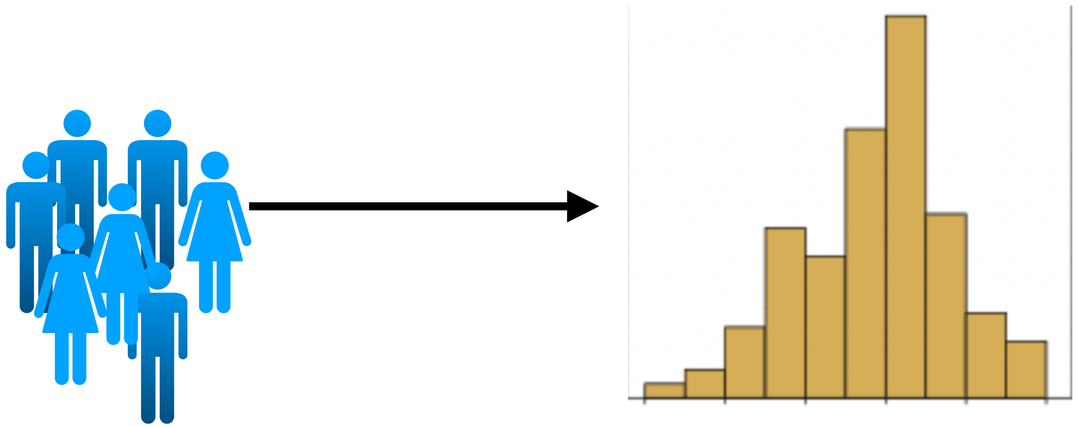
Verifiable Differential Privacy For When The Curious Become Dishonest

Ari Biswas, University Of Warwick/Amazon

Joint work with
Graham Cormode, University Of Warwick/Meta AI

Motivating Problem: DP Histograms



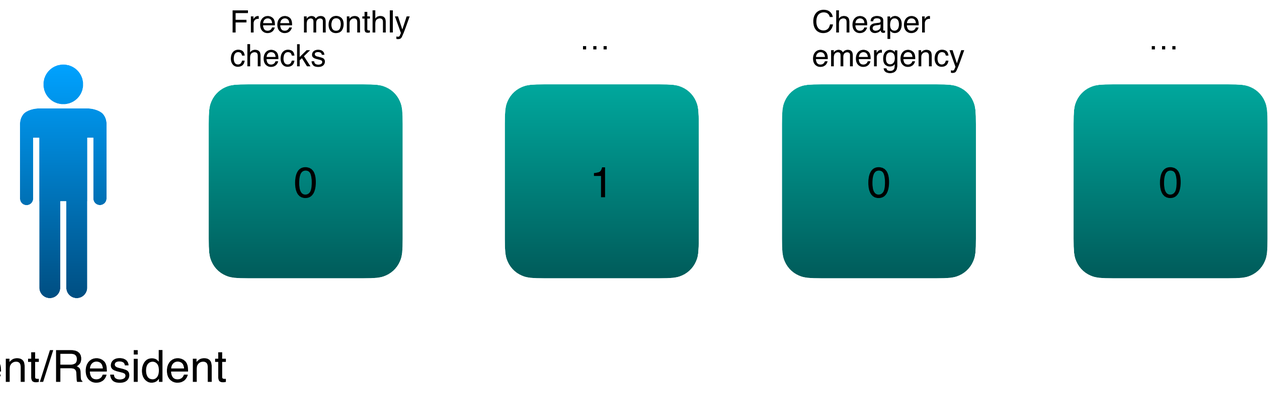
The local government of Wolvercote, a small village in Oxfordshire want to know if they should increase **yearly** public healthcare spending.

In order to gauge public opinion they conduct a survey over the population of the village

Motivating Problem: DP Histograms

Survey Prompt:

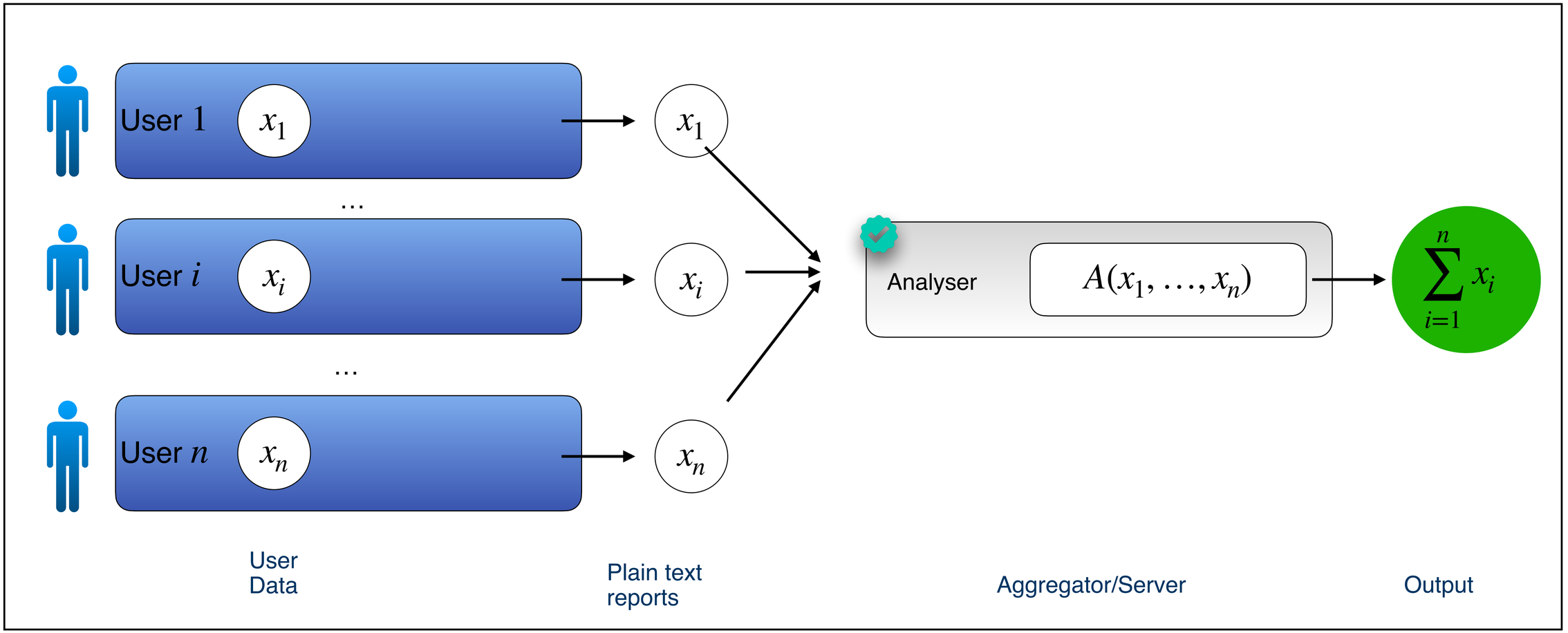
Out of the following M, please select the one that best describes your health needs.



