VLDPPipeline の要点

- 目的：ローカル差分プライバシー (LDP) で露呈する *入力操作攻撃*・*出力操作攻撃* を暗号学的に防ぎつつ，ローカルモデル／シャッフルモデル両方で効率良く検証可能にする
- 結果：クライアント実行 < 2 s，サーバ検証 5–7 ms/クライアントで完了する実装を報告。

この論文の新規性

1. 初の効率的な**VLDP**スキーム **in** シャッフルモデル
    - これまでVLDPはローカルモデルのみでの構成例が知られており、シャッフルモデル向けの暗号的検証スキームは存在しなかった
2. 真正性と匿名性の相反解消
    - 入力真正性（署名付き入力）とレポートの匿名性（シャッフル後のunlinkability）は相反する要件ですが、両立を実現するために「クライアント識別情報をすべて証明の秘密部に移動」する設計を導入
3. ワンショット対話のみ
    - 乱数シード登録と署名交換は一度だけ行い、その後は各レポートに対して対話を不要とすることで、クライアント・サーバー双方の通信量とレイテンシを最小化しています

3 章 Preliminaries — 構成要素

コミットメント

- 用途: クライアントは乱数鍵 k_c を先にコミットしてから開示し、後からの値の差し替えを防ぎます。

デジタル署名：採用プリミティブ: **Schnorr 署名**

- 用途: 秘密鍵で入力 (x) | | 時刻に署名して入力真正性を示す

NIZK-PK (zk-SNARK)採用プリミティブ: Groth16 zk-SNARK

- 用途: クライアントは「Pedersen＋署名＋LDP 処理の正当性」を約 200 B の証明で提出し、サーバは 1 回のペアリングで検証します。
- 特性と採用メリット: 証明が小さく通信負荷が低い、検証はミリ秒級で高スループット；trusted setup を要しますがサーバを準信頼とする前提で許容するらしい

5 章 脅威モデル

クライアント

- プログラムは完全に悪意的＋相互に共謀可。
- ただし Trusted Enclave 内部は改竄不能

サーバ

- セミホーネスト（正規手順は守るが推測は最大化）。

シャッフラー

- **Honest-but-curious** な非協力第三者。実装は mixnet 等を想定

⬆️シャッフラーもサーバ も「正規手順を守るが、入出力の中身を全部保存してあとで解析する」モデル

役割ごとの前提

アクター	想定	なぜその設定か
クライアント	悪意的・相互に共謀可	少数の不正者でも統計を歪め得るため，最悪ケースを想定
サーバー	セミホーネスト（手順通りだが推測は行う）	大規模集計役を単純化し，TEE＋署名/NIZK による検証で安全性を担保
シャッフラー	**honest-but-curious**（Mixnet ノード相当）	出力経路を隠してプライバシー増幅を狙うが，データ改竄はしない前提
ネットワーク	盗聴可能・改竄不可能 (TLS 等)	暗号チャネルは敷設済みと仮定

【セミホーネストモデルの具体例】

- サーバー：大手ブラウザ企業やモバイル OS 提供社が統計集計サーバーを運営
    - プロトコル通りに DP 出力・証明を検証し集計するが、受動的に個別情報を解析しようとする
- シャッフラー：独立監査機関やオープンコンソーシアムがミックスノードを提供
    - メッセージを必ず並べ替えて転送するが、タイミングやパターンからメタデータを読み取ろうとする

【残存脅威】

1. 送信時間や大きさからバレる
    - 深夜だけ送れば「夜に活動する人」とわかるなど、時刻やメッセージの大きさだけで本人を推測される可能性。
2. サーバーとシャッフラーが手を組む
    - 2者がログ（並び順・時刻）を突き合わせると「誰がどのデータか」を推定可
3. 少人数データでバレる
    - 人数が少ない地域や珍しい属性だけの集計は、推定されやすい。

