Splintering: A resource-efficient and private scheme for distributed matrix inverse

Praneeth Vepakomma, Susovan Pal, Ramesh Raskar

Abstract
Performing computations while maintaining privacy is an important problem in today’s distributed device ecosystem. Consider the following set up where the client would like to perform an operation of computing the inverse of a large matrix that it hosts, but would like to use the superior computing ability of the server to do so while ensuring differential privacy of any communications from the client to the server. We present a scheme for splitting the client data into privatized shares called splinters that are transmitted to the server in such a setting. Splinters are noisy linear combinations of the original sensitive matrix with several random matrices. The server performs the requested operations on these shares instead of on the raw client data at the server. The obtained intermediate results are sent back to the client where they are assembled by the client using private coefficients that it holds in order to obtain the final result. Our mechanism ensures that any communication made from the client in terms of the shares is private with respect to it’s matrix that needs to be inverted and the client’s sensitive coefficients that it holds.

2012 ACM Subject Classification Design and analysis of algorithms → Distributed algorithms, Security and privacy → Security services → Privacy-preserving protocols

Keywords and phrases Dummy keyword

Digital Object Identifier 10.4230/LIPIcs...2023.23

Funding Praneeth Vepakomma, Susovan Pal, Ramesh Raskar: (Optional) author-specific funding

Acknowledgements I want to thank . . .

1 Introduction

We describe a method for resource efficient computation in distributed linear algebra via differentially private shares of data called splinters, that are noisy linear combinations of the sensitive input matrix with several random matrices. This enables resource constrained client devices to receive the inverse of the matrix as a service from server without having to completely perform the needed inversion on-device. The server performs computations over these splinters and the corresponding results are sent back to the client. The client has required coefficients and a recipe of specific operations to perform over these intermediate results, referred to as unsplintering in order to obtain the required final result. The scheme integrates well with any state of the art non-distributed matrix inversion scheme of choice such as (1), that the server could use in the context of splintering.

2 Related work

1. Coded Computing was introduced based on coding theory to perform distributed computations with benefits such as robustness to straggler clients, byzantine robustness to malicious clients, information theoretic privacy. Coded computing scheme for distributed matrix multiplication in particular include (16; 8; 15) for straggler mitigation,(7) for robustness from malicious actors that return corrupted results, (14) for resiliency from stragglers, byzantine robustness from malicious actors and information theoretic privacy at the same time. A survey of some of the coded computing methods is provided in
(11). Coded computing based on MDS codes for matrix inverse is provided in (4) and a scheme based on GASP codes is provided in (6). While some of the above works provide privacy guarantees from an information theoretic notion, our work focuses on differential privacy guarantees for resource-efficient distributed matrix inverse. One other significant difference is that while the above methods used encodings based on coding theory, our shares are just noisy (and private) linear combinations, even though the final operation that we perform such as the matrix inverse is a non-linear operation.

2. Cryptographically secure and differentially private methods

Homomorphic encryption based protocols for delegating several linear algebra computations were presented in (12) along with secure verification guarantees on the obtained solution in \(O(n^2 \log n)\) time. Differentially private mechanisms for some linear algebra computations in the streaming setting with space complexity guarantees were given in (13) while mechanisms for differentially private matrix and tensor factorizations were given in (9).

Our work instead focuses on the standard (non-streaming) setting for differentially private matrix inverse in the distributed setting.

3 System interactions

We now detail the first-order idea of splintering in the non-private setting. For a \(d\) dimensional input query matrix \(X\), the client device creates \(d\) shares corresponding to \(X\) as \(\{Z_1, Z_2 \ldots Z_d\}\) so that

\[ X = \text{Splint}(Z_1, Z_2 \ldots Z_d) \forall i \in 1..d \]

The most basic splint function that allows for such a representation is a linear combination using coefficients \(\alpha_i\) as

\[ X = \sum_{i}^{d} \alpha_i Z_i \forall i \in 1..d \]

The \(\alpha_i's\) are not shared with any other entity, be it another client or a server. The splinters \(Z_i\) are shared with the server. The server performs a set of application dependent operations on the splinters \(Z_i, \forall i \in 1 \ldots d\) and sends results \(\{\beta_i\}\) back to client on either all or a subset of the \(d\) shares. The client performs a local computation called \(\text{UnSplint}\) using original shares \(Z_i\), its corresponding \(\alpha_i's\) that are known only to the client and received \(\beta_i's\) obtained from the server. This unsplintering operation reveals the true result \(l\) of the intended application to the client.

\[ l = \text{UnSplint}(\alpha_i, Z_i, \beta_i), \forall i \in 1..d \]

Note that although \(x\) is represented via a linear combination, the computation of \(\{\beta_i\}\) and \(\text{UnSplint}\) is not necessarily linear.