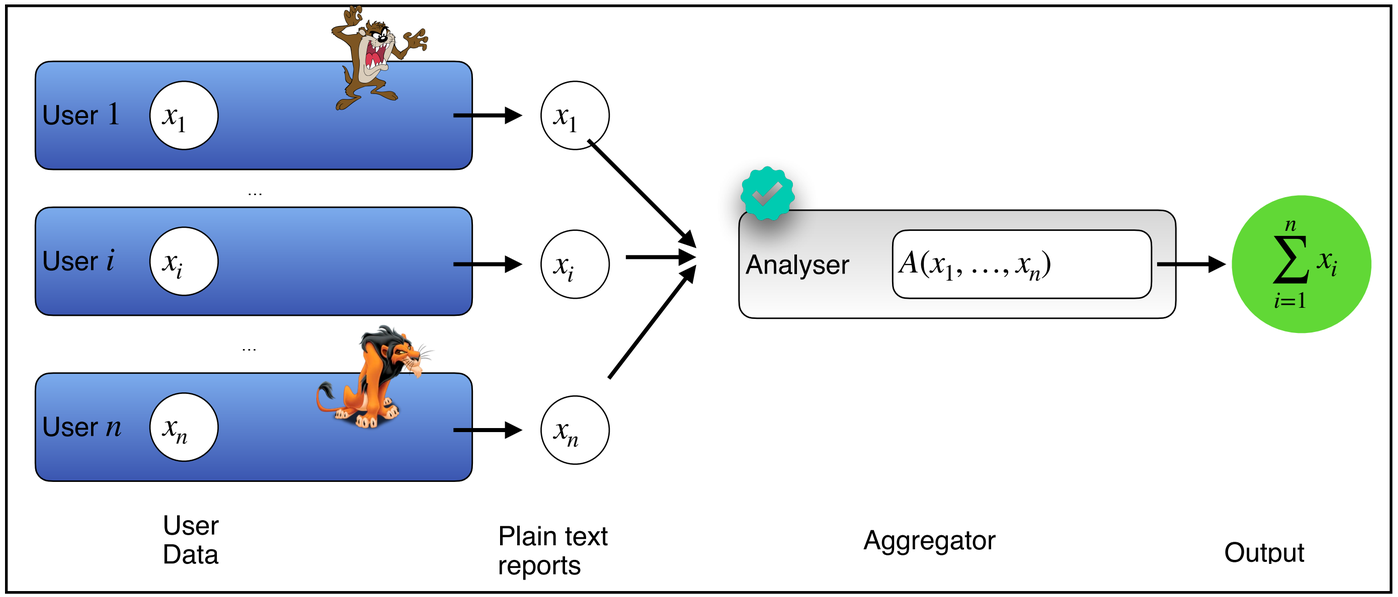
Users can only select one out of M options, i.e. they can vote for only one coordinate/candidate.

Input Format: We will assume that the user's input or vote is always a one-hot encoded vector $x \in \mathbb{Z}_q^M$ where q is a large prime number (much larger than n, the number of residents)

An Ideal Solution



Notation And Assumptions



Will always assume secure point to point and broadcast channels.

Ideal/Trusted Party

Passive Adversary/Honest but curious/ Semi honest

Follow all prescribed protocols but try and learn as much additional information from protocol transcript:

