POPLAR/STAR MEASUREMENTS

https://sofiaceli.com/thoughts/star_verification.pdf

Sofía Celi
Brave Research

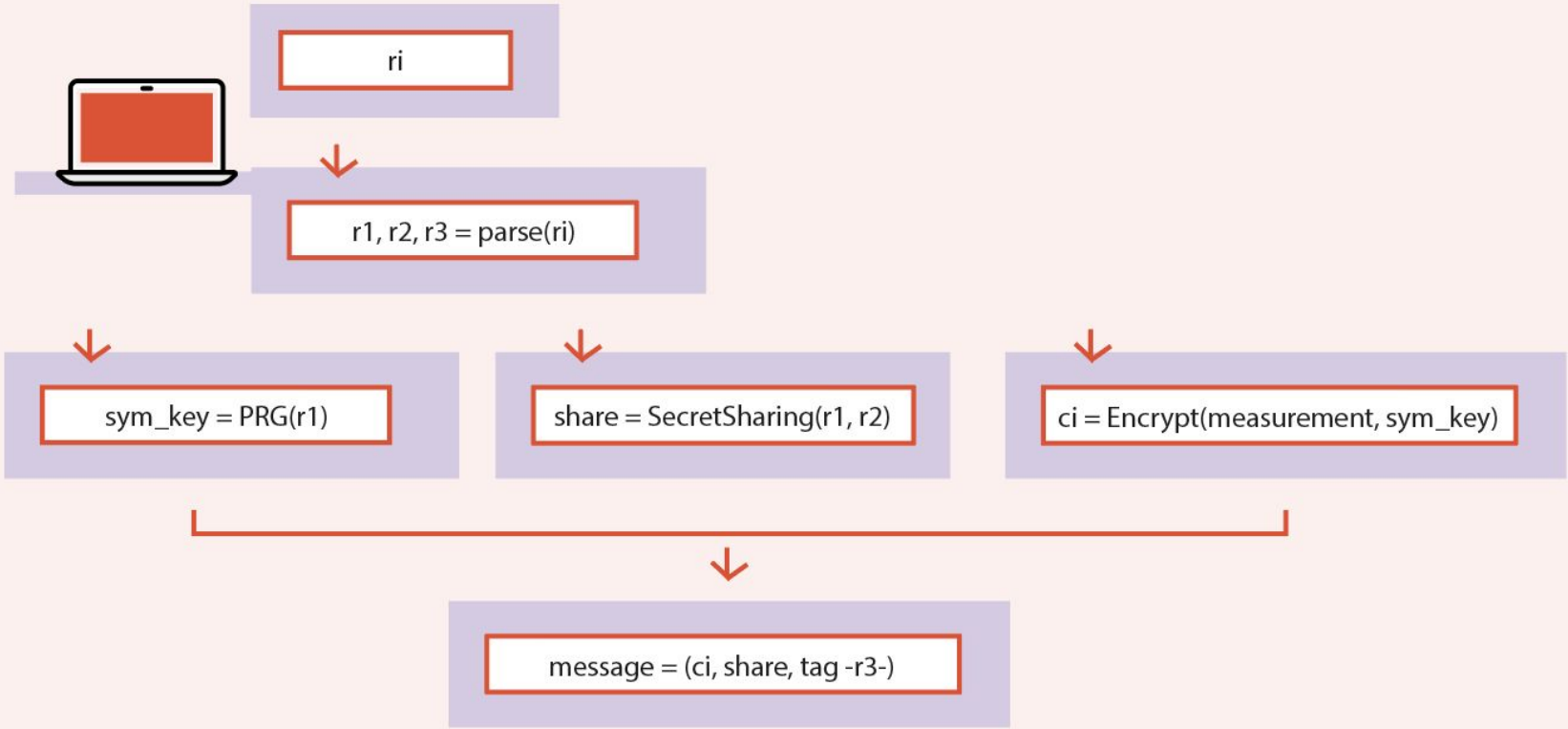
Notation

- k is the threshold used for performing server-side aggregation.
- n is the total report size submitted by C clients.
- C is the set of all clients.
- S is the aggregation server.
- O is the randomness server used in STAR.
- m is a message to secret-share.
- t is an integer $\in$ N that states in POPLAR that a string σ appears in a list $(a_1, \ldots a_c)$ more than t times.
- σ is a string to search for in a list.
- l is the length of σ.

