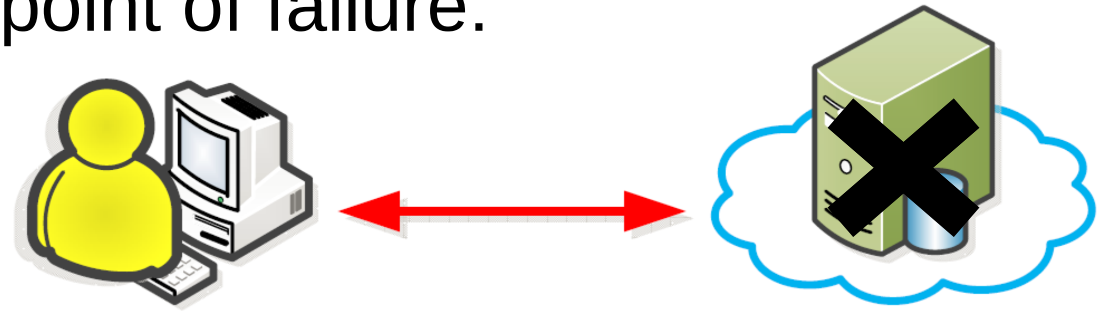
Convergent Dispersal: Toward Storage-Efficient Security in a Cloud-of-Clouds

Mingqiang Li[1], Chuan Qin[1], **Patrick P. C. Lee**[1], Jin Li[2]
[1]The Chinese University of Hong Kong, [2]Guangzhou University

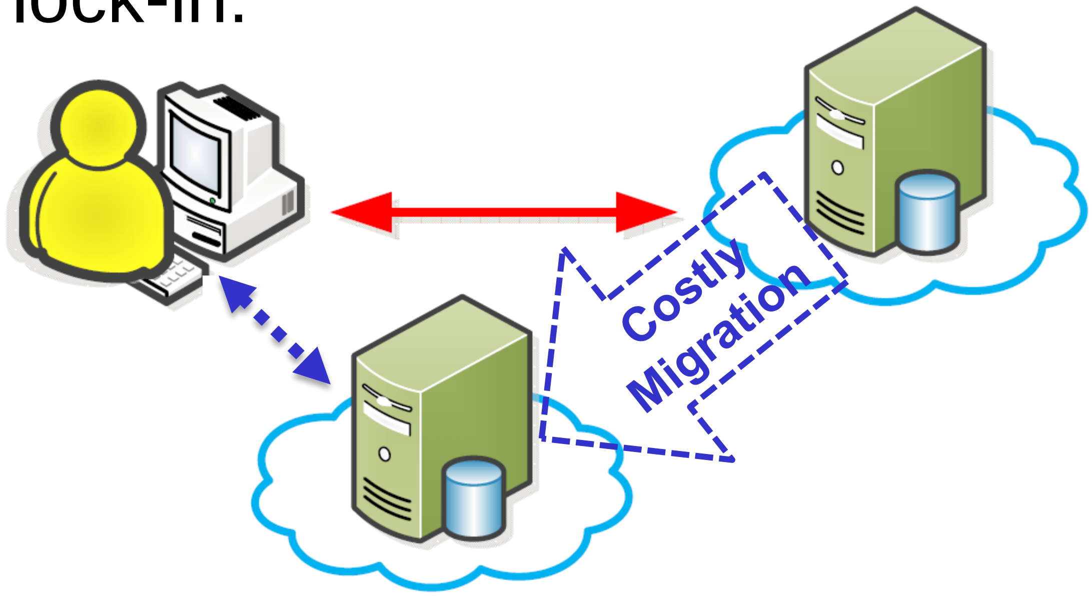
HotStorage '14

Single Cloud Problems

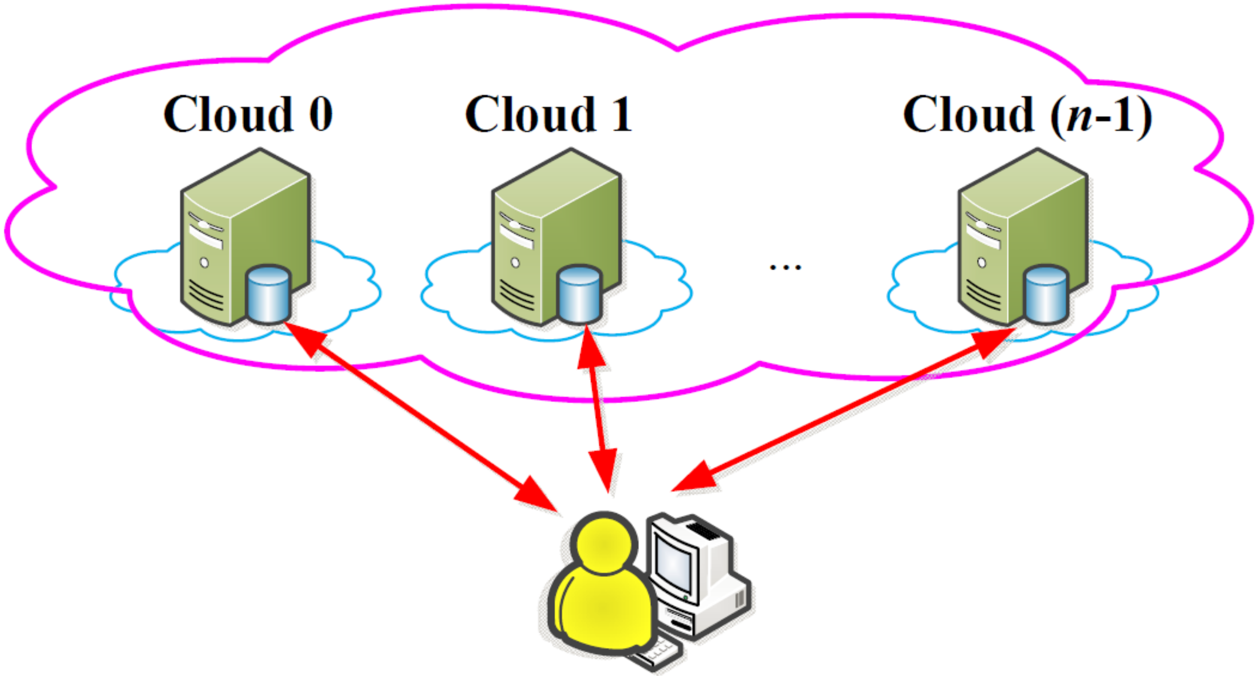
- Single point of failure:



- Vendor lock-in:



Cloud-of-Clouds

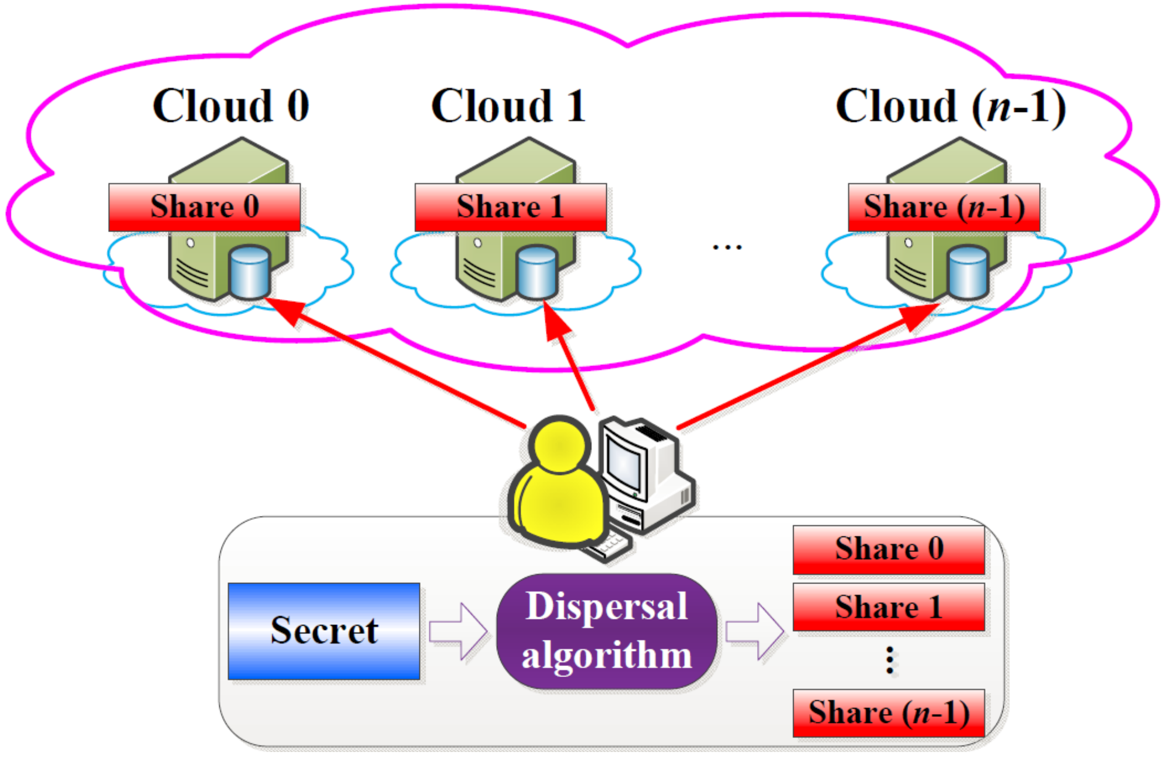


- Exploits diversity of multiple cloud storage vendors:
  - Provides fault tolerance
  - Avoids vendor lock-in
  - **Improves security**

Diversity → Security

- **Threat model**: provides data confidentiality
- Traditional encryption:
  - Encrypts data with a **key** and protects the key
  - Key management is challenging
- Leveraging diversity:
  - Disperses data across multiple clouds
  - Data remains confidential even if a subset of clouds is compromised
    - Assumption: infeasible for attackers to compromise all clouds
  - Security is achieved without keys → **keyless security**

Keyless Security

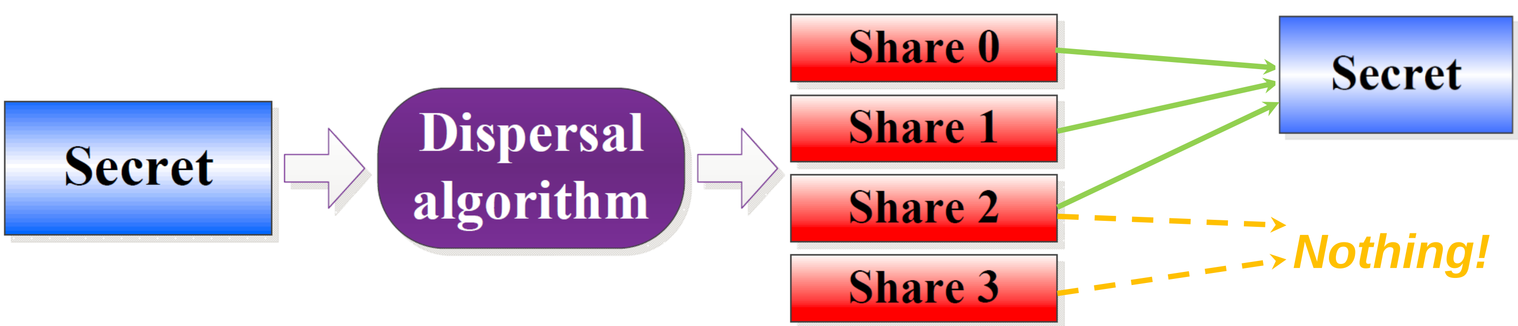


- Major building block: **dispersal algorithm**
  - Given a secret, outputs multiple shares
  - Secret remains inaccessible without enough shares