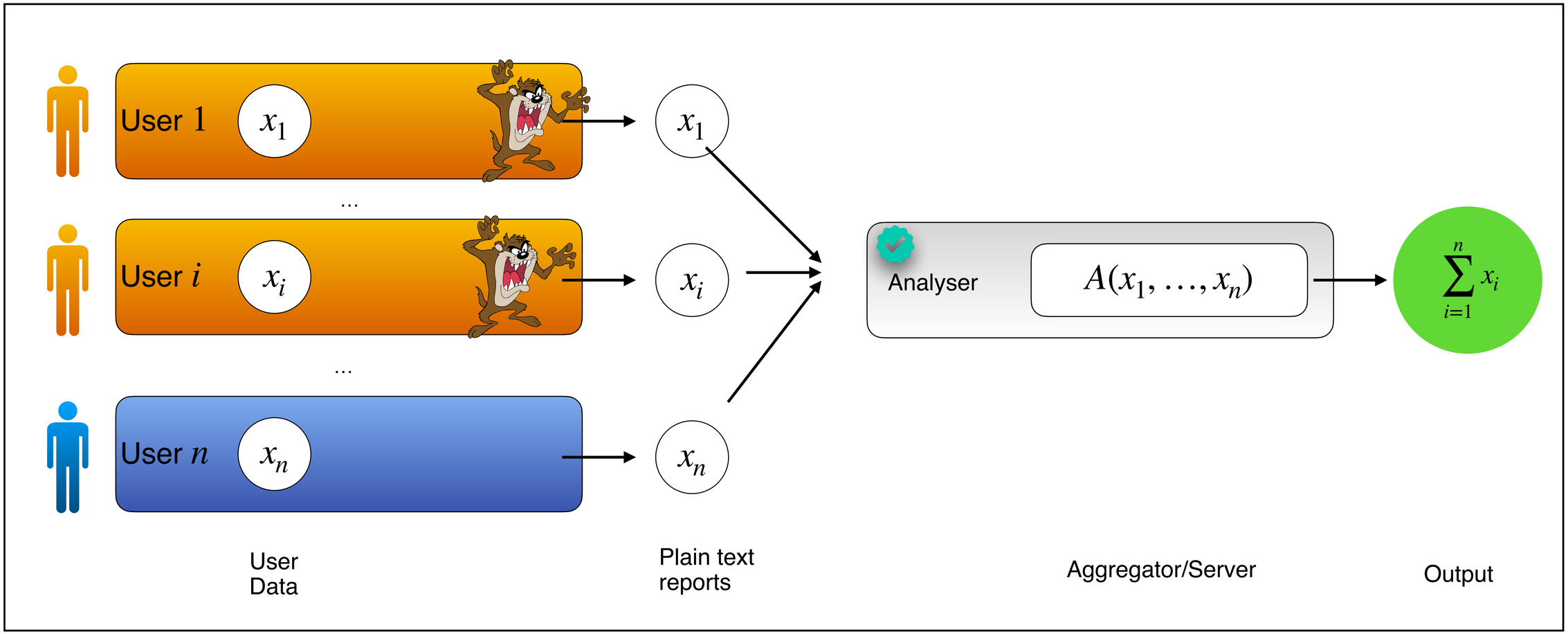
Static Active Adversaries

Deviates from prescribed protocol arbitrarily but is computationally bounded

PPT Adversary

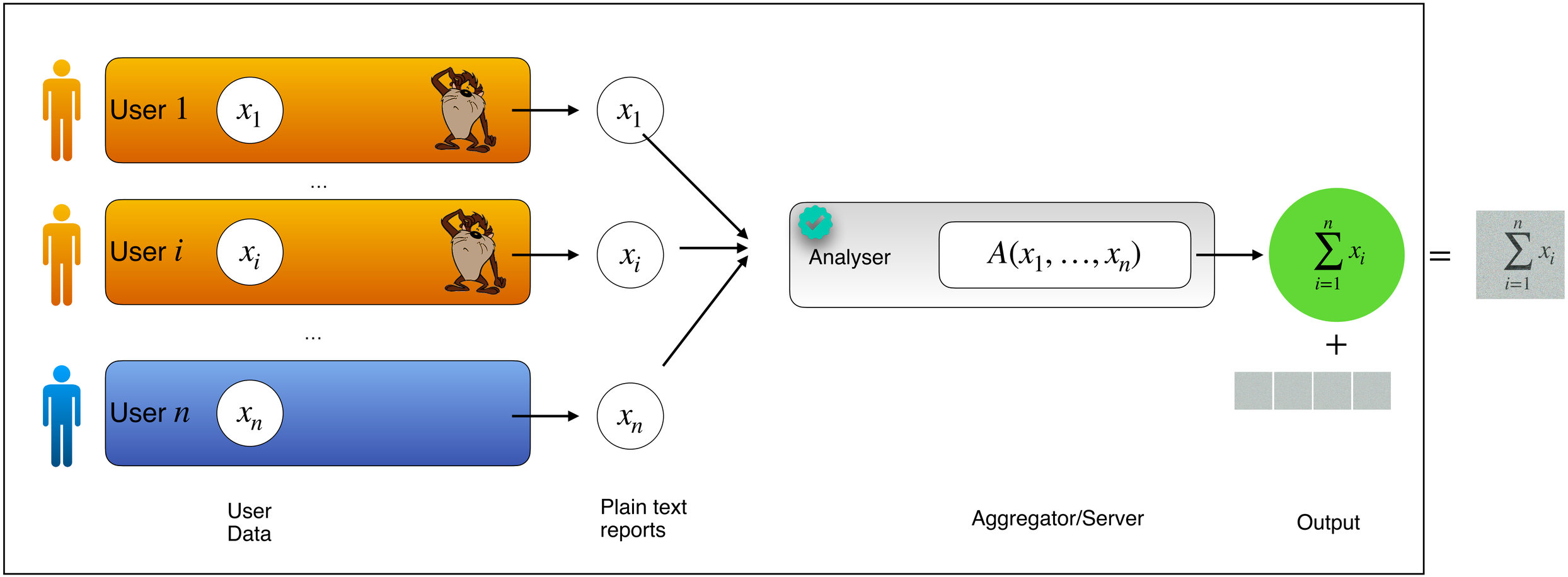
Unbounded Adversary

Adversary Controls $n-1$ Users



User n is new to the village, and by comparing with last years numbers his input is compromised

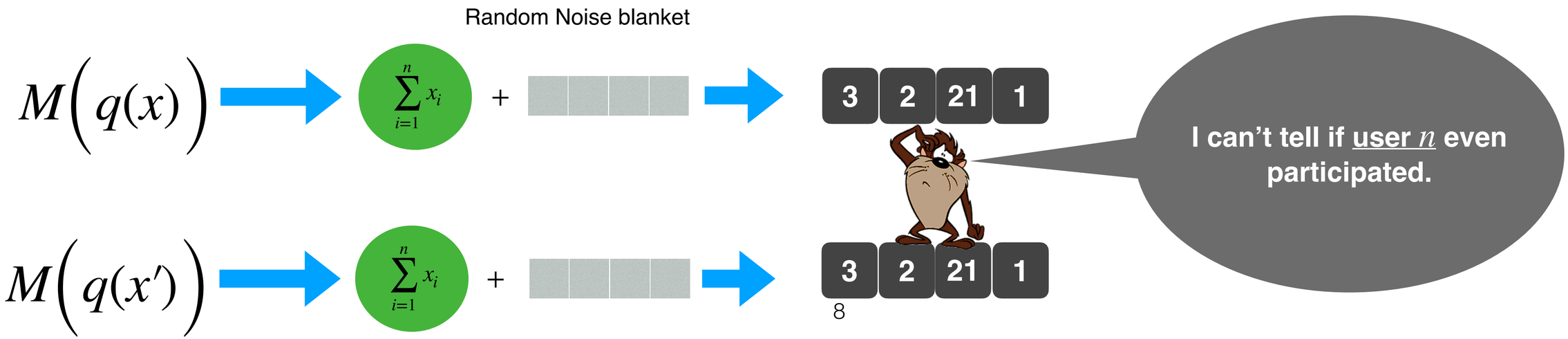
Add Random Noise To Protect The User



How Much Noise? Differentially Private Noise

An algorithm $M : (X^n \times Q) \rightarrow Z$ satisfies (ϵ, δ) differentially private if for every two neighbouring datasets $x \sim x' \in X^n$ and for every query $q \in Q$ we have

$$\forall T \subseteq Z,\ \mathbb{P}[M(x, q) \in T] \leq e^{\epsilon}\, \mathbb{P}[M(x', q) \in T] + \delta$$



How Do You Add Differentially Private Noise

K-incremental functions

A function $q : \mathcal{X}^n \to \mathbb{Z}^M$ is called k-incremental if for all neighbouring datasets $X \sim X'$ we have $||q(X) - q(X')||_\infty \leq k$. T**he histogram counting query is 1-incremental.**

Noise from smooth distribution

$$M\Big(q(X)\Big) := \underbrace{q(X)}_{k\text{-incremental function}} + \vec{Y}$$

$M\Big(q(X)\Big)$ is $(\epsilon\Delta, \delta\Delta)$ differentially private, where $(Y_1, \ldots, Y_M) \overset{\text{i.i.d}}{\sim} D$ where Δ is the global sensitivity of q.

Ghazi, B., Golowich, N., Kumar, R., Pagh, R. and Velingker, A., 2021, October. On the power of multiple anonymous messages: Frequency estimation and selection in the shuffle model of differential privacy.

How Do You Add Differentially Private Noise

Smooth Distributions

A distribution D over $\mathbb{Z}$ is (ϵ, δ, k)-**smooth** if for all $k' \in [-k, k]$ we have