STAR

- Each client constructs a ciphertext by encrypting their measurement (and any auxiliary data) using an encryption key derived deterministically from randomness (derived, in turn, from the measurement)
- The client then sends:
  a. the ciphertext
  b. a *k-out-of-n* secret share of the randomness used to derive the encryption key
  c. a deterministic tag informing the server which shares to combine
- The aggregation server groups reports with the same tag, and recovers the encryption keys from those subsets of size ≥ k

Share creation



Aggregation and reveal phase

Groups together messages with the same tag into a set

Divides the set into subsets of K size

r1 = recover(share)

sym_key = PRG(r1)

measurement = decrypt(ci, sym_key)

STAR

STAR is a scheme that uses:

- An algorithm that generates deterministic randomness: OPRFs, Hashes, AES-based...
- Secret-sharing scheme
- Sorting algorithm

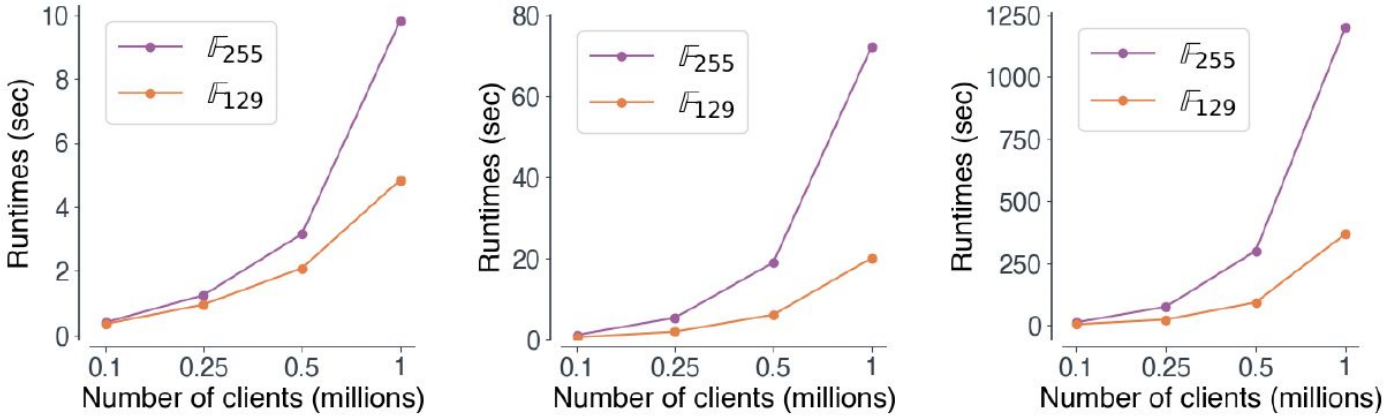


Figure 9: Aggregation server computation runtimes (seconds) based on number of clients. Graphs from left-to-right corresponding to a threshold $\kappa \in \{0.01\%, 0.1\%, 1\%\}$ of total number of client inputs. Performance is compared for both fields $\{\mathbb{F}_{129}, \mathbb{F}_{255}\}$.

Taken from the ACM-CSS, 2022 publication: "STAR: Secret Sharing for Private Threshold Aggregation Reporting" by Alex Davidson, Pete Snyder, E.B. Quirk, Joseph Genereux, Bejamin Livshits, Hamed Haddadi

STAR

Aggregation computational times is dependent on:

- Secret-sharing algorithm:
  - STAR uses a “adept secret-sharing (ADSS)” [BDR20] scheme
  - **Share generation:** sharing an *m*-byte message *M* takes *O(m)* time and, more concretely, about the amount of time to symmetrically encrypt and hash *M*
  - **Secret recovery:** Message recovery takes the same time as sharing
  - The scheme can be adapted to perform error-correction but it becomes exponential on the amount of shares passed to the recovery procedure (worst-case of 2^n)

[BDR20] Mihir Bellare, Wei Dai, and Phillip Rogaway. Reimagining secret sharing: Creating a safer and more versatile primitive by adding authenticity, correcting errors, and reducing randomness requirements. 2020(4):461–490, October 2020.

Malicious shares and malicious clients

Not having error-correction (or verifiability) opens an attack against the STAR protocol.

- There exists a set *x* of malicious clients that corrupt *j* amount of shares.
- The recovery procedure will be unable to pinpoint which share is invalid (corrupted): the recovery procedure will halt and the whole batch of *k* size will be discarded.
- This gives the possibility for malicious clients to perform DoS attacks with the goal of discarding sets of honest measurements.