Dispersal Algorithm

- **(n, k, r)** dispersal algorithm:
  - Secret is dispersed into n shares
  - Secret can be reconstructed from any k shares ($k < n$)
  - Secret cannot be inferred (even partially) from any r shares ($r < k$)

- Example: (4, 3, 2)



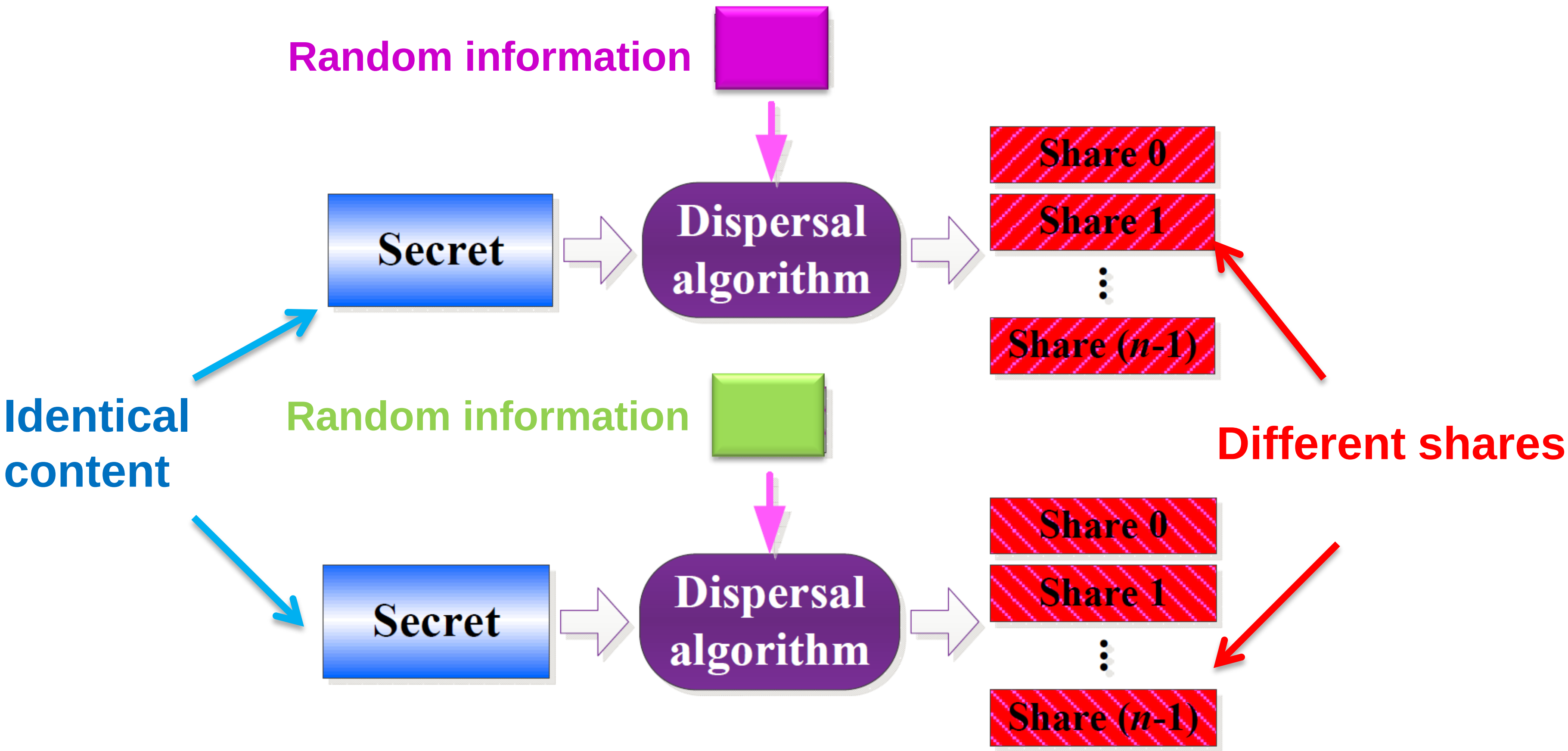
State of the Art

- Ramp secret sharing scheme (RSSS) [Blakley and Meadows, CRYPTO'84]
  - Combines Rabin's information dispersal $(r = 0)$ and Shamir's secret sharing scheme $(r = k\text{-}1)$
  - Makes tradeoff between storage space and security
- AONT-RS [Resch et al., FAST'11]
  - Combines all-or-nothing-transform and Reed-Solomon encoding
- *Main idea: embeds random information into dispersed data*

Deduplication

- Cloud storage uses **deduplication** to save cost
- Deduplication avoids storing multiple data copies with identical content
  - Saves storage space
  - Saves write bandwidth
- However, state-of-the-art dispersal algorithms break deduplication
  - Root cause: **security builds on embedded randomness**

Deduplication



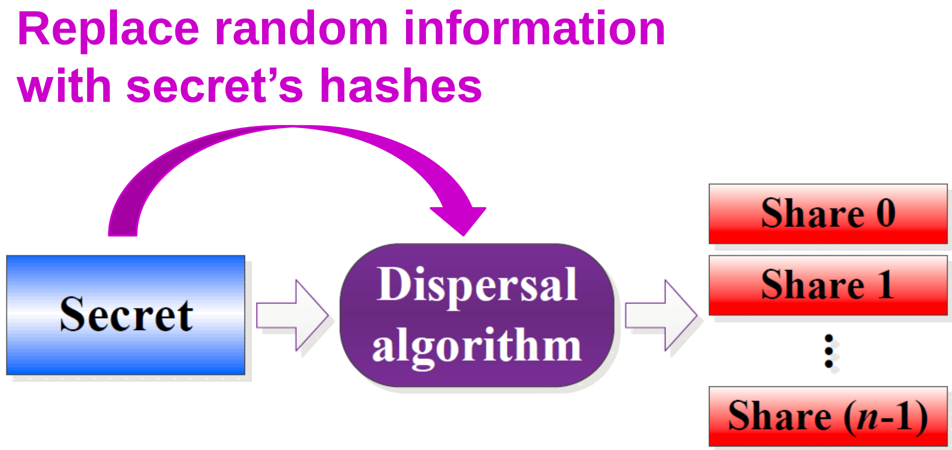
Q: Can we preserve both deduplication and keyless security in dispersal algorithms?