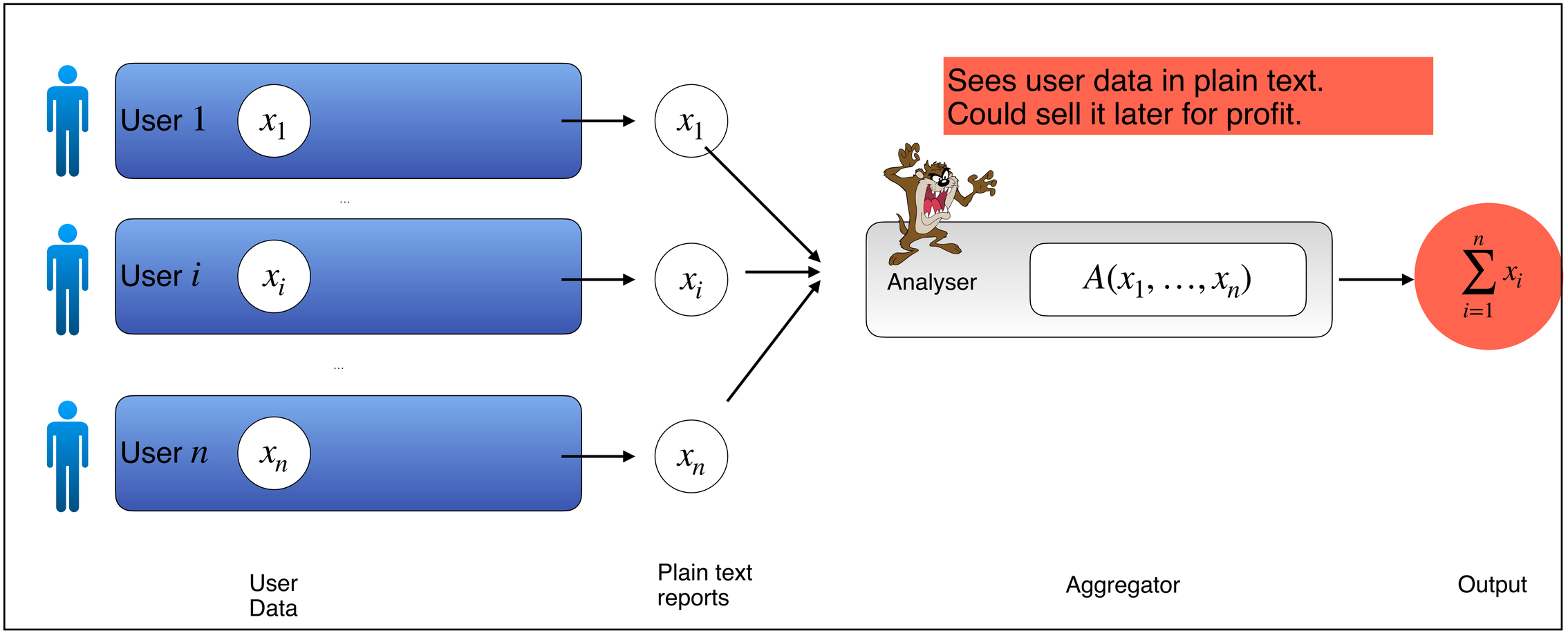
$$\Pr_{Y \sim D}\left[\frac{\Pr_{Y' \sim D}[Y' = Y]}{\Pr_{Y' \sim D}[Y' = Y + k']} \geq e^{|k'|\epsilon}\right] \leq \delta$$

Ghazi, B., Golowich, N., Kumar, R., Pagh, R. and Velingker, A., 2021, October. On the power of multiple anonymous messages: Frequency estimation and selection in the shuffle model of differential privacy.

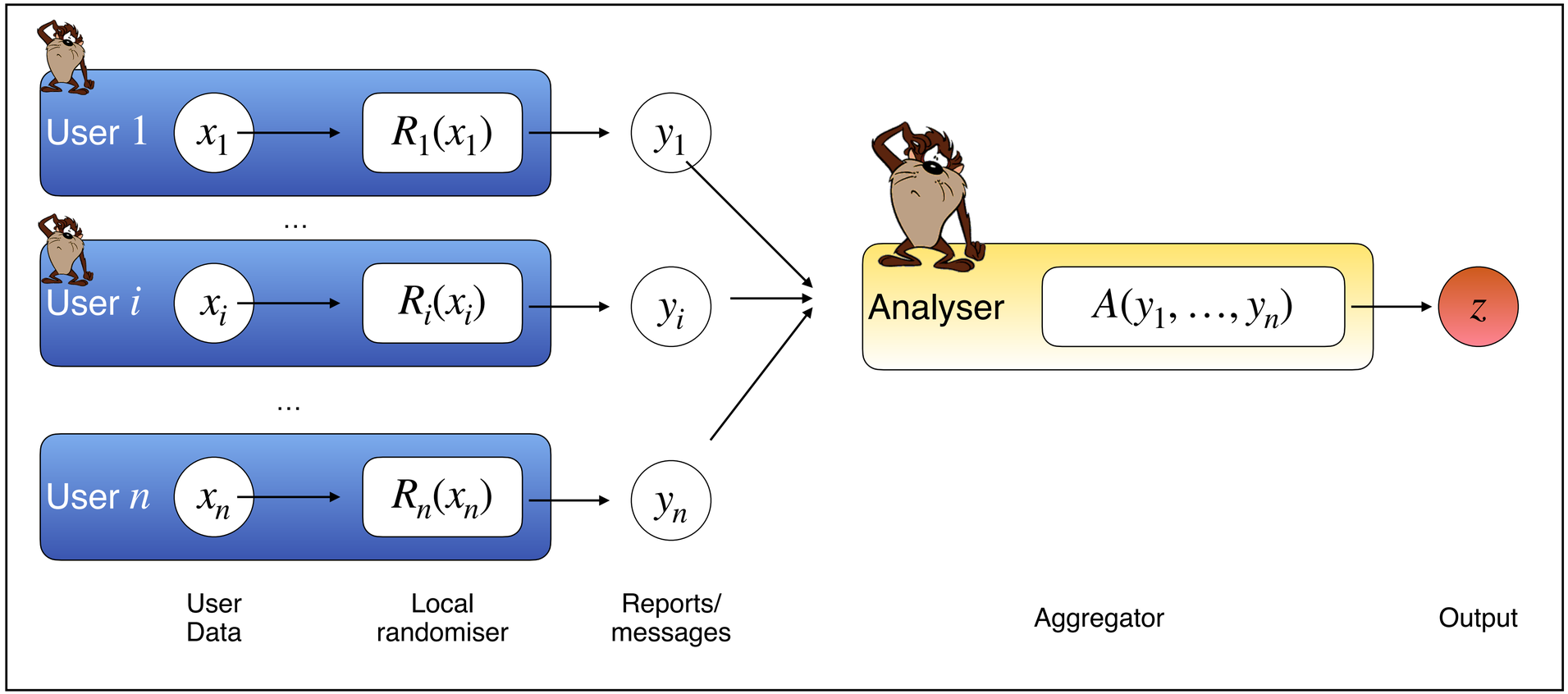
Examples of Smooth distributions:

Binomial, Gaussian, Poisson, Negative Binomial, Laplace

What If I Cannot Trust The Curator Itself



Local Differential Privacy: Noisify At The Other End

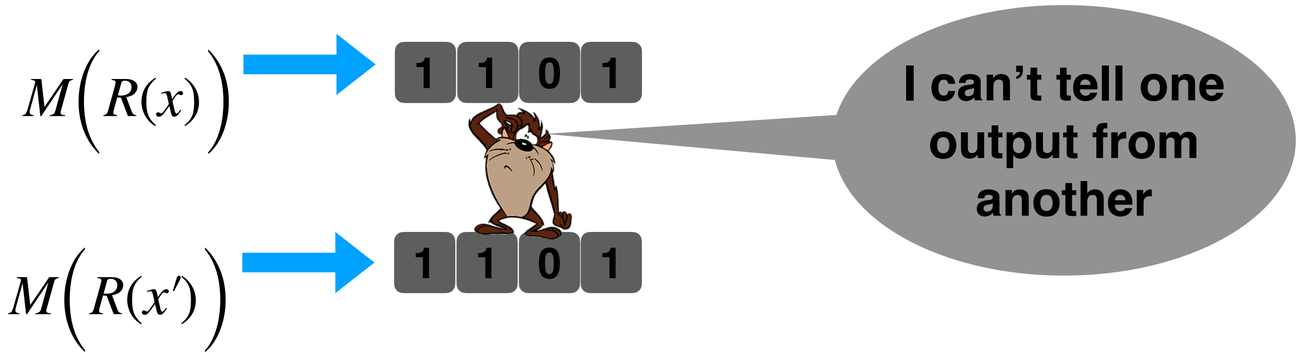


When not mentioned specifically we assume $|x_i| = |y_i|$

Local Differential Privacy

An algorithm $R : X \to Y$ satisfies (ϵ, δ) local differential privacy if for every two users $x, x' \in X$

$$\forall T \subseteq Y,\ \mathbb{P}[R(x) \in T] \leq e^{\epsilon}\, \mathbb{P}[R(x') \in T] + \delta$$



