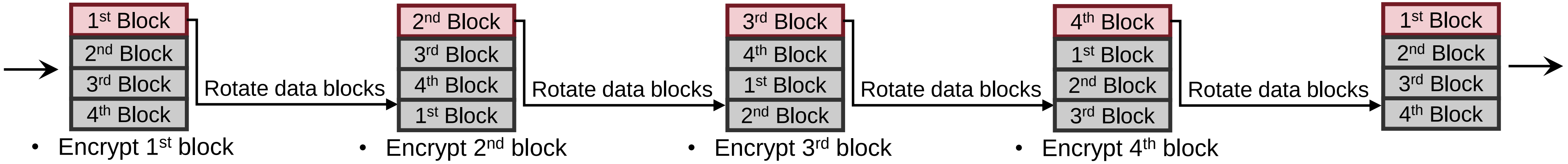
Challenge and Design

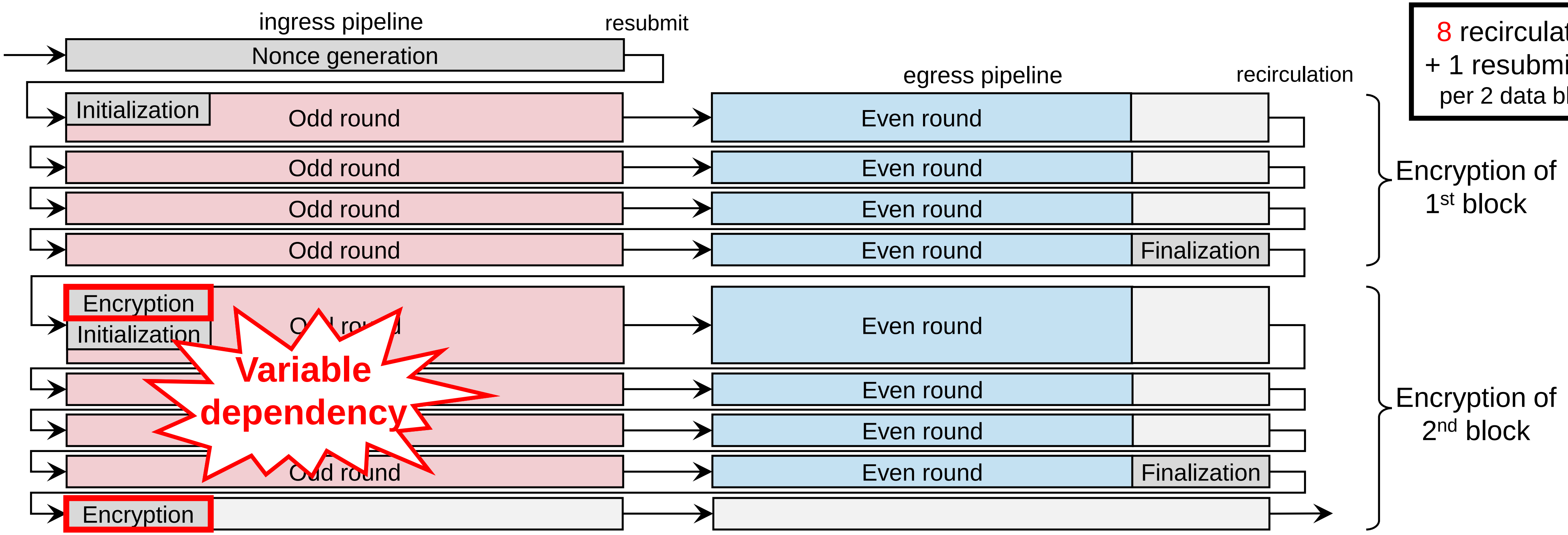
- Dependency among operations in QR
  - **Action dependencies** in Round step (**12 steps**) fully occupies Tofino's pipelines (**12 stages**)
    - ➢ Optimize the implementation of QRs in even rounds for 11 stages, rather than 12 stages
      - Tofino's special instruction makes it possible to execute rotation and addition in one stage
    - ➢ Place Finalization step on the last stage of egress pipeline
    - ➢ Superimpose Initialization and Encryption/decryption step on the first step of odd rounds