Our Contributions

- **Convergent Dispersal**: a data dispersal design that preserves both dedup and keyless security
  - Can be generalized for any distributed storage systems
- Two implementations:
  - CRSSS: builds on RSSS [Blakley and Meadows, CRYPTO'84]
  - CAONT-RS: builds on AONT-RS [Resch et al., FAST'11]
- Evaluation on computational performance
  - CRSSS and CAONT-RS are complementary in performance for different parameters
  - Best of CRSSS and CAONT-RS achieves ≥ 200MB/s

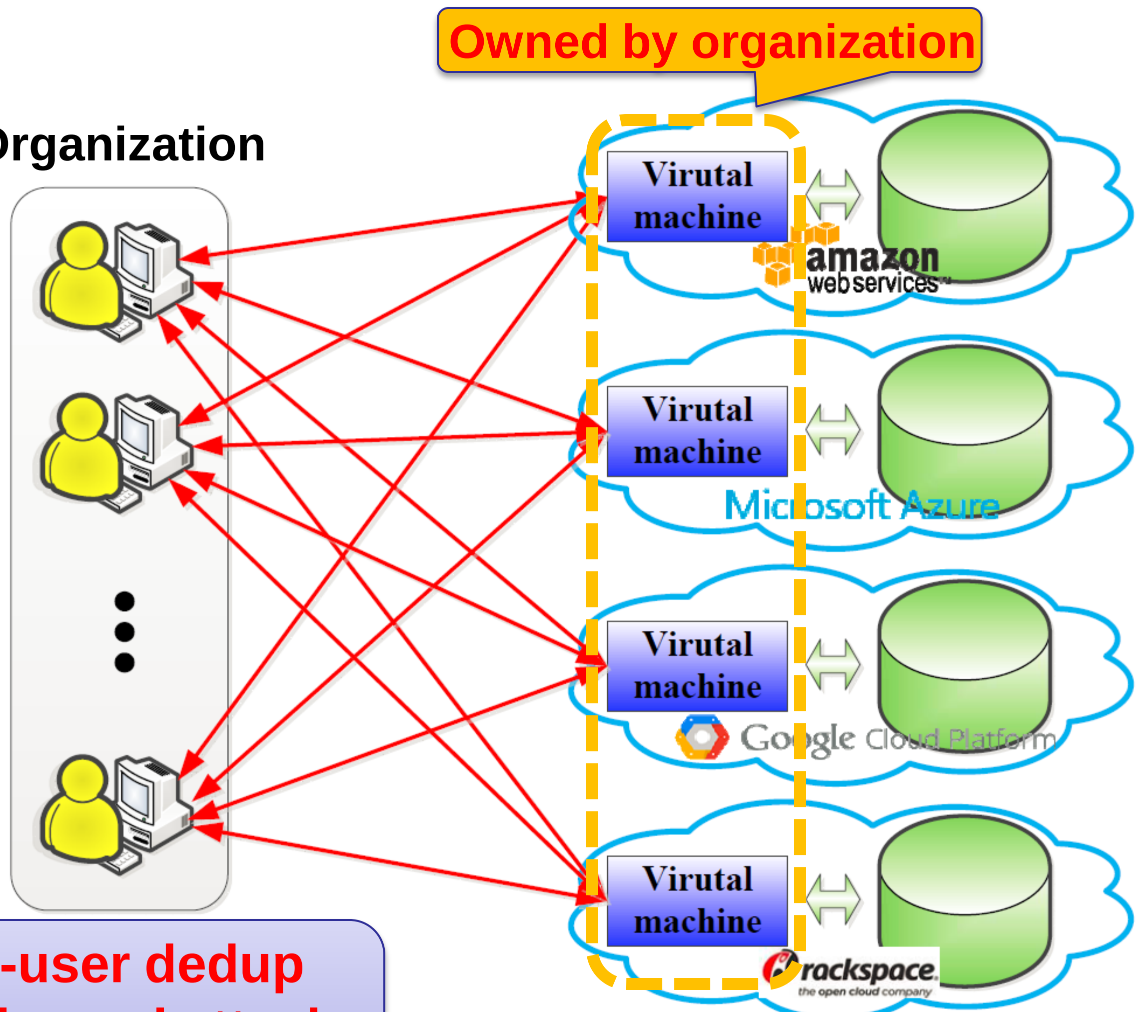
Key Idea

- Inspired by convergent encryption [Douceur et al., ICDCS'02]
  - Key is derived from cryptographic hash of the content
  - Key is deterministic: same content → same ciphertext
- Convergent dispersal:



Same secret → same shares

Deployment Scenario



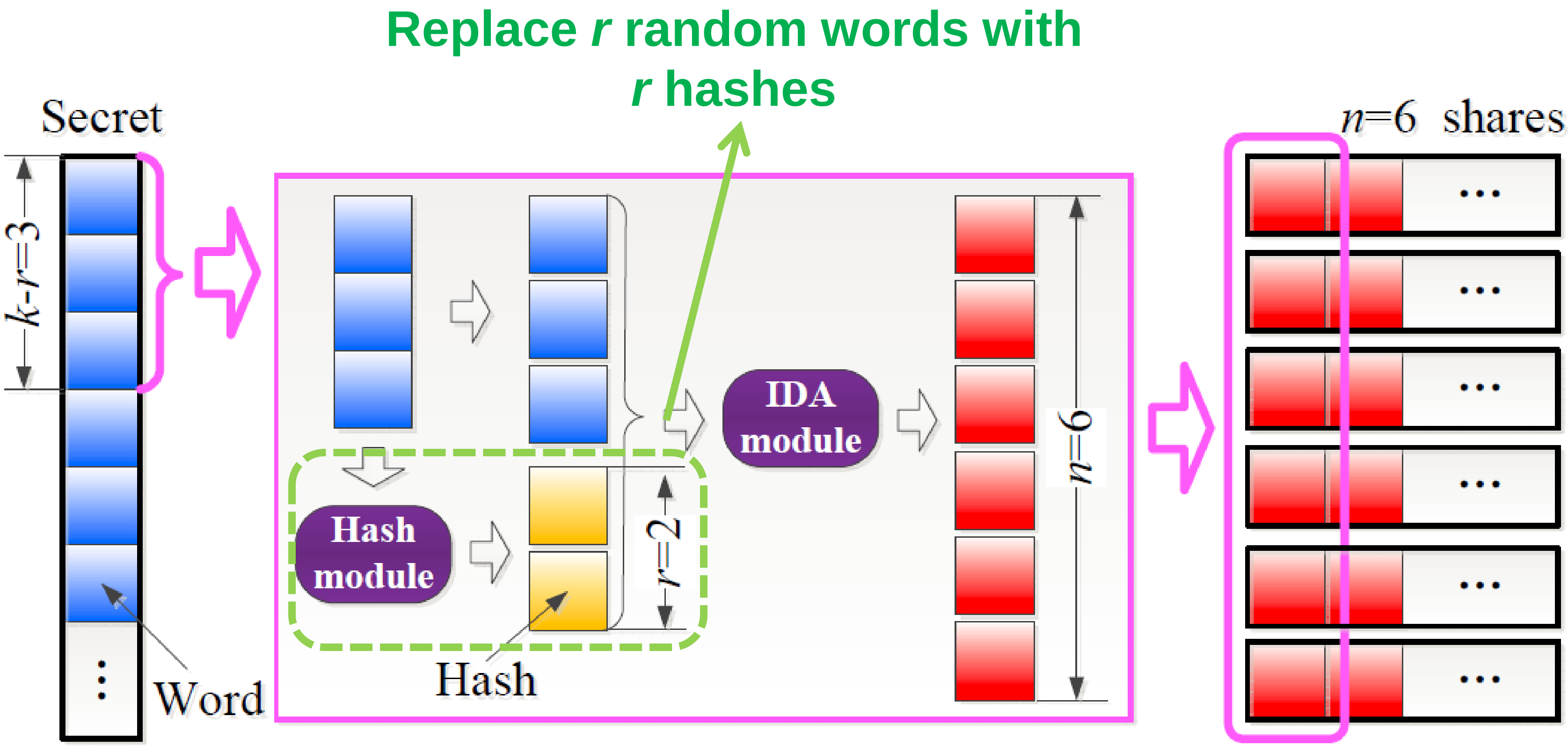
Single-user dedup before uploads

Cross-user dedup by VMs

Avoids cross-user dedup due to side-channel attacks
[Harnik et al., IEEE S&P'10]

CRSSS

- Example: $n = 6, k = 5, r = 2$



CRSSS

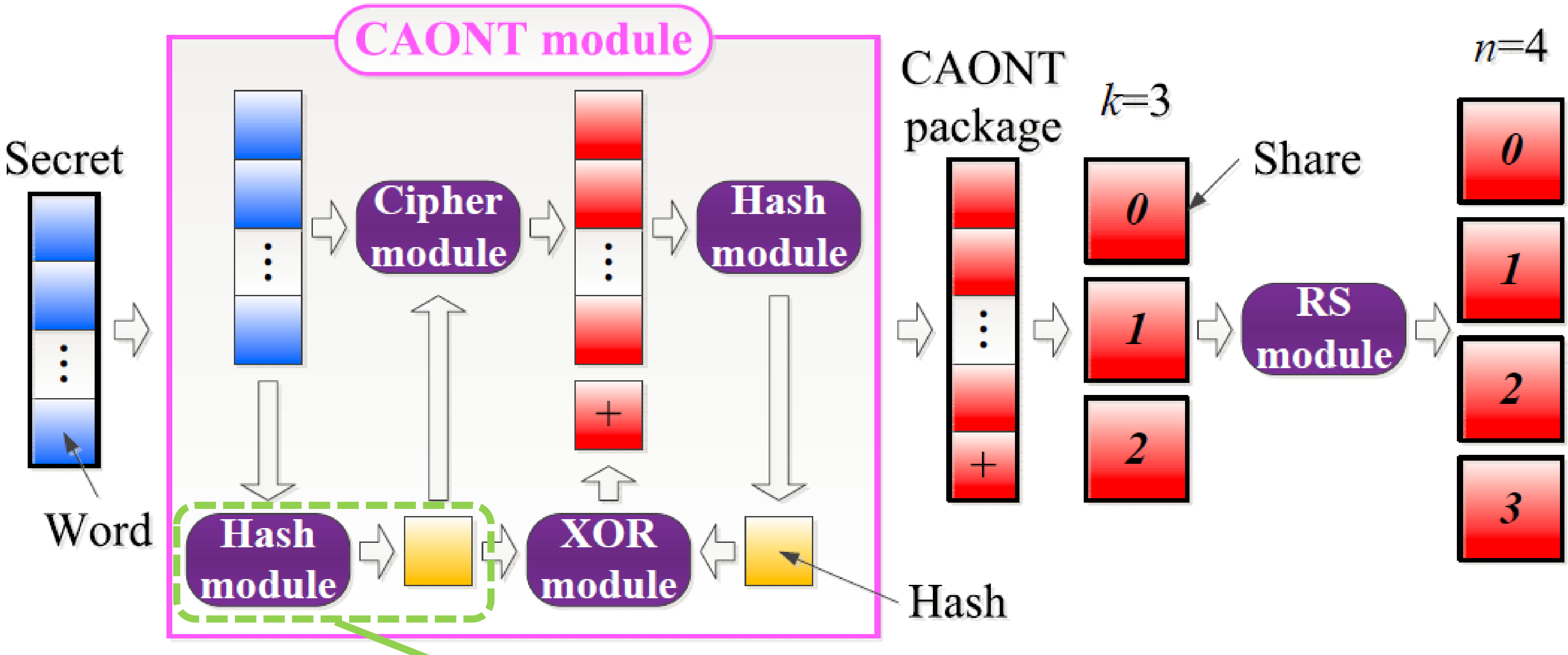
- Generate r hashes from k-r secret words:

$$h_i = \mathbf{H}(D, i), \text{for } i = 0, 1, \cdots, r-1,$$

- D = data block of the k-r secret words
- i = index
- $\mathbf{H}$ = cryptographic hash function (e.g., SHA-256)

CAONT-RS

- Example: n =4, k=3, r = k -1 = 2:



Replace the random key with a hash

CAONT-RS

- Transform s secret words d_0, d_1, …, d_{s-1} into s+1 CAONT words c_0, c_1, …, c_s:

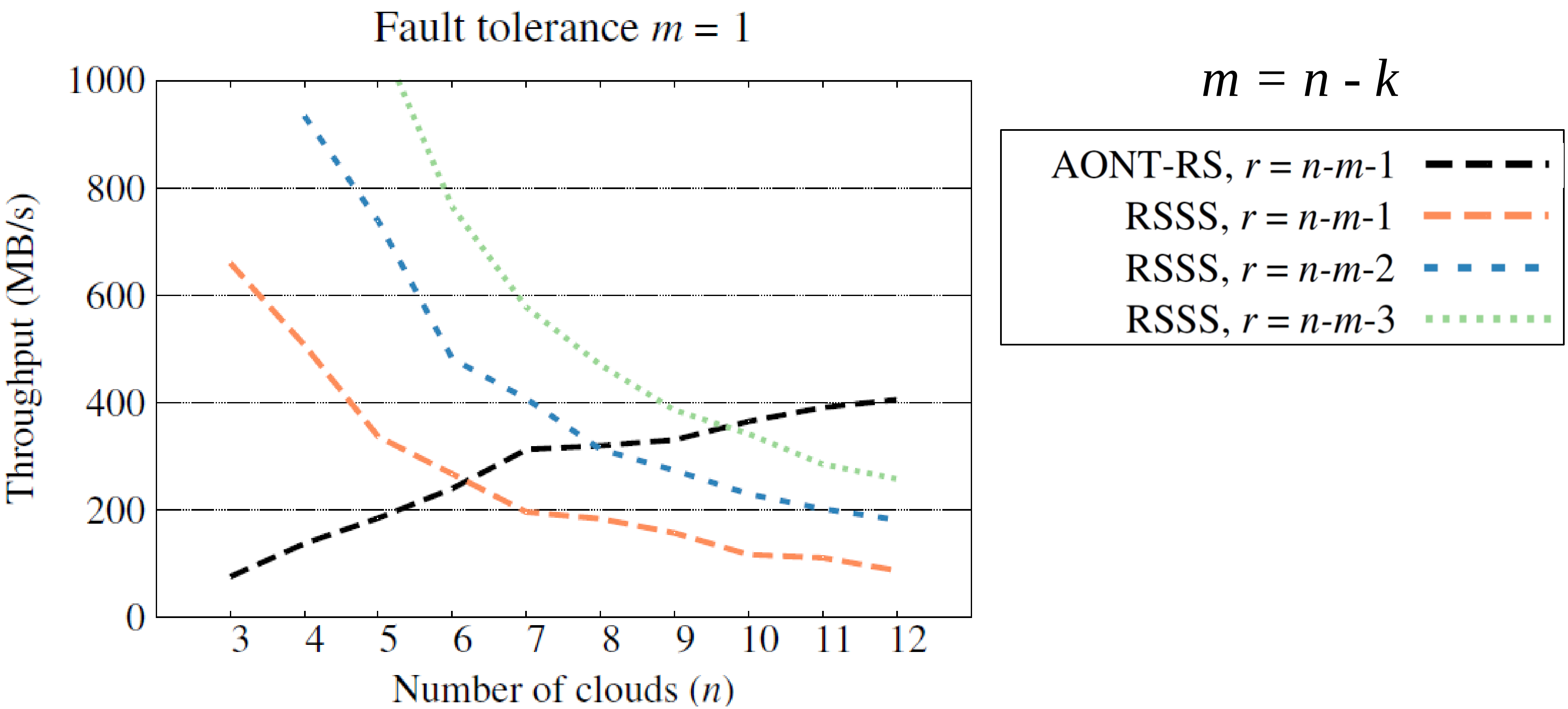
$$c_i = d_i \oplus \mathbf{E}(h_{key}, i), \text{for } i = 0, 1, \cdots, s-1,$$
$$c_s = h_{key} \oplus \mathbf{H}(c_0, c_1, \cdots, c_{s-1})$$

  - $\oplus$ = XOR operator
  - h_{key} = hash key generated from the secret via a cryptographic hash function (e.g., SHA-256)
  - i = index
  - $\mathbf{E}$ = encryption function (e.g., AES-256)