Randomised Response Is LDP

Let $p \in (0,1/2)$

$$y_i = \begin{cases} x_i & \text{with pr. } \frac{1}{2} + p \\ 1 - x_i & \text{with pr. } \frac{1}{2} - p \end{cases}$$

$f = \sum_{i=1} x_i$ What we want to estimate

$\hat{f} = \sum_{i=1}^{n} \left[\frac{1}{2p}(y_i - 1/2 + p) \right]$ What we estimate

$\mathbb{E}[\hat{f}] = f$

(In expectation they are the same)

LDP error $|\hat{f} - f| \leq O(\frac{\sqrt{n}}{\epsilon})$ ← Unavoidable

Central error $|\hat{f} - f| \leq O(\frac{1}{\epsilon})$

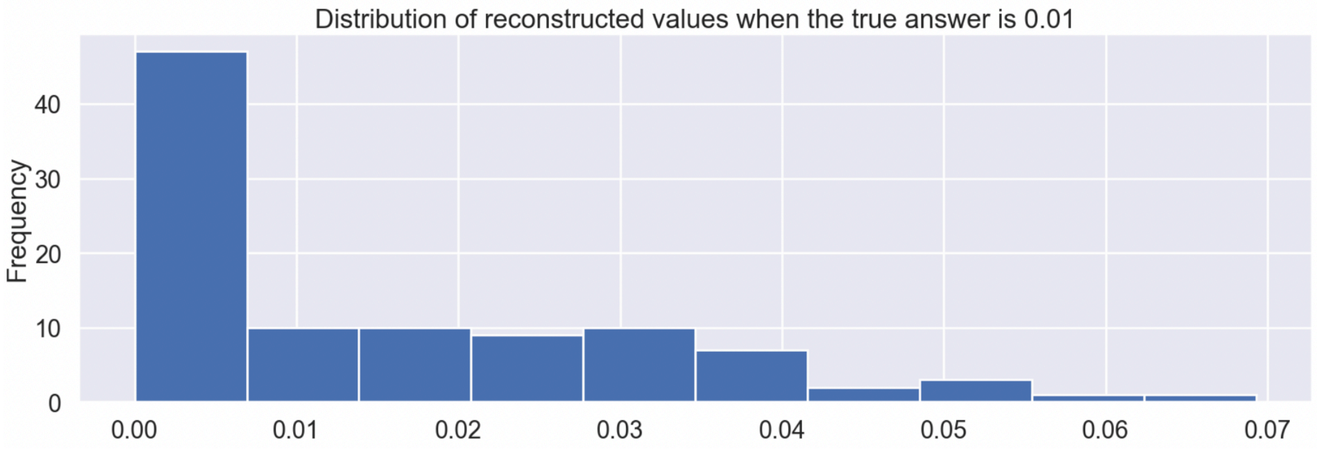
Randomised Response is optimal in LDP

Duchi, Jordan, and Wainwright, 'Local Privacy, Data Processing Inequalities, and Statistical Minimax Rates'.

If you can attack Randomised Response, you can attack all LDP algorithms. Thus Randomised Response generalises LDP algorithms.

Cheu, Smith, and Ullman, 'Manipulation Attacks in Local Differential Privacy'.

Local Differential Privacy Is Quite Weak In Terms Of Privacy



1 million people in the survey.
1% of the population says YES

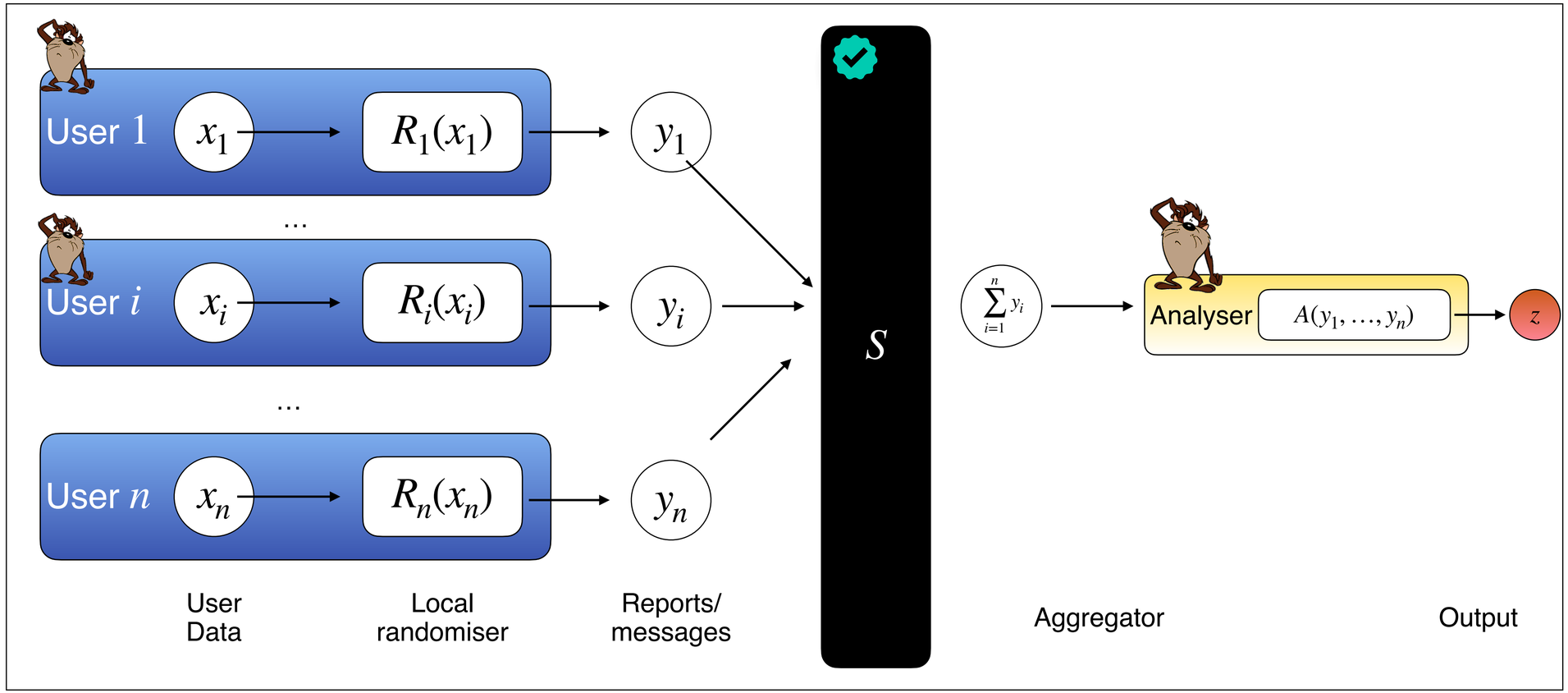
With $p = 0.49$
Roughly 1/3 of the time, randomised response claims 0% of the population said YES

If $p < 0.1$, then at least 90% of the time the central server observes the users true input — this is quite poor when compared to semantic security provided by cryptography

Comparison is unfair — LDP imposes stricter privacy constraints than central privacy

Each user has to generate enough to noise to hide himself as opposed to each user has to generate enough noise to hide amongst the crowd.