**Efficient setting** Note that there is a computational benefit for the client only if the \(\text{UnSplint}\) operation and the generation of splinters can be performed more efficiently by the client in comparison to entirely performing required service (say, matrix inverse) on its own premise.

**Private setting** Differentially privacy is quite a popular mathematical notion of privacy (5) for releasing outputs of queries. The splintering mechanism in it’s entirety is differentially private if it is ensured that any communication made from the client in terms of the shares is differentially private with respect to the matrix that needs to be inverted as well as with respect to the client’s sensitive coefficients that it holds. These coefficients are useful for
the client to perform the unSplint operation. It is worth noting that without the privacy constraint, the client can as well trivially send over all of its matrix to the server in order to completely save itself from performing any computations. Therefore, it is worth noting that splintering is useful when operated under a setting that is both efficient and private. That said, a privacy-compute-communication trade-off comes into play in practice in order to efficiently perform splintering with differentially private guarantees.

4 Examples under various settings

We now provide some examples of splintering and unsplintering operations under various settings of a.) non-private and non-efficient, b.) non-private and efficient, c.) private and efficient.

4.1 Splintering for sigmoid (non-private and non-efficient setting)

Theorem 1. For a scalar input \( x \in \mathbb{R} \) expressed using scalar real-valued splinters \( \{z_1, z_2, \ldots, z_k\} \) as \( x = \sum_{i=1}^{k} \alpha_i z_i \), the unsplintering operation to compute the sigmoid function \( s(x) \) using \( s(z_1), s(z_2), \ldots, s(z_k) \) is given by

\[
s(x) = \frac{1}{1 + \prod_{i=1}^{k} \left( \frac{1-s(z_i)}{s(z_i)} \right)^{-\alpha_i}}
\]

Proof. The sigmoid function is given by

\[
s(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}
\]

This can be rearranged as

\[
\frac{1 - s(x)}{s(x)} = e^{-x}
\]

By following our proposed approach of using splinters we substitute \( x = \sum_{i=1}^{k} \alpha_i z_i \) in the above form of sigmoid to get

\[
s(x) = \frac{1}{1 + e^{-\sum_{i=1}^{k} \alpha_i z_i}} = \frac{1}{1 + \prod_{i=1}^{k} e^{-\alpha_i z_i}}
\]

Now substituting this into the rearranged form of the sigmoid above we get

\[
s(x) = \frac{1}{1 + \prod_{i=1}^{k} \left( \frac{1-s(z_i)}{s(z_i)} \right)^{-\alpha_i}}
\]

Therefore the final scheme only requires computing \( s(z_i)'s \) at the server while the client can figure out \( s(x) \) by using \( \alpha_i's \) that are known only to the client and not shared with server.

As the setting is non-private, the success of several attacks cannot be ruled out in this setting. We now provide one other example, in this setting with regards to computing the softmax function.
4.2 Splintering for softmax (non-private and non-efficient setting)

**Theorem 2.** For a real-valued input vector $\mathbf{x}$ expressed using splinters $\{Z_1, Z_2, \ldots, Z_k\}$ as $\mathbf{x} = \sum_{i=1}^k \alpha_i Z_i$, the unsplintering operation to compute $s(\mathbf{x})$ where $e^{z_1} = a_1, e^{z_2} = a_2$ and $e^{z_3} = a_3$ and $w_1 = \frac{a_1}{a_1 + a_2 + a_3}, w_2 = \frac{a_2}{a_1 + a_2 + a_3}$ and $w_3 = \frac{a_3}{a_1 + a_2 + a_3}$ as follows

$$\text{softmax}(\alpha_1 Z_1 + \alpha_2 Z_2) = \frac{e^{\alpha_1 Z_1 + \alpha_2 Z_2}}{\sum_{j=1}^d (e^{\alpha_1 Z_{1,j}})(e^{\alpha_2 Z_{2,j}})}$$

(1)