Challenge and Design

- Encrypting/decrypting multiple blocks
  - **Variable dependencies** between keystream and all data blocks preclude the implementation
    - ➢ Regards data blocks as circular buffer
    - ➢ Always encrypts block at the head position in PHV, and rotates data blocks in ingress parser

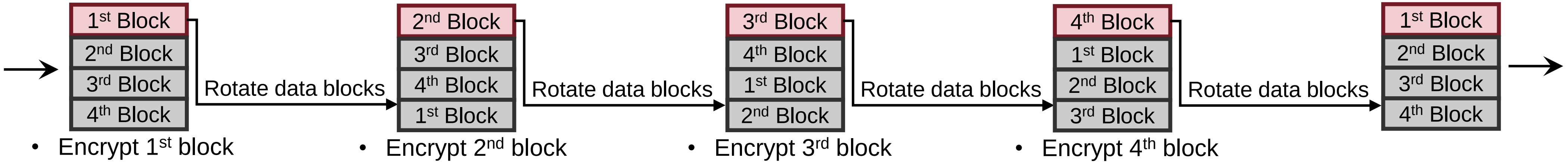


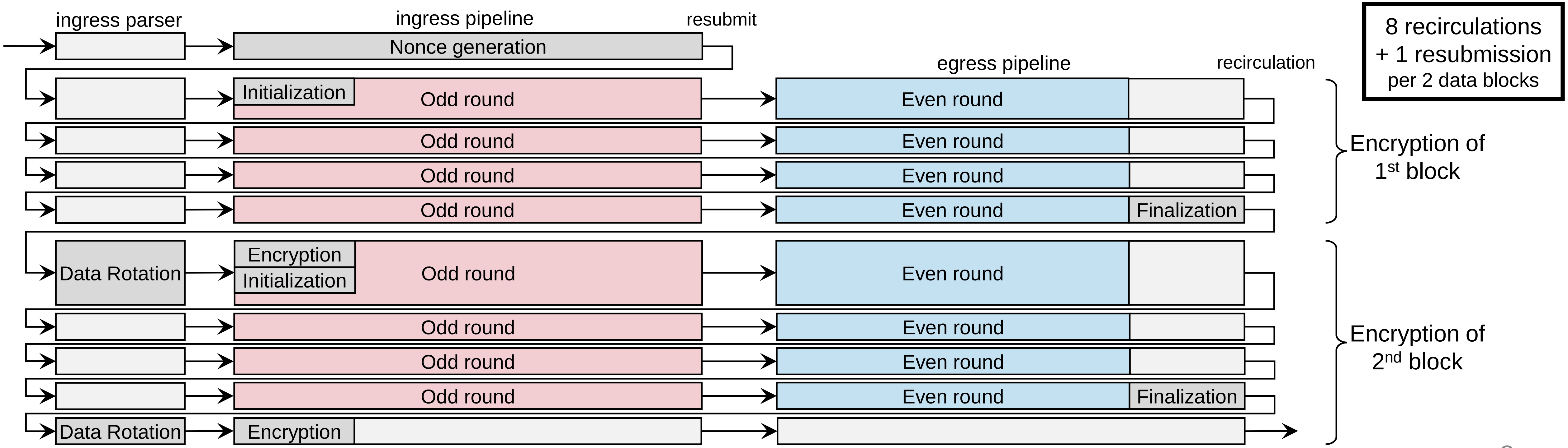


8 recirculations
+ 1 resubmission
per 2 data blocks

Challenge and Design

- Encrypting/decrypting multiple blocks
  - **Variable dependencies** between keystream and all data blocks preclude the implementation
    - ➢ Regards data blocks as circular buffer
    - ➢ Always encrypts block at the head position in PHV, and rotates data blocks in ingress parser