Shuffle Privacy



If we do not restrict $|x| = |y|$ we can do just as well as central

Balcer and Cheu, 'Separating Local & Shuffled Differential Privacy via Histograms'.

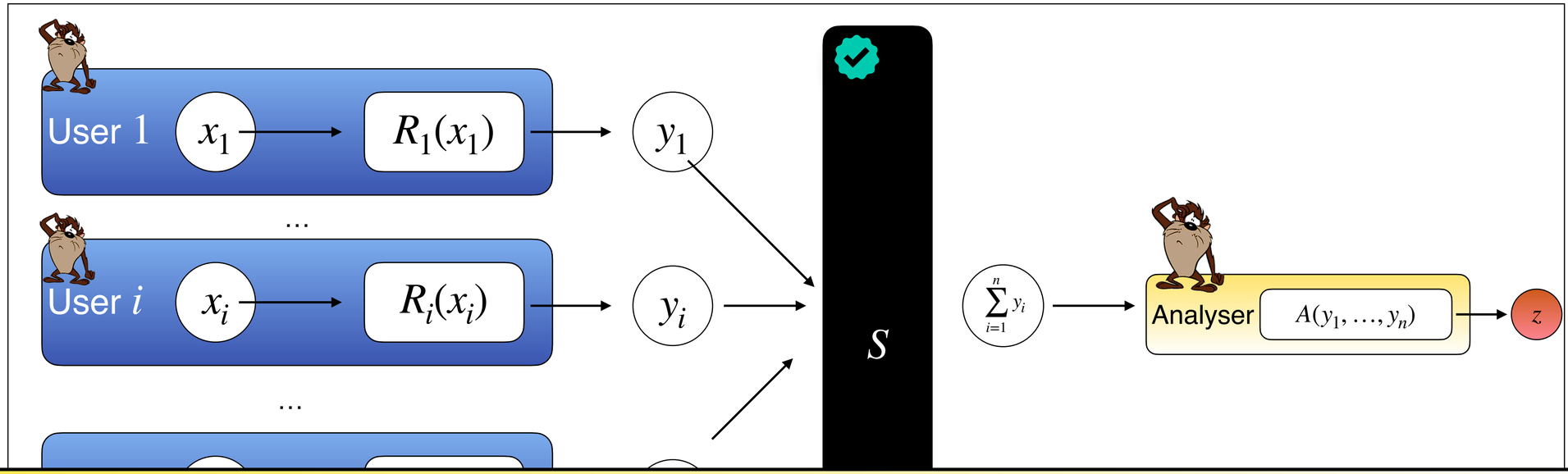
If we do not restrict $|x| = |y|$ we can do just as well as central

Ghazi et al., 'Differentially Private Aggregation in the Shuffle Model'.

Randomised Response gives near central error

Balle et al., 'The Privacy Blanket of the Shuffle Model'.

Shuffle Privacy



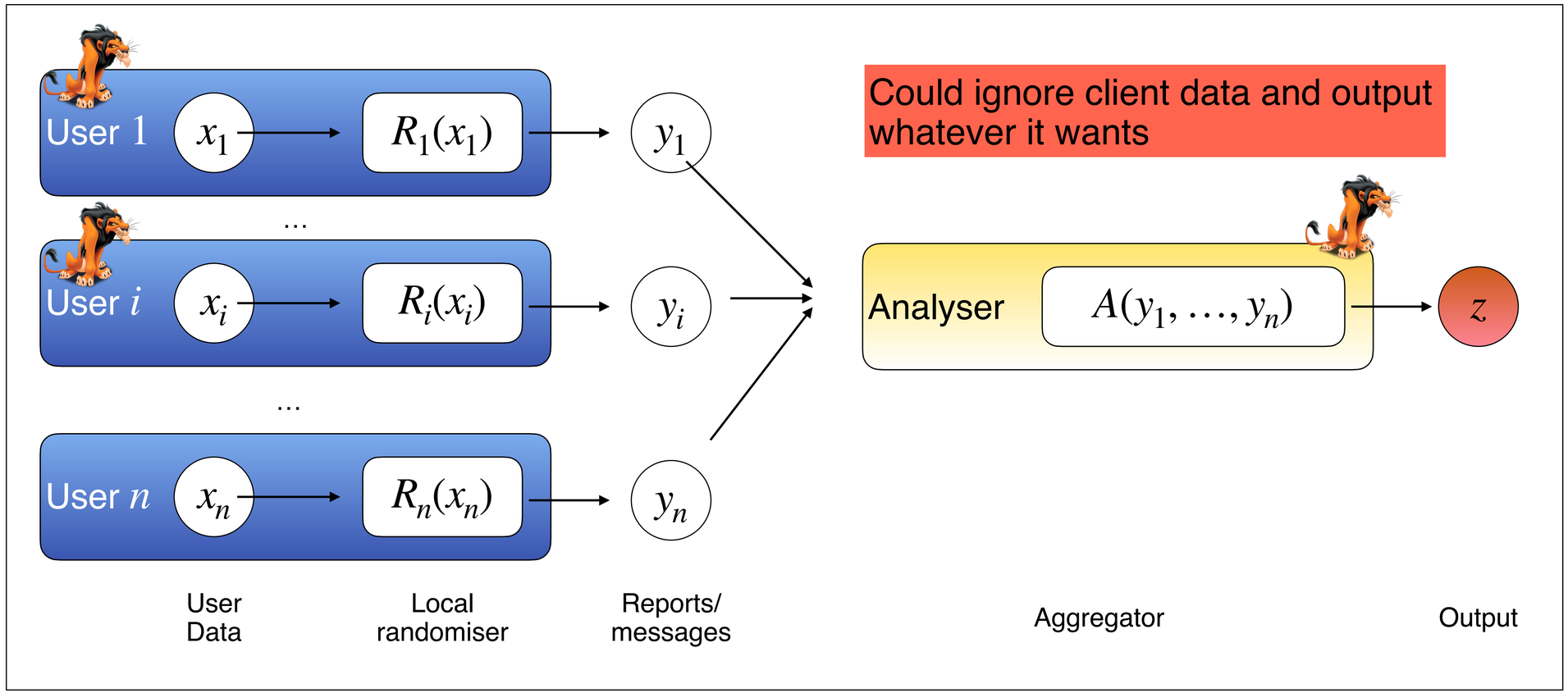
Rich line of work with semi-honest models

If we do
just as w

Balcer ar
Shuffled

al

What Happens When The Curious Become Dishonest?



Without the guarantee of semi-honest behaviour, impossible to distinguish between a corrupt run of the protocol and an honest run with different parameters.