Proof.

$$\text{softmax}(\alpha_1 Z_1 + \alpha_2 Z_2) = \frac{e^{\alpha_1 Z_1 + \alpha_2 Z_2}}{\sum_{j=1}^d e^{\alpha_1 Z_{1,j}} + e^{\alpha_2 Z_{2,j}}}$$

(2)

$$= \frac{1}{\sum_{j=1}^d (e^{\alpha_1 Z_{1,j}})(e^{\alpha_2 Z_{2,j}})}$$

(3)

Assume one component each of $Z_1, Z_2$ are known and let $\vec{w}_i = \text{softmax}(Z_i)$. Then $Z_i = \text{softmax}^{-1}(w_i; a_{1i})$. Therefore we plugin this inverse in 2 to rewrite softmax as

$$\text{softmax}(\alpha_1 Z_1 + \alpha_2 Z_2) = \frac{e^{\alpha_1 \vec{w}_{1i} + \alpha_2 \vec{w}_{2i}}}{\sum_{j=1}^d e^{\alpha_1 \vec{w}_{1,j}} + e^{\alpha_2 \vec{w}_{2,j}}}$$

(4)

**Analytical inverse of softmax when one component is known:** Let $e^{z_1} = a_1, e^{z_2} = a_2$ and $e^{z_3} = a_3$. Then $w_1 = \frac{a_1}{a_1 + a_2 + a_3}, w_2 = \frac{a_2}{a_1 + a_2 + a_3}$ and $w_3 = \frac{a_3}{a_1 + a_2 + a_3}$ and therefore

$$\frac{w_1}{e^{z_1}} = \frac{w_2}{e^{z_2}} = \frac{w_3}{e^{z_3}} = \lambda$$

This implies that

$$z_2 = \log \left( \frac{w_2}{w_1} \right) e^{z_1}$$

5 Splintering for matrix inverse (non-private and efficient setting)

We now propose a splintering scheme for the important operation of matrix inversion in this section, but in the non-private, yet efficient setting. We consider a setting where the client has a large sensitive matrix $M_{n \times n}$ and would like to use the service of a computationally powerful server in order to privately obtain the inverse $M^{-1}$.

**Theorem 3.** The unsplintering operation to compute the matrix inverse $M$ using two splinters $Z_1, Z_2$ with secret coefficients $\alpha_1, \alpha_2$ while only having to compute the inverse of the splinters is given by

$$(\alpha_1 Z_1 + U Z_2 V)^{-1} = 1/\alpha_1 Z_1^{-1} - 1/\alpha_1 Z_1^{-1} U (Z_2^{-1} + V 1/\alpha_1 Z_1^{-1} U)^{-1} 1/\alpha_1 V Z_1^{-1}$$

where $U Z_2 V$ is of the form

$$\begin{bmatrix} \alpha_3 & \alpha_4 \\ \alpha_3 & \alpha_4 \end{bmatrix}$$
Proof. In order to obtain the inverse of a private matrix $M_{n \times n}$, we split it into the form using $A_{n \times n}, U_{n \times p}, V_{p \times n}$ and $Z_2$ of dimension $p \times p$ as

$$M^{-1} = (A + UZ_2V)^{-1}$$

where $A$ is written in terms of a splinter matrix $Z_1$ of dimension $n \times n$ as $A = \alpha_1Z_1$ and $UZ_2V$ is of the form

$$\begin{bmatrix} \alpha_3 & \alpha_3 \\ \alpha_3 & \alpha_3 \end{bmatrix} \begin{bmatrix} \alpha_4 & \alpha_4 \\ \alpha_4 & \alpha_4 \end{bmatrix}$$

by the popular matrix inversion lemma (Sherman–Morrison–Woodbury formula)

$$(A + UZ_2V)^{-1} = A^{-1} - A^{-1}U(Z_2^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

Therefore the proposed scheme now is to send $Z_1, Z_2$ to the server which sends back $Z_1^{-1}, Z_2^{-1}$ to the client that holds $M$ along with $\alpha_1, \alpha_2, \alpha_3$. The client then obtains the final solution $M^{-1}$ by computing

$$(\alpha_1Z_1 + UZ_2V)^{-1} = 1/\alpha_1Z_1^{-1} - 1/\alpha_1Z_1^{-1}U(Z_2^{-1} + V1/\alpha_1Z_1^{-1}U)^{-1}1/\alpha_1VZ_1^{-1}$$

Computational savings: In cases where $n >> p$, the matrix $(Z_2^{-1} + VA^{-1}U)^{-1}$ which is of dimension $k \times k$ is much easier to invert than the original private data matrix $M_{n \times n}$ thereby offloading the heavier computation onto the server while preserving privacy and requiring a much smaller computation on the client.

