# Differentially Private Aggregation of Distributed Time-series with Transformation and Encryption

## Part 4 — Differentially Private Aggregation 2

presenter: Le Chen

Nanyang Technological University

lechen0213@gmail.com

October 27, 2013

# Outline

# Reference

- Cynthia Dwork.
  *Differential Privacy*.
  Invited talk at ICALP, Venice, Italy, July 10-14, 2006.
  Automata, Languages and Programming, Lecture Notes in Computer Science Volume 4052, 2006, pp 1-12.
- Wikipedia.
  <http://en.wikipedia.org/wiki/Differential_privacy>
- Elaine Shi, T-H. Hubert Chan, Eleanor Rieffel, Richard Chow and Dawn Song.
  *Privacy-Preserving Aggregation of Time-Series Data*.
  In Network and Distributed System Security Symposium (NDSS), 2011.

# Reference

T-H. Hubert Chan, Elaine Shi, and Dawn Song
*Privacy-Preserving Stream Aggregation with Fault Tolerance*.
16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012. Financial Cryptography and Data Security, Lecture Notes in Computer Science Volume 7397, 2012, pp 200-214.

Vibhor Rastogi, Suman Nath
*Differentially Private Aggregation of Distributed Time-series with Transformation and Encryption*.
ACM SIGMOD 2010, New York, NY. Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pages 735-746.

# Motivation of Aggregation

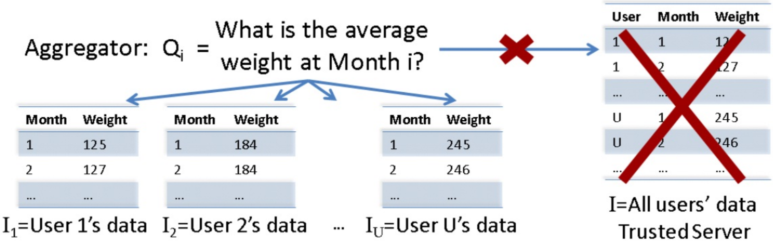



**Figure 1: System Model (Users with data $I_1, \ldots, I_U$ Aggregator issues recurring query $\mathbf{Q} = \mathbf{Q}_1, \ldots, \mathbf{Q}_n$ No trusted server has $I = I_1 \cup I_2 \ldots \cup I_U$ to evaluate $\mathbf{Q}(I)$)**

Figure: System Model

# Notations

- $I = I_1 \cup I_2 \cdots \cup I_U$ is a set of vectors.
- $I_i = [I_{i1}, I_{i2}, \cdots, I_{in}]$ is a $n \times 1$ vector, where $I_{ij} \in \{0, 1\}$ represents whether the average weight of user is over 200 lb in month $j$.
- $nbrs(I)$: the data set obtained from adding/removing one user's data from $I$.
- $Q_j(I)$ outputs "the number of users over 200 lb" in month $j$, i.e. $\sum_{n=1}^{U} I_{ikj}$.
- $Q(I) = \{Q_1(I), Q_2(I), \cdots Q_n(I)\}$.

# LPA (Laplace Perturbation Algorithm)

- By adding $LAP(\lambda)$ noise to the final results, $LPA(Q, \lambda)$ is $\epsilon$-differentially private for $\lambda = \Delta(Q)/\epsilon$.
- $\tilde{Q} = Q(I) + LAP^n(\lambda)$.
- DLPA: Each user $u$ adds its noise $n_u$ to $x_u$.
- $\tilde{Q} = \sum_u (x_u + n_u) = Q(I) + \sum_u n_u$.

# Basic Distributed Protocol

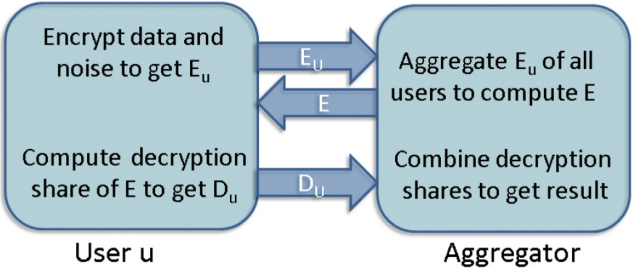



**Figure 2: Basic Distributed Protocol** (homomorphic property exploited to aggregate users' encryption & threshold property to combine users' decryption shares)

# Distributed Decryption

- Distributed keys: The private key $\lambda$ is shared by $U$ users as $\lambda = \sum_u \lambda_u$.
- Each user $u$ computes his decryption share $c_u = c^{\lambda_u}$.
- The aggregator combines $c' = \prod_{u=1}^{U} c_u$.
- The final result $t = \frac{L(c' \mod N^2)}{L(g^\lambda \mod N^2)} \mod N$.