Solutions

- Use ADSS with error-correction → can be expensive
- Use a secret sharing scheme with verifiability (Feldman's scheme [Fel87] or Pedersen scheme [Ped92])
- Perform error-correction with a different construction which is the subject of a publication under review. In the work we arrive to a construction that achieves O(log n)

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. pages 427–437, 1987.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. pages 129–140, 1992.

Feldman’s scheme

- Create *k* amount of **commitments** (proof that a share is valid):
  - It runs once for a set of *n* shares. It is linear on the size of *k* (it generates *k* commitments for a set of n shares).
- Verify on **each share** that the set of *k* commitments is valid:
  - It is linear on the size of k for a single share.
  - This phase can be expanded to verify a whole subset/set, in which case:
    - Iit is $O(n * k)$ for verifying the set of n shares.
    - It is $O(k^2)$ for verifying a subset of k size.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. pages 427–437, 1987.

Feldman’s scheme

- **Worst-case complexity:** This case occurs when a corrupted share is placed at the end of the set/subset. The cost is: cost of verifying a single-share *(O(k)) * (size of set ∨ size of subset): O(k ∗ (n ∨ k))*
- **Average-case complexity**: Average case can be affected if the corruption probabilities for each share vary (which is the case here as only a subset of clients can be considered malicious): in this case, the average case depends on the probability of the attacker of corrupting a set of shares and of the network on delivering them in a specific order.
- **Best-case complexity**: This case occurs when there is only one corrupted share per set/subset and it is placed at the start of the set/subset. The cost is: cost of verifying a single-share *(O(k)) * 1 * number of sets/subsets: O(1 ∗ (1 ∨ |{x ⊂ n : |x| = k}|))*

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. pages 427–437, 1987.

Measurements of Feldman’s scheme

- Benchmarks in Rust of the secret sharing scheme (Shamir Secret Sharing) with verifiability (the Feldman’s scheme): the code is not optimized.
- The Rust implementation can be found here: https://github.com/claucece/secret-sharing-extra
- We defined the following parameters:
  - threshold (which is k, the subset size)
  - report size (which is n, the total size of the measurements reported)
  - In all cases by secret we used a string of 32 bytes in size.
- We report numbers when using curve25519/Ristretto and Sec256k1 for the field and elliptic curve operations.
- We ran our benchmarking on a MacBook Pro with arm64, Darwin Kernel Version 22.3.0, Apple M1 Max chip.
- We are using Rust with version 1.68.0.

Measurements of Feldman's scheme

Report size (n)	Threshold (k)	Verification Time
256	10	0.08
	25	0.20
	50	0.40
	100	0.80
	128	1.01
1024	10	0.32
	25	0.79
	50	1.58
	100	3.17
	128	4.04
1280	10	0.40
	25	0.99
	50	1.97
	100	3.94
	128	5.04

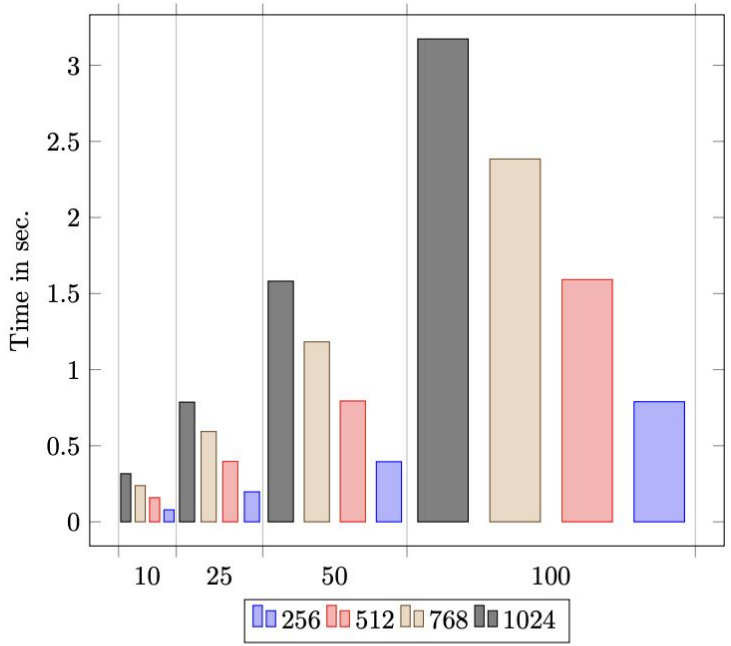
Using Curve25519/Ristretto. Numbers are reported in seconds