This can further be generalized to 3 splinters as follows where

$$U = [U_1 \ U_2], V = [V_1 \ V_2], Z = [Z_1 \ 0 \ Z_2]$$

so that

$$A + U_1Z_1VT + U_2Z_2VT = A + UZV^T$$

Then, just apply the Woodbury matrix identity as above to complete the proof. It is well-known in linear algebra that if one can invert a nonsingular $n \times n$ matrix in $O(T(n))$ time, then one can multiply $n \times n$ matrices in $O(T(3n))$ time. To see this, let $A$ and $B$ be matrices and consider the following $3n \times 3n$ matrix:

$$D = \begin{bmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{bmatrix}$$

1 where $I$ is the $n$-by- $n$ identity matrix. One can verify by direct calculation that

$$D^{-1} = \begin{bmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{bmatrix}$$

Inverting $D$ takes $O(T(3n))$ time and we can find $AB$ by inverting $D$. Note that $D$ is always invertible since its determinant is 1. Therefore, a splintering method for one operation benefits the other.
In this section, we share a mechanism for matrix inversion in a private and efficient setting. Prior to that, we introduce some relevant preliminaries and terminology required in our use-case.

### 6.1 Preliminaries for privacy

A mechanism \( A : X^n \to M \) is a randomized algorithm that takes a dataset \( D \) as input, and outputs a privatized version of the summary \( f \) on \( D \). The mechanism \( A \) satisfies \((\epsilon,0)\) differential privacy (also pure differential privacy) if, for all adjacent datasets \( D \sim D' \) and for all measurable sets \( S \) of \( M \) the following holds:

\[
P[A(D) \in S] \leq \exp(\epsilon) P[A(D') \in S]
\]

The intuition is that the change of a single element of the data space \( X \) does not significantly alter the output distribution of the mechanism. As a relaxation, the mechanism \( A \) satisfies \((\epsilon,\delta)\)-differential privacy (also approximate differential privacy) if, for all adjacent datasets \( D \sim D' \) and for all measurable sets \( S \) of \( M \):

\[
P[A(D) \in S] \leq \exp(\epsilon) P[A(D') \in S] + \delta
\]

Intuitively, \( \delta \) can be thought of as the probability of privacy failure, when pure privacy is not guaranteed.

### 6.2 Terminology for private setting

There are three kinds of splinters that we use in our scheme, the significance of which shows up in the differentially private version of the splintering mechanism that we provide for the operation of a matrix inverse.

1. **Major splinter**: These are splinters that depend on both the input matrix \( M \) and some randomization.
2. **Minor splinter**: These are splinters that do not depend on the input matrix, and only depend on random matrices.
3. **Decoy splinter**: These are a larger set of random matrices, from which the minor splinters can be sampled.

A simple illustration of this terminology is given below.

![Figure 1](image-url)

**Figure 1** We show an example of splintering a face image using decoy splinters, minor splinters and major splinters. On the right side, we show the effect of the reconstruction on the precision of choosing the right unsplintering coefficients denoted by \( \alpha \)’s.
6.3 Main idea for privatization

The main idea for privatization is that mixing up sensitive datasets with a linear combination followed by a sufficient additive noise, leads to a mixed dataset that is differentially private. Moreover, the variance with which $N$ is generated in order to obtain $\epsilon$-differential privacy, happens to reduce with an increase in the number of mixture components (which in our case are splinters). The noise can be calibrated using (3) or (10).

6.4 Level of noise after linearly combining the splinters

The following is the relation between the privacy level $\epsilon$ that is maintained and required noise variance to generate $N$ along with our own annotation made in parentheses to draw an exact analogy for our use-case.