Evaluation Setup

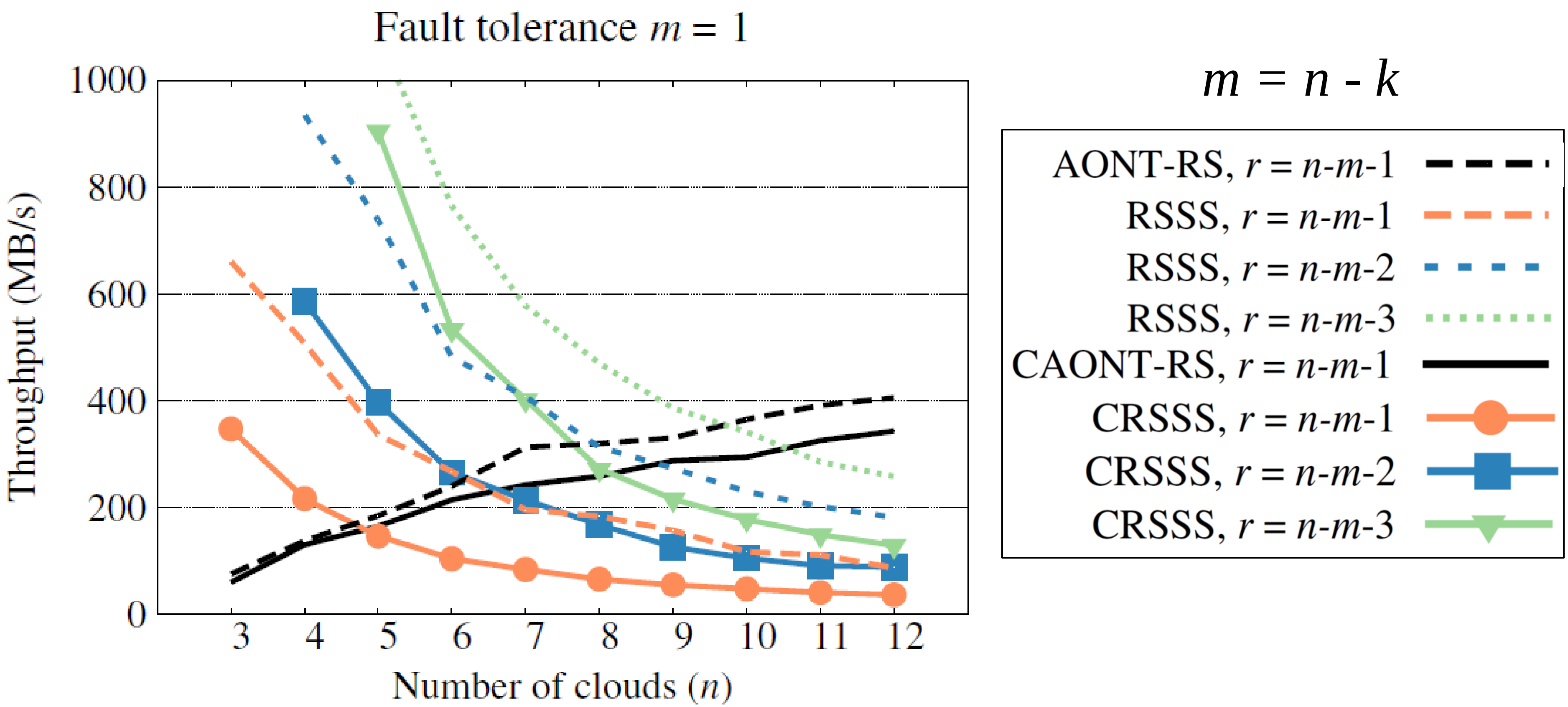
- Evaluate the computational throughput of CRSSS and CAONT-RS
- Setup:
  - OpenSSL for encryption (AES-256) and hash (SHA-256)
  - Jerasure [Plank, 2014] & GF-Complete [Plank, 2013] for encoding
  - Implementation in C
- Compare:
  - RSSS vs. CRSSS
  - AONT-RS vs. CAONT-RS
  - CRSSS vs. CAONT-RS