Measurements of Feldman's scheme

Report size (n)	Threshold (k)	Verification Time
256	10	0.04
	25	0.08
	50	0.16
	100	0.31
	128	0.40
1024	10	0.16
	25	0.34
	50	0.66
	100	1.28
	128	1.62
1280	10	0.20
	25	0.43
	50	0.84
	100	1.61
	128	2.05

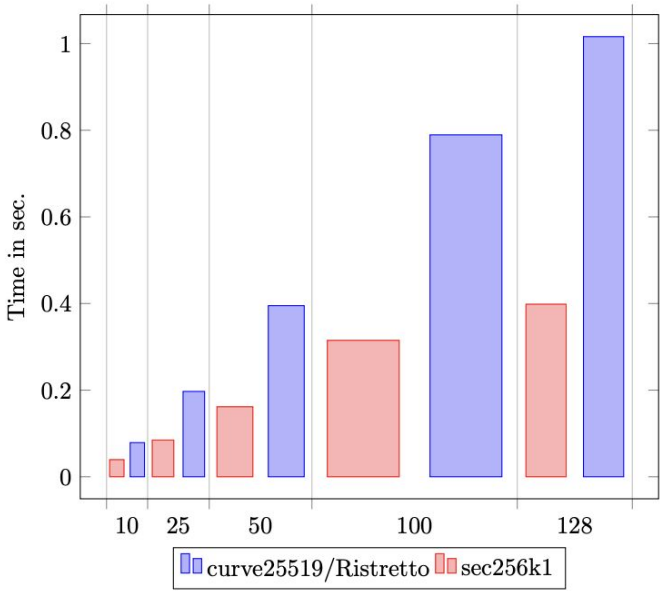
Using sec256k1. Numbers are reported in seconds

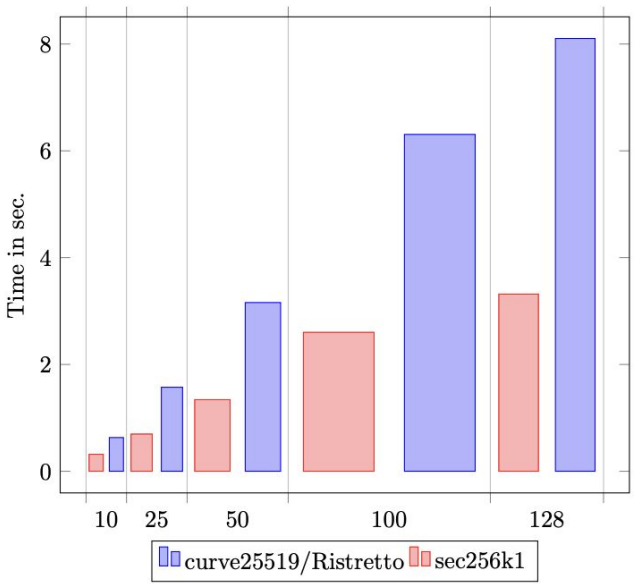
Measurements of Feldman’s scheme



Comparison of benchmarks: x-axis states the threshold sizes, y-axis states the times. The numbers that relate to the colours represent the set of size n. This is using the times reported when using Curve25519/Ristretto.

Measurements of Feldman's scheme





Comparison of benchmarks: x-axis states the threshold sizes, y-axis states the times. The numbers that relate to the colours represent the curve choice: Curve25519/Ristretto or sec256k1. The numbers are for n = 256 (left) and n = 2048 (right).

Smart STAR-VSS algorithm

1. Perform the recovery functionality first on a subset x_i of k size. If it fails on the subset, it runs the verification of that single subset x_i.
2. The verification algorithm will remove the invalid shares (the subset r of x_i), and return a subset of size $k - |r|$.
3. The returned subset of size $k - |r|$ can be used to construct a k size subset (by fetching w shares from the set y of n size so that $k - |r| + |w| = k$), and perform recovery again.

- Arrive to a better time complexity of the overall scheme.

Lightweight Techniques for Private Heavy Hitters: POPLAR

- POPLAR [BBC+21] uses incremental distributed point functions, a cryptographic primitive that builds on standard distributed point functions (DPFs).
  - Each client holds a *l*-bit string and a set of servers aggregates them.
- Cost is linear in *l*: length (in bits) of the string to search for.