> 追加対策の例
> ・送信タイミングをランダムにずらす
> ・シャッフラーを複数経由にする
> ・ノイズ（ε, δ）の強さを上げる など

プロトコルの図

- 任意の数(今回はn)のCLientと　Shuffler　Serverが一つずついる
- どうやら最初に乱数を生成してる

Tariq Bontekoe, Hassan Jameel Asghar, and Fatih Turkmen

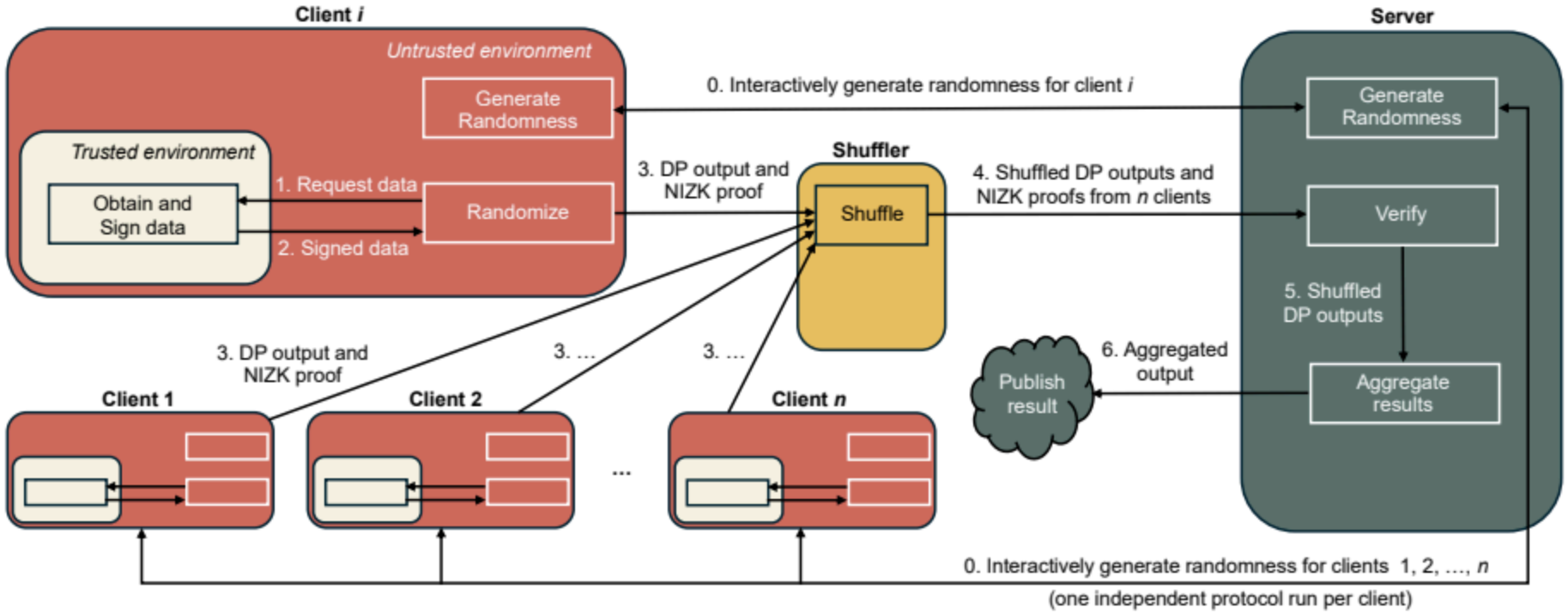

Figure 3: System model for the VLDPPipeline**. For multiple time steps** *j*, **the clients reiterate the steps as explained further on.**

登場人物と役割

- クライアント（**Client**）
  - 信頼できる環境（Secure Enclave等）で「生データを取得→署名」
  - それを使って「差分プライバシー化＋証明生成」を行い、結果を送信
- シャッフラー（**Shuffler**）
  - クライアントから受け取ったデータをバラバラに並べ替え、匿名化してサーバーへ中継
- サーバー（**Server**）
  - 受け取った出力と証明をチェック
  - 問題なければ統計を集計・公開

処理の流れ（全６ステップ）

1. 乱数の協調生成 **(**ステップ **0)**
    - クライアントとサーバーが鍵を出し合い、後で使う共通乱数 **ρ (**ロー**)** を安全に作る。
        - **ρ** : この後のノイズやシャッフル順を決める乱数シード。
2. データの署名 **(**ステップ **1→2)**
    - クライアントの信頼環境（TEE）が生データ **x** に秘密鍵で署名し、対になる署名 σ_x を付ける。
        - **x** : ユーザーの元データ。
        - σ_x : x が本物であることを示すデジタル署名。

3. プライバシー化＋証明生成 **(**ステップ **2)**
    - x と ρ を使ってノイズ付与したデータ **ẋ (**ティルダ **x)** を作り、
    - その手続きが正しいと示すゼロ知識証明 **π (**パイ**)** を生成する。
        - **ẋ** : 差分プライバシー(LDP)済みのデータ。
        - **π** :「確かにルール通りにノイズを足した」ことを示す証明。
4. 送信 **(**ステップ **3)**
    - (ẋ, π) を シャッフラーへ送る。
    - ※ローカルモデルの場合はシャッフラーを経由せず直接サーバーへ送る。

5. 並べ替え **(ステップ 4→5)**
    - シャッフラーが n 件の (ẋ, π) をランダム順に並べ替え、まとめてサーバーへ
        - **n** : 参加クライアントの総数。
6. 検証・集計 **(ステップ 5→6)**
    - サーバーが各 π を検証し、正しければ対応する ẋ を受理。
    - 全クライアント分の ẋ を集計し、最終統計を公開。

```
シャッフラーのログ例
10:00:02 ①(IP_A, 512byte)
10:00:03 ②(IP_B, 520byte)
10:00:05 ③(IP_C, 508byte)
…（ここで順序をランダムに）…
10:00:06 → サーバーへ送信 (512B)
10:00:07 → サーバーへ送信 (520B)
10:00:08 → サーバーへ送信 (508B)
```

```
サーバーのログ例
10:00:06 ①'(512B)
10:00:07 ②'(520B)
10:00:08 ③'(508B)
```

共謀すると？

- シャッフラーが「最初に送った 512B は IP_A のメッセージ」と暴露。
- サーバーは 10:00:06 に届いた 512B を照合し、「①'＝A」と復元。
- この照合を全メッセージで行えば、“並べ替え前後” が 1 対 1 で対応し匿名性が失

それぞれ「わかる／わからない」

- シャッフラー
  - *わかる*：クライアント側ネットワーク情報、到着順序。
  - *わからない*：最終的にサーバーがどの順序で受信したか。
- サーバー
  - *わかる*：検証済みデータの中身、サーバー側での到着順序。
  - *わからない*：どの IP がどのメッセージを送ったか、シャッフラー受信順序。
- 協力（ログ突き合わせ）すると「順序＋サイズ＋時刻」が合致し、誰のデータか紐付けられる危険が残る。

> 防御策のイメージ
>
> - シャッフラーを 複数段 にして 1 段でも協力しなければ安全。
> - バッチ転送や 固定サイズパディング で時刻・サイズ手掛かりを潰す。