Evaluation



$m = n - k$

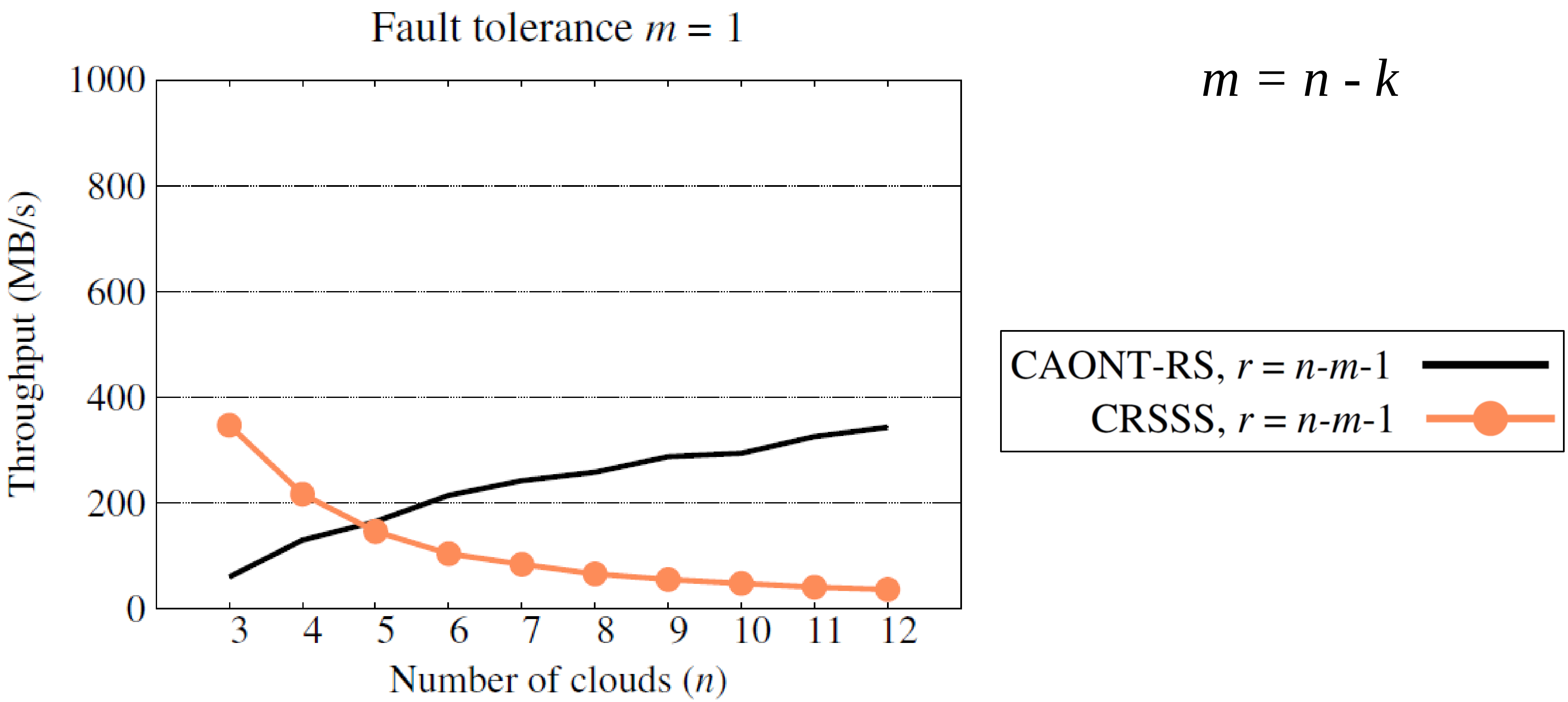
Evaluation



- CRSSS has much higher overhead (~30%) than RSSS due to more hash computations; yet, CAONT-RS has limited overhead (~8%) over AONT-RS

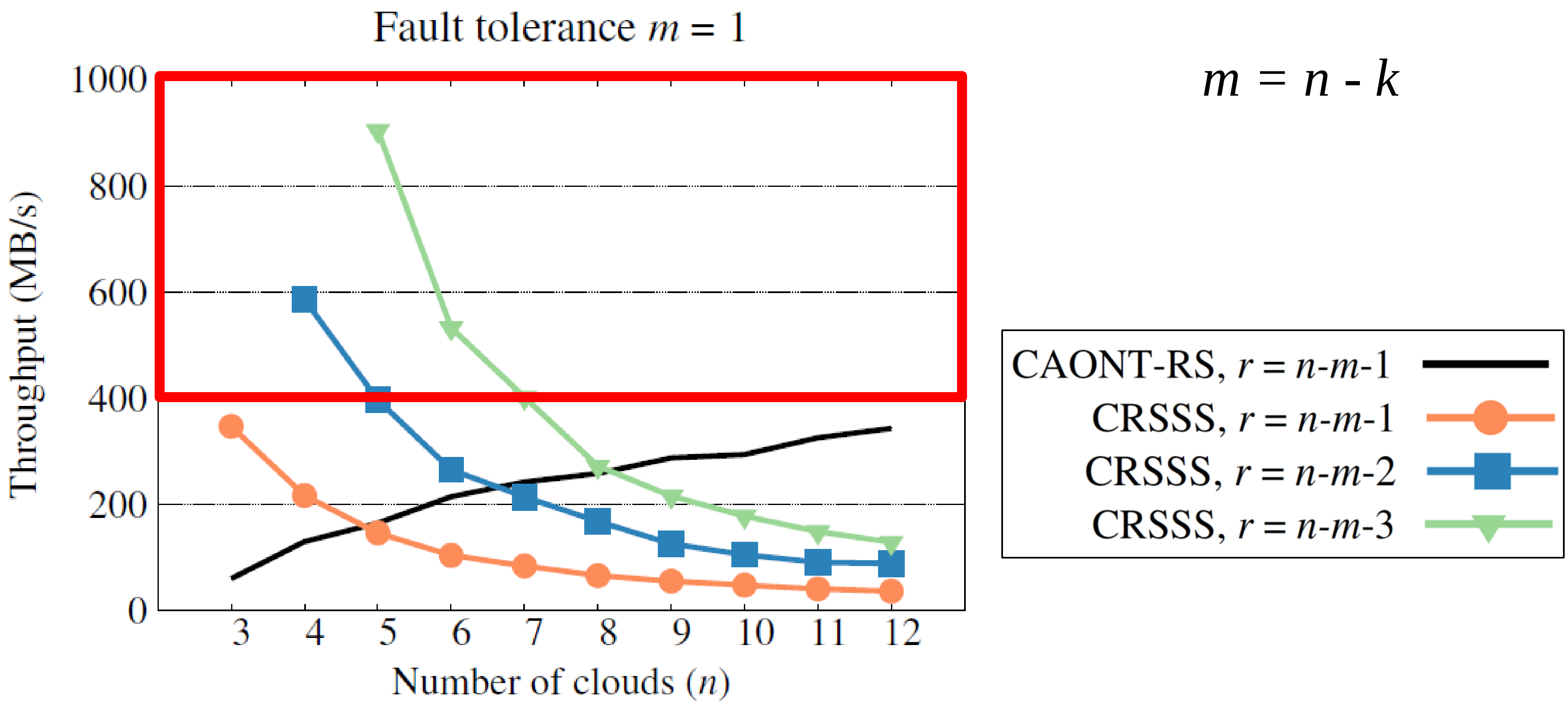
Evaluation

$m = n - k$



- CRSSS and CAONT-RS are complementary in performance: CRSSS decreases in throughput due to more hashes, while CAONT-RS increases in throughput due to RS encoding

Evaluation



$m = n - k$

- For smaller r, CRSSS achieves much higher throughput (>400MB/s), but with higher storage overhead
  → tradeoff between throughput and storage

Conclusions

- Defines a framework of **convergent dispersal** that enables keyless security and deduplication
- Two implementations based on state-of-the-art: **CRSSS** and **CAONT-RS**
  - Both are complementary in performance
- Future work:
  - Complete cloud storage prototype
  - Cost-performance analysis
  - Security analysis
  - Evaluation in real-world deployment