# Protocol for Computing Exact Sum

- Encrypt-Sum$(x_u, r_u)$: the encryption of $\sum_{u=1}^{U}(x_u + r_u) = Q + \sum_{u=1}^{U} r_u$.
- Decrypt-Sum$(c, r_u)$: each user $u$ computes a decryption share $c'_u = c^{\lambda_u} g^{-r_u \lambda}$.
- Final result: the aggregator aggregates $c' = \prod_{u=1}^{U} c'_u$ and gets $Q = \frac{L(c' \mod N^2)}{L(g^{\lambda} \mod N^2)} \mod N$.

# Protocol for Computing Noisy Sum

- Let $Y_i \sim N(0, \lambda)$ for $i \in \{1, 2, 3, 4\}$ be four Gaussian random variables. Then $Z = Y_1^2 + Y_2^2 - Y_3^2 - Y_4^2$ is a $Lap(2\lambda^2)$ random variable.

**Algorithm 5.4** Encrypt-Noisy-Sum($x_u, r_u$)

1: User $u$ chooses five random numbers $r_u^1, r_u^2, \ldots, r_u^5$ from $\mathbb{Z}_m$ and computes $r_u = r_u^1 + r_u^2 - r_u^3 - r_u^4 + r_u^5$.
2: User $u$ generates four $N(0, \sqrt{2\lambda}/U)$ random variables $y_u^1, \ldots, y_u^4$.
3: Let $c^j$ =Encrypt-Sum-Squared($y_u^j, r_u^j$) for $j \in \{1, 2, 3, 4\}$.
4: Let $c^5$ =Encrypt-Sum($x_u, r_u^5$)
5: Aggregator computes $c = \frac{c^1 c^2 c^5}{c^3 c^4}$.

# Protocol for Computing Noisy Sum

**Algorithm 5.3** Encrypt-Sum-Squared($y_u, r_u$) Protocol

1: User $u$ computes $c_u = Enc(y_u + a_u + b_u)$ and sends it to the aggregator.
2: The aggregator computes $c = \prod_{u=1}^{U} c_u$ and sends it to each user $u$.
3: Each user $u$ generates a random $r_u \in \mathbb{Z}_m$, computes $c_u = c^{y_u - a_u + b_u} Enc(r_u)$.
4: The aggregator collects $c_u$ from each user and computes $c' = (\prod_{u=1}^{U} c_u) Enc(a^2)$

where $a = \sum_u a_u$, $Enc(a^2)$ is public, and $\sum_u b_u = 0$.

# $L_2$ Sensitivity

- If $Q$ is a query sequence, $Q(I)$ and $Q(I')$ are each vectors.
- $L_1$ distance metric, denoted as $|Q(I) - Q(I')|_1$, that measures the Manhattan distance $\sum_j |Q_j(I) - Q_j(I')|$ between these vectors.
- $L_2$ distance metric, denoted as $|Q(I) - Q(I')|_2$, that measures the Euclidean distance $\sqrt{\sum_j (Q_j(I) - Q_j(I'))^2}$.

# Sensitivity

DEFINITION 2.2 (SENSITIVITY [7]). *Let* $\mathbf{Q}$ *be any query sequence. For* $p \in \{1, 2\}$, *the* $L_p$ *sensitivity of* $\mathbf{Q}$, *denoted* $\Delta_p(\mathbf{Q})$, *is the smallest number such that for all* $I$ *and* $I' \in nbrs(I)$,

$$\left|\mathbf{Q}(I) - \mathbf{Q}(I')\right|_p \leq \Delta_p(\mathbf{Q})$$

*For a single snapshot query* $Q_i$, *the* $L_1$ *and* $L_2$ *sensitivities are the same, and we write* $\Delta(Q_i) = \Delta_1(Q_i) = \Delta_2(Q_i)$.