**Theorem 4.** (Privacy guarantee): Fix the mixture degree $\ell$ (which are the number of splinters in our case), the noise level $\sigma_X$ and the number of mixtures $T$ (these are the number of samples, which is 1 in our case). For any $\delta > 0$, $\text{DPMix}(\ell)$ is $(\epsilon, \delta)$-DP such that

$$\epsilon = \min_{\alpha \in \{2, 3, \ldots\}} \frac{T \epsilon'_\alpha}{\alpha - 1} + \frac{\log(1/\delta)}{\alpha - 1}$$

where

$$\epsilon'_\alpha = \frac{1}{\alpha - 1} \log \left( 1 + \left( \frac{\ell}{n} \right)^2 \left( \frac{\alpha}{2} - 1 \right) \min \left( 4 \left( \frac{\Delta^2}{\sigma^2_X} - 1 \right), 2e^{\frac{\Delta^2}{\sigma^2_X}} + 4G(\alpha) \right) \right)$$

$$G(\alpha) := \sum_{j=3}^{\alpha} \left( \frac{\ell}{n} \right)^j \left( \frac{\alpha}{j} \right) \sqrt{B(2\lfloor j/2 \rfloor) \cdot B(2\lceil j/2 \rceil)}$$

$$B(\ell) := \sum_{i=0}^{\ell} (-1)^i \left( \frac{\ell}{i} \right) e^{\frac{\ell(1-i)}{2\sigma^2_X}} \Delta^2, \Delta^2 := \left( \frac{d_X}{\sigma^2_X} \right)$$

A less-tighter privacy-noising relation is given using Laplace distribution based noise as given below based on (3).

$$\epsilon = T \max\{A, B\} \leq \frac{T}{k\sigma}$$

where

$$A = \log \left( 1 - \frac{k}{n} + e^{\frac{k}{2\sigma^2_X}} \frac{k}{n} \right), B = \log \frac{n}{n - k + ke^{-\frac{k}{2\sigma^2_X}}}$$

7 Generation of Splinters

We now walk-through the details of generating splinters in our proposed mechanism. As $k - 1$ out of $k$ splinters used are data-independent and sampled from a different chosen distribution each. The client first generates $k - 1$ minor splinters (data-independent samples) as $R_i \sim \mathcal{N}(0, \Sigma_i), \forall i \in \{1 \ldots d - 1\}$. Only one splinter is dependent on data $X$ and is generated as

$$Z_1 = \frac{1}{\alpha_1} (X - \sum_{i \neq 1} R_i)$$

where $\alpha_1$ is the corresponding privatized version of coefficients for the data dependent splinter.

The rest of the coefficients are also secret and only known to the client and never shared.
Algorithm 1 Splintering and Unsplintering

Input: Sensitive Matrix $M$, Public Matrices $R_1, \ldots, R_{k-1}$

Client executes the following:

Use scalings for minor splinters: $R_i = \alpha_i \beta_i Z_i$

Compute Major Splinter: $Z_1 = \frac{1}{k-1} \left( M - \sum_{i=1}^{k-1} R_i \right) + N$ where $N$’s entries are sampled from a white Gaussian(10) or Laplace distribution(3).

Use Scalar DP(2) to privatize the coefficients $\alpha_i, \beta_i$ to get $\hat{\alpha}_i, \hat{\beta}_i$

Compute rest of splinters: $\hat{Z}_i = \frac{R_i}{\hat{\alpha}_i \hat{\beta}_i}$

Communicate: Send $Z_1, \ldots, Z_k$ to server

Server executes the following:

Server computes $Z_1^{-1}, \ldots, Z_k^{-1}$ and sends them back to client

Client executes the following:

Unsplintering: Client unsplinters to get $1/\alpha_1 Z_1^{-1} - 1/\alpha_1 Z_1^{-1} U (Z_0^{-1} + V 1/\alpha_1 Z_1^{-1} U)^{-1} 1/\alpha_1 V Z_1^{-1}$

Return $M^{-1}$

with the server.

Rescaling step Once the data dependent coefficient and splinter have been generated, the rest of the data-independent splinters are scaled by their corresponding privatized secret coefficients as $Z_i = \frac{1}{\hat{\alpha}_i} Z_i$. The secret coefficients are privatized using scalar DP mechanism of (2) which is an optimal mechanism. The optimality properties of scalar DP (2) are given below.

Note: The notation below that is internal to the algorithm for scalar DP should not be overloaded with any of the same symbols used in the rest of the paper for a different context.