Cheu, Smith, and Ullman, ‘Manipulation Attacks in Local Differential Privacy’.

End Of Act I: Summary

1. Under central privacy the central analyser has too much power. They can see user inputs in plaintext and can output whatever they want. We are at the mercy of this central analyser.

2. Local privacy is a very weak definition of privacy and the error rates are too high.

3. Shuffle privacy — an attempt to bridge the gap between local and central privacy still requires a secure shuffler/secure aggregator which is not efficient under the single curator model. Not immune to malicious clients. Although the analyser cannot see plaintext reports, it can manipulate the output as it desires

ACT II: Distributed Privacy

Find $k \geq 2$ aggregators with conflicting interests.

For example:
In an election, pick 1 analyser leaning right and the other leaning left.

Analyser 1

Analyser 2

…

Analyser k

Key Assumption

Don't have enough money to corrupt all K curators. Let's hope that
as long as at least **1** server is semi honest we will be fine

What Is Possible Under These New Assumptions

- Perfect Privacy of inputs + Differential privacy + Output is untampered.

Not possible: Need number of bad actors to be strictly less than $\frac{K}{3}$

Ben Or, Goldwasser and Widgerson, ‘Completeness theorems for non cryptographic fault tolerant distributed computation’

- Perfect Privacy of inputs + Differential privacy + Output is not guaranteed to be meaningful

Poplar: The focus is on lightweight protocols and the emphasis is on privacy

Boneh et al., ‘Lightweight Techniques for Private Heavy Hitters’.

- Computational privacy of inputs + differential privacy + output is “guaranteed” to be meaningful
  - If an adversary violates any of this, the honest server detects this and tells everyone that they cheated and voids the protocol.

OUR WORK

K aggregators and only guaranteed that 1 server is semi honest

Schwartz-Zippel Lemma

*Let $P(x_1, \ldots, x_n)$ be a **non-zero polynomial** of total degree $d \geq 0$ over a field $\mathbb{F}$. Let S be a finite subset of $\mathbb{F}$ and let $r_1, \ldots, r_n$ be selected at random independently and uniformly from S. Then*

$$\Pr\left[P(r_1, \ldots, r_n) = 0\right] \leq \frac{d}{|S|}$$

Informal defintion

Linear Additive Secret Sharing

- Two algorithms SHARE and RECONSTRUCT
- For any $s \in \mathbb{Z}_q$
- $\texttt{SHARE}(s; k) = [s]_1, \ldots, [s]_K$
  - such that $[s]_i \xleftarrow{R} \mathbb{Z}_q$ for $i \in [K-1]$

    and $[s]_K = s - \sum_{i=1}^{K-1} [s]_i (\mod q)$
- $\texttt{RECONSTRUCT}([s]_1, \ldots, [s]_K) = \sum_{i=1}^{K} [s]_i (\mod q) = s$