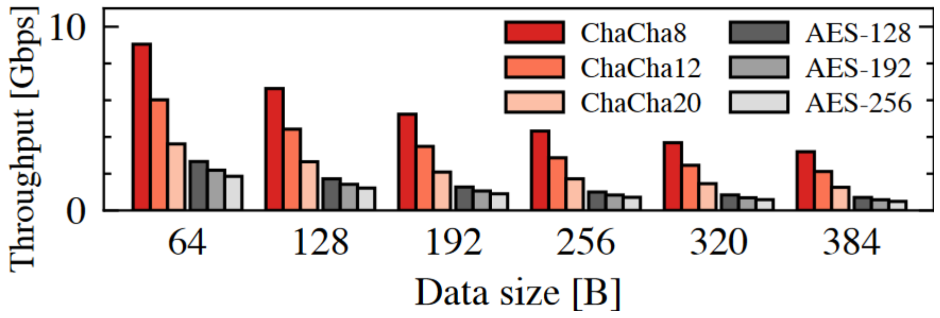


Evaluation

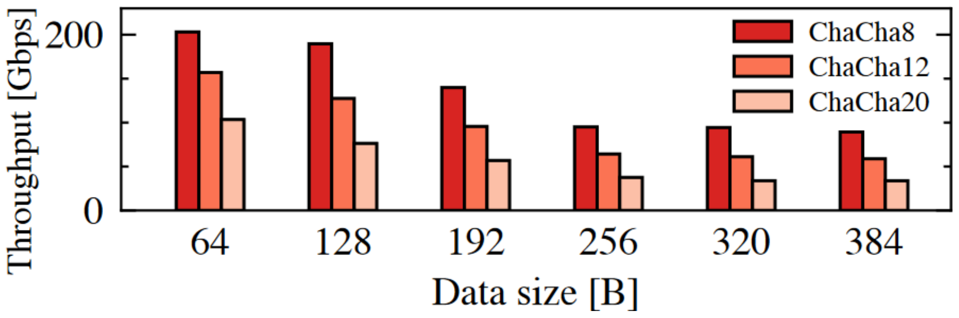
- Comparison to AES-Tofino [5] (with one recirculation port)
  - 8 and 12-round ChaCha are **3+ ~ 4+ times** faster than AES-128 and -256, resp
  (If AES-Tofino utilizes egress pipelines, rate of speedup is 1.5+ ~ 2+)

- Maximum throughput (with 29 recirculation ports of 2-pipeline Tofino)
  - **64B data**: 8-round ChaCha achieves **203.1Gbps**
  - **256B data**: 8-round ChaCha achieves **95.1Gbps**
  - **384B data**: 8-round ChaCha achieves **89.3Gbps**

- Memory utilization
  - ChaCha utilizes only 1.35% SRAM and 1.74% TCAM (43 entries), whereas AES-Tofino utilizes 14.98% SRAM (57k entries)



Throughput with one recirculation port



Throughput with 29 recirculation ports

Conclusion

- We implement cryptographic primitive based on ChaCha on Tofino switches
- Our implementation outperforms AES-based approach in terms of throughput and small memory footprint

- Future work
  - Implementing authenticated encryption
  - Mitigating overhead of recirculations by splitting and merging packets

Thank You for Listening!

Our code is available at: https://github.com/Hasegawa-Laboratory/ChaCha-Tofino

[1] Wang, Liang, et al. "Programmable in-network obfuscation of DNS traffic." *NDSS: DNS Privacy Workshop*. 2021.

[2] Yoshinaka, Yutaro, et al. "Feasibility of Network-layer Anonymity Protocols at Terabit Speeds using a Programmable Switch." *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*. IEEE, 2022.

[3] Meier, Roland, Vincent Lenders, and Laurent Vanbever. "ditto: WAN Traffic Obfuscation at Line Rate." *NDSS Symposium 2022*. 2022.

[4] Lin, Yi-Bing, Tse-Jui Huang, and Shi-Chun Tsai. "Enhancing 5g/iot transport security through content permutation." *IEEE Access* 7 (2019): 94293-94299.

[5] Chen, Xiaoqi. "Implementing AES encryption on programmable switches via scrambled lookup tables." *Proceedings of the Workshop on Secure Programmable Network Infrastructure*. 2020.

[6] Yoo, Sophia, and Xiaoqi Chen. "Secure keyed hashing on programmable switches." *Proceedings of the ACM SIGCOMM 2021 Workshop on Secure Programmable network INfrastructure*. 2021.

[7] Bernstein, Daniel J. "ChaCha, a variant of Salsa20." *Workshop record of SASC*. Vol. 8. No. 1. 2008.