[BBC+21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. pages 762–776, 2021.

Measurements of POPLAR

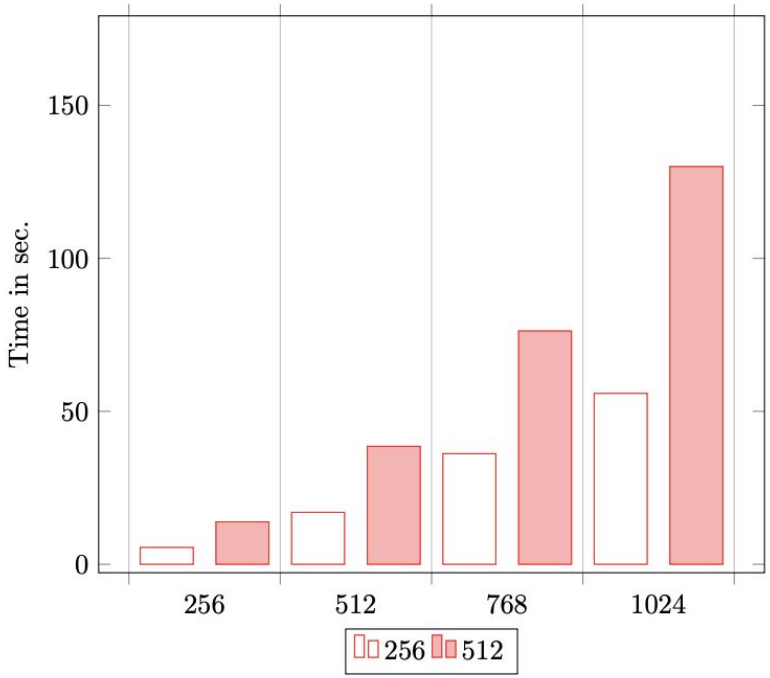
- Benchmarks in Rust, using the measurements framework as defined in https://github.com/henrycg/heavyhitters
- The code compiles only for a older Rust version (we are using, hence, 1.47.0) on a older MacBook Pro 12.3.1 with 2.3 GHz Dual-Core Intel Core i5 (model I5-7360U) with x86 64.
- This makes the POPLAR measurements perhaps not as accurate as the ones taken for the VSS scheme.
- Ongoing work to “update” the POPLAR rust code.
- Measures the running time from the moment after the servers collect the last incremental DPF keys from the clients until the servers produce their output.
- It tests both over 256-bit length strings and 512-bit length strings (we couldn’t compile the code with longer strings).

Measurements of POPLAR

Input size	Client requests	Threshold	Total Time
256	**256**	0.1%	5.51
	512	0.1%	16.96
	768	0.1%	36.15
	1024	0.1%	55.86
	1280	0.1%	86.42
	1536	0.1%	126.58
	1792	0.1%	155.29
512	**256**	0.1%	13.88
	512	0.1%	38.58
	768	0.1%	76.27
	1024	0.1%	130.00
	1280	0.1%	171.00
	1536	0.1%	294.67
	1792	0.1%	332.42

Numbers are reported in seconds. The threshold here represents that more than 0.01% clients hold a specific string.

Measurements of POPLAR



Comparison of benchmarks: x-axis states the number of client request, y-axis states the times (in seconds). The numbers that relate to the colours represent the input size l.

Conclusions

- STAR with verifiability (STAR-VSS) is efficient for $k > 10$ and $k < 128$~ (useful values in practice).
- STAR with verifiability (STAR-VSS) is efficient depending on the curve/field chosen: sec256k1 is faster than curve25519/ristretto.
- The performance of POPLAR is sensitive to the size of the message, while STAR is not
- POPLAR performs better than STAR-VSS given
  - a large % of malicious inputs
  - large values of k
  - small messages

- It is difficult to properly compare the schemes as they grow depending on different parameters:
  - Both, however, are useful in practice.

Future work

- Measure the whole STAR functionality with VSS → probably not much difference
- Update the POPLAR codebase → perform benchmarks in the *libprio* rust code
- Formalise the verification scheme in the same formal framework as the ADSS one
- Formalise and present the results of scheme with error-correction that arrives to *O(log n)*

- *Ongoing research and engineering work!*

THANK YOU!

@claucece