# Sensitivity Example

EXAMPLE 2.1. *Consider a query $Q$ counting the number of users whose weight in month 1 is greater than 200 lb. Then $\Delta(Q)$ is simply 1 as $Q$ can differ by at most 1 on adding/removing a single user's data. Now consider $\mathbf{Q} = Q_1, \ldots, Q_n$, where $Q_i$ counts users whose weight in month $i$ is greater than 200 lb. Then $\Delta_1(\mathbf{Q})$ is $n$ (for the pair $I, I'$ which differ in a single user having weight $>$ 200 in each month $i$) and $\Delta_2(\mathbf{Q}) = \sqrt{n}$ (for the same pair $I, I'$).*

# DFT (Discrete Fourier Transform)

- The DFT of an $n$-dimensional sequence $X$ is a linear transform giving another $n$-dimensional sequence.
- $DFT(X)$: the $j^{th}$ element $DFT(X)_j = \sum_{i=1}^{n} e^{\frac{2\pi\sqrt{-1}}{n}ji} X_i$.
- $DFT(X)$: the $j^{th}$ element of the Inverse DFT is $IDFT(X)_j = \frac{1}{n}\sum_{i=1}^{n} e^{-\frac{2\pi\sqrt{-1}}{n}ji} X_i$.
- Furthermore, $IDFT(DFT(X)) = X$.

# $DFT^k(X)$

- Denote $DFT^k(X)$ as the first $k$ elements of $DFT(X)$.
- The elements of $DFT^k(X)$ are called the Fourier coefficients of the $k$ lowest frequencies and they compactly represent the high-level trends in $X$.
- An approximation $X'$ to $X$ can be obtained from $DFT^k(X)$ as follows:
  - $PAD^n(DFT^k(X))$: appends $n-k$ 0 to $DFT^k(X)$,
  - compute $X' = IDFT(PAD^n(DFT^k(X)))$.

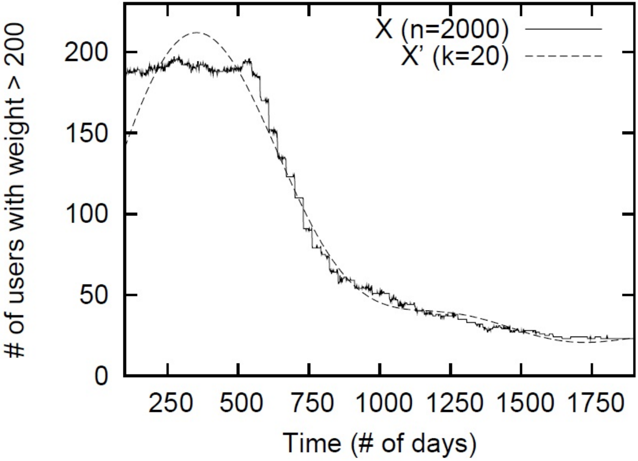
# $DFT^k(X)$

- Obviously $X'$ may be different from $X$ as ignoring the last $n-k$ Fourier coefficients may introduce some error.
- Denote $RE_j^k(X)$, short for reconstruction error at the $j^{th}$ position, to be the value $|X_j' - X_j|$.

# Reconstruction Error

EXAMPLE 4.1. *To give a sense of the reconstruction error, we consider a sequence* $\mathbf{X}$ *of length* $n = 2000$ *representing the number of people with weight* $> 200$ *in a real dataset (more details in Section 7), counted once every day over 2000 days. Fig. 3 shows the reconstructed sequence,* $\mathbf{X}'$*, using* $k = 20$ *DFT coefficients along with the original sequence* $\mathbf{X}$*.* $\mathbf{X}$ *shows the temporal trend in the # of overweight people in the dataset. As shown,* $\mathbf{X}'$ *captures the trend accurately showing that the reconstruction error is small even when compressing from* $n = 2000$ *to* $k = 20$ *DFT coefficients.*

# Reconstruction



# FPA (Fourier Perturbation Algorithm)

**Algorithm 4.1** FPA$_k$(**Inputs:** sequence $\mathbf{Q}$, parameter $\lambda$)

1: Compute $\mathbf{F^k} = \mathbf{DFT}^k(\mathbf{Q}(I))$.
2: Compute $\tilde{\mathbf{F}}^k = \text{LPA}(\mathbf{F^k}, \lambda)$
3: Return $\tilde{\mathbf{Q}} = \mathbf{IDFT}(\mathbf{PAD}^n(\tilde{\mathbf{F}}^k))$

- The parameter $\lambda$ in $FPA_k$ needs to be adjusted in order to get $\epsilon$-differential privacy.
- Since $FPA_k$ perturbs the sequence $F^k$, $\lambda$ has to be calibrated according to the $L_1$ sensitivity, $\Delta_1(F^k)$, of $F^k$.