Theorem 5. (2) Let $\varepsilon > 0, k \in \mathbb{N}$, and $0 \leq r_{\text{max}} < \infty$. Then the mechanism $\text{ScalarDP}(\cdot, \varepsilon; k, r_{\text{max}})$ is $\varepsilon$-differentially private and for $Z = \text{ScalarDP}(r, \varepsilon; k, r_{\text{max}})$, if $0 \leq r \leq r_{\text{max}}$, then $E[Z] = r$ and

$$E[(Z - r)^2] \leq \frac{k + 1}{e^\varepsilon - 1} \left[ r^2 + \frac{r_{\text{max}}^2}{4k^2} + \frac{(2k + 1) (e^\varepsilon + k) r_{\text{max}}^2}{6k (e^\varepsilon - 1)} \right] + \frac{r_{\text{max}}^2}{4k^2}$$

By choosing $k$ appropriately, we immediately see that we can achieve optimal mean-squared error as $\varepsilon$ grows.

Theorem 6. (2) Let $k = \left[ e^{\varepsilon/3} \right]$. Then for $Z = \text{ScalarDP}(r, \varepsilon; k, r_{\text{max}})$,

$$\sup_{r \in [0, r_{\text{max}}]} E[(Z - r)^2 | r] \leq C \cdot r_{\text{max}}^2 e^{-2\varepsilon/3}$$
Algorithm 2 Scalar-DP(2) using notation specific to (2)

Privatize the magnitude with absolute error: ScalarDP

Require: Magnitude \( r \), privacy parameter \( \varepsilon > 0 \), \( k \in \mathbb{N} \), bound \( r_{\text{max}} \)

\[
\text{Start by Sampling } J \in \{0, 1, \ldots, k\}
\]

such that

\[
J = \begin{cases} 
\left\lfloor kr/r_{\text{max}} \right\rfloor & \text{w.p. } (\left\lfloor kr/r_{\text{max}} \right\rfloor - kr/r_{\text{max}}) \\
\left\lceil kr/r_{\text{max}} \right\rceil & \text{otherwise.}
\end{cases}
\]

Use randomized response to obtain

\[
\hat{J} | (J = i) = \begin{cases} 
i & \text{w.p. } \frac{e^\varepsilon}{e^\varepsilon + k} \\
\text{uniform in } \{0, \ldots, k\} \backslash i & \text{w.p. } \frac{k}{e^\varepsilon + k}
\end{cases}
\]

Debias \( \hat{J} \), by setting

\[
Z = a(\hat{J} - b) \text{ for } a = \left(\frac{e^\varepsilon + k}{e^\varepsilon - 1}\right) r_{\text{max}} \text{ and } b = \frac{k(k + 1)}{2(e^\varepsilon + k)}
\]

Return \( Z \)

for a universal (numerical) constant \( C \) independent of \( r_{\text{max}} \) and \( \varepsilon \).

All the secret coefficients \( \alpha_i \) have are chosen from a \( p \)-bit base-2 floating point system allowed by the computer architecture, where \( p \in \{16, 32, 64\} \). Therefore, in order to reconstruct a data matrix \( X \) from scaled \( Z_i \)’s; one would need access to the secret coefficients. Every communication from the client to server in this scheme involves a different set of splinters sub-sampled from the union of decoy and minor splinters. That said, our scheme is a one-shot scheme per matrix inversion, there by \( T = 1 \). Moreover we assume that our matrices have a \( \ell_1 \) norm that is \( \leq 1 \). Note that for a fixed noise level, the privacy guarantee increases with an increase in \( \ell \) (number of splinters). Therefore, if one chooses a larger value of \( \ell \), then a smaller amount of noise is sufficient to achieve the target privacy level.

8 Conclusion

We provide a new scheme called splintering that is well suited for distributed and private matrix inverse computation. The scheme integrates well with any state of the art non-distributed matrix inversion scheme of choice such as (1), that the server could use in the context of splintering. We would also like to extend this scheme to other useful scientific computing operations in the distributed and private setting and thereby build a toolbox for splintering based computational pipelines.

References


2 Abhishek Bhownick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers.
23:10 REFERENCES

2 Eitan Borgnia, Jonas Geiping, Valeria Cherepanova, Liam H Fowl, Arjun Gupta, Amin Ghiasi, Furong Huang, Micah Goldblum, and Tom Goldstein. Dp-instahide: Data augmentations provably enhance guarantees against dataset manipulations.