THEOREM 4.1. *Denote* $\mathbf{F^k} = \mathbf{DFT}^k(\mathbf{Q}(I))$ *the first* $k$ *DFT coefficients of* $\mathbf{Q}(I)$. *Then, (i) the* $L_1$ *sensitivity,* $\Delta_1(\mathbf{F^k})$, *is at most* $\sqrt{k}$ *times the* $L_2$ *sensitivity,* $\Delta_2(\mathbf{Q})$, *of* $\mathbf{Q}$, *and (ii)* $FPA_k(\mathbf{Q}, \lambda)$ *is* $\epsilon$*-differentially private for* $\lambda = \sqrt{k}\Delta_2(\mathbf{Q})/\epsilon$.

# Proof

- (i) holds since $\Delta_2(F^k) \leq \Delta_2(Q)$, as the $n$ Fourier coefficients have the same $L_2$ norm as $Q$, while $F^k$ ignores the last $n-k$ Fourier coefficients,
- and $\Delta_1(F^k) \leq \sqrt{k}\Delta_2(F^k)$, due to a standard inequality between the $L_1$ and $L_2$ norms of a sequence.
- Let $x_j = |Q_j(I) - Q_j(I')|$, then $\Delta_1(Q) = \sum_j |Q_j(I) - Q_j(I')|$ $= \sum_j x_j$, $\Delta_2(Q) = \sqrt{\sum_j (Q_j(I) - Q_j(I'))^2} = \sqrt{\sum_j x_j^2}$. Since the inequality

$$\frac{\sum_j x_j}{n} \leq \sqrt{\frac{\sum_j x_j^2}{n}}$$

holds, we can see that $\Delta_1(F^k) \leq \sqrt{k}\Delta_2(F^k)$.

# Proof

- (ii) follows since for $\lambda = \sqrt{k}\Delta_2(Q)/\epsilon \geq \Delta_1(F^k)/\epsilon$,
- $\tilde{F}^k = LPA(F^k, \lambda)$ computed in step 2 is $\epsilon$-differential private, and $\tilde{Q}$ in step 3 is obtained using $\tilde{F}^k$ only.

# Accuracy

THEOREM 4.2. *Fix* $\lambda = \sqrt{k}\Delta_2(\mathbf{Q})/\epsilon$ *so that* $FPA_k(\mathbf{Q}, \lambda)$ *is* $\epsilon$-*differentially private. Then for all* $i \in \{1, \ldots, n\}$, *the* $error_i(FPA_k)$ *is* $k/\epsilon + RE_i^k(\mathbf{Q}(I))$.

- The theorem shows that the error by $FPA_k$ for each query is $k/\epsilon + RE_i^k(Q(I))$, but the LPA yields an error of $n/\epsilon$.
- Since the reconstruction error, $RE_i^k(Q(I))$, is often small even for $k << n$, we expect the error in $FPA_k$ to be much smaller than in LPA.
- This hypothesis is confirmed in our experiments that show that $FPA_k$ gives orders of magnitude improvement over LPA in terms of error.

# Choosing the Right $k$

- So far we have assumed that $k$ is known to us.
- Since $error_i(FPA_k)$ is $k/\epsilon + RE_i^k(Q(I))$, a good value of $k$ is important in obtaining a good trade-off between the perturbation error, $k/\epsilon$, and the reconstruction error, $RE_i^k(Q(I))$.
- If $k$ is too big, the perturbation error becomes too big, while if $k$ is too small the reconstruction error becomes too high.

# Choosing the Right $k$

- We can often choose $k$ based on prior assumptions about $Q(I)$.
- For instance, if $Q(I)$ is such that the Fourier coefficients corresponding to $Q(I)$ decrease exponentially fast, then only a constant number (say $k = 10$) of Fourier coefficients need to be retained during perturbation.
- Our experiments show that this naive method is applicable in many practical scenarios as Fourier coefficients of many real-word sequences decrease very rapidly.

# Conclusion & Discussion

- Last week, we introduced an aggregation protocol supports distributed differential privacy and distributed decryption.
- This week we talked about using (DFT) Discrete Fourier Transform in LPA (Laplace Perturbation Algorithm) to get FPA (Fourier Perturbation Algorithm), and discussed the error of DFT and chosen of $k$.
- FPA can achieve the same differential privacy level with less error, the noise actually reduces by a factor of $n/k$.