Adversary in possession of $K-1$ shares learns no Shannon information about s

Example in $\mathbb{Z}_{11}$ and $K = 3$

Secret $s = 7$

$\texttt{SHARE}(s) =$

$[s]_1 = 4$

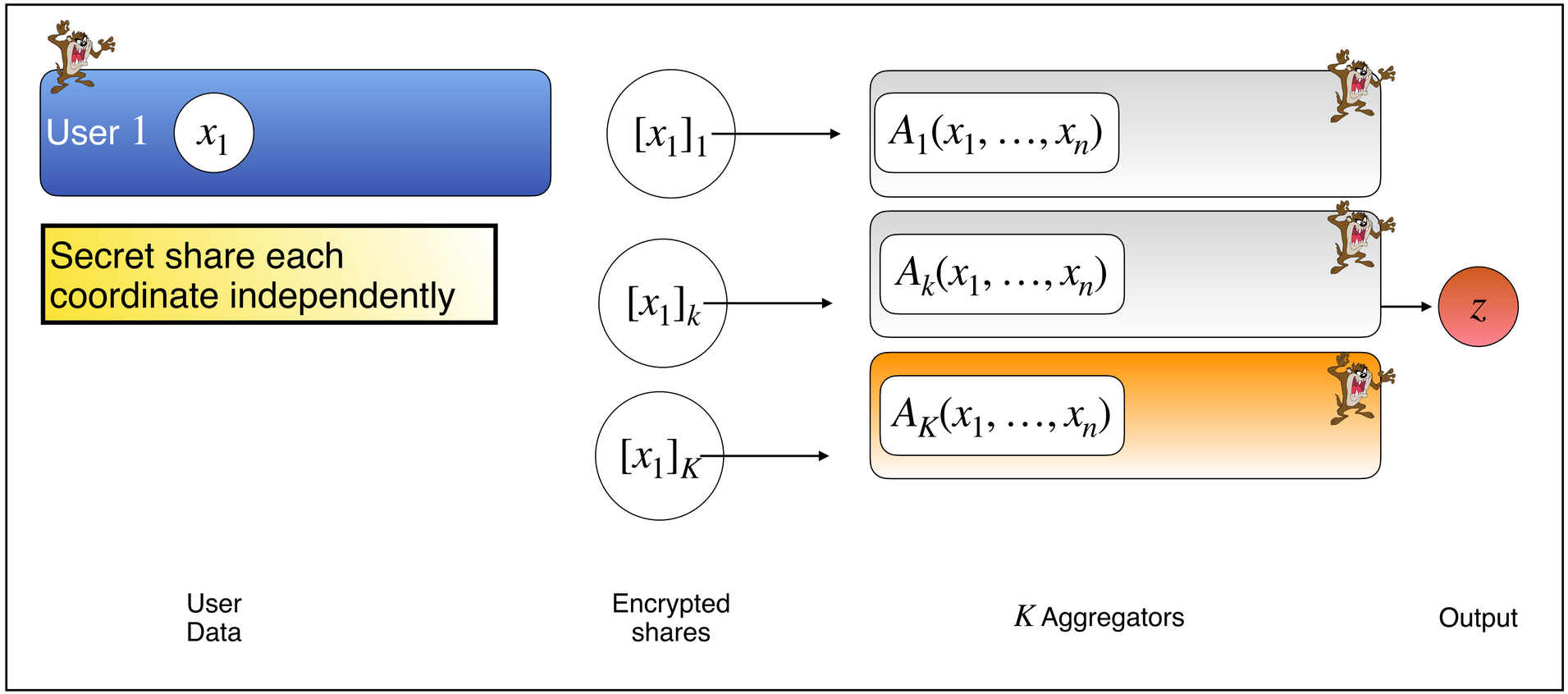
$[s]_2 = 5$

$[s]_3 = 9$

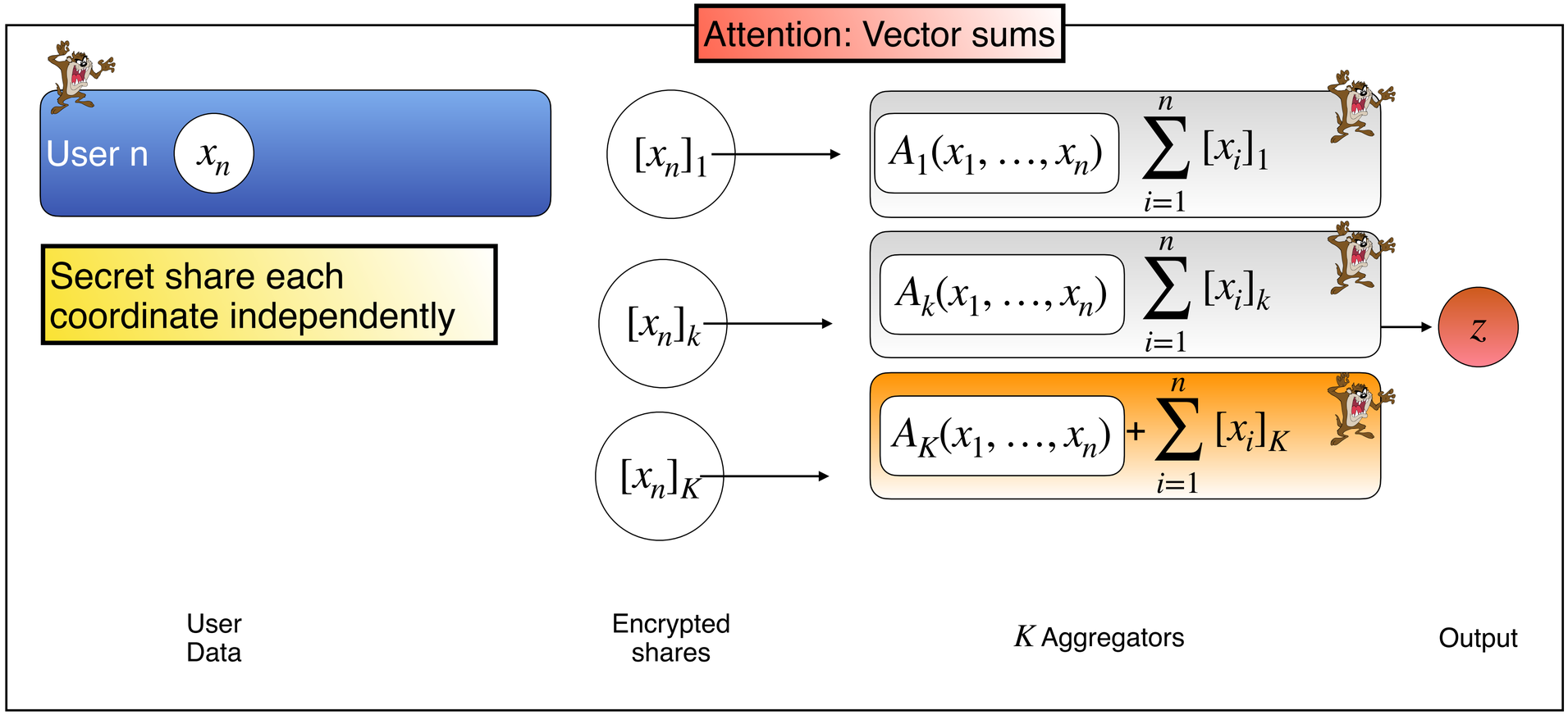
$\texttt{RECONSTRUCT}:$

$$\sum_{i=1}^{3} [s]_i = (4 + 5 + 9) \mod 11 = 7$$

PRIO - Boneh *et al. (Semi-Honest Servers)*

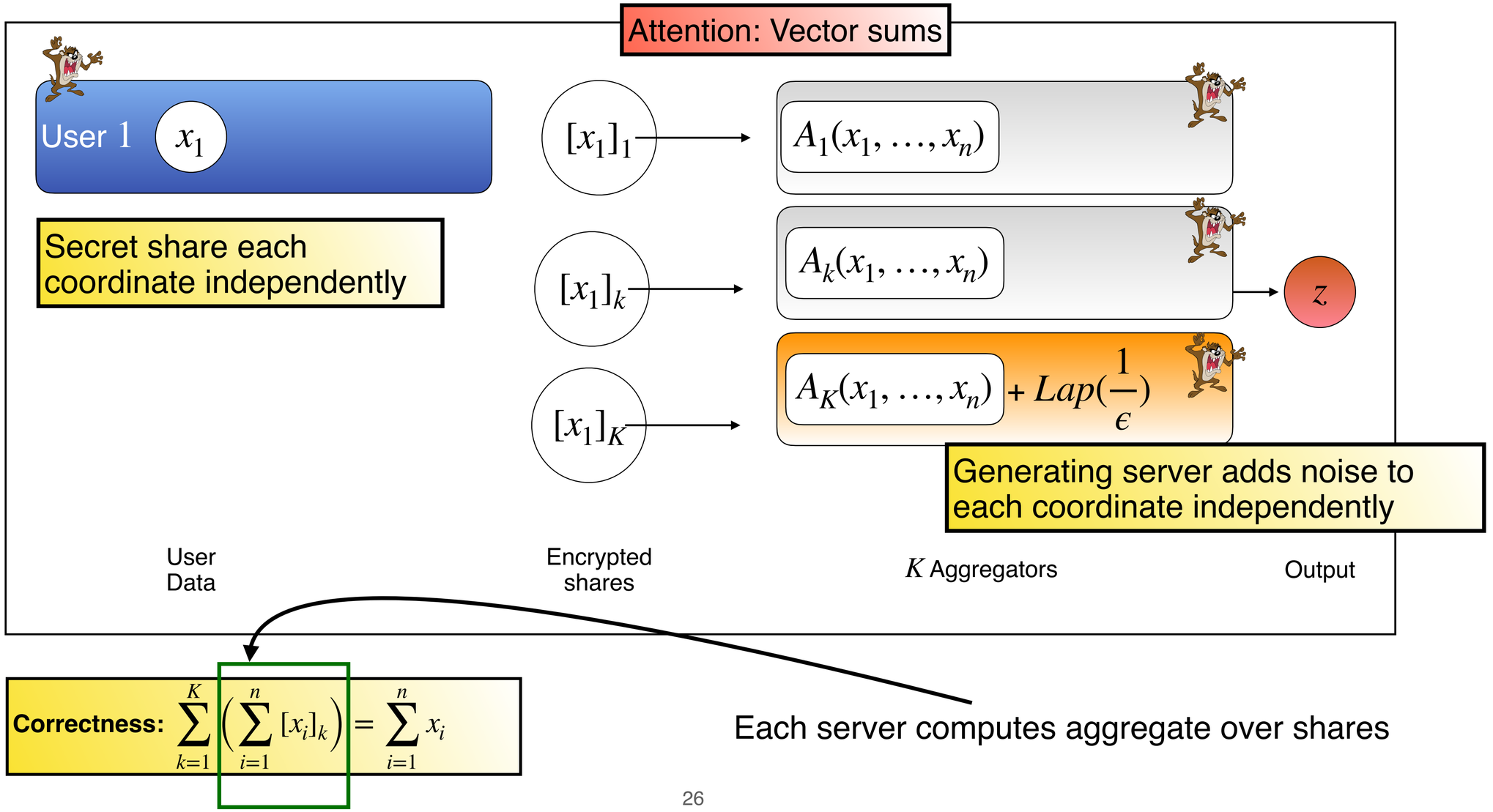


PRIO - Boneh *et al. (Semi-Honest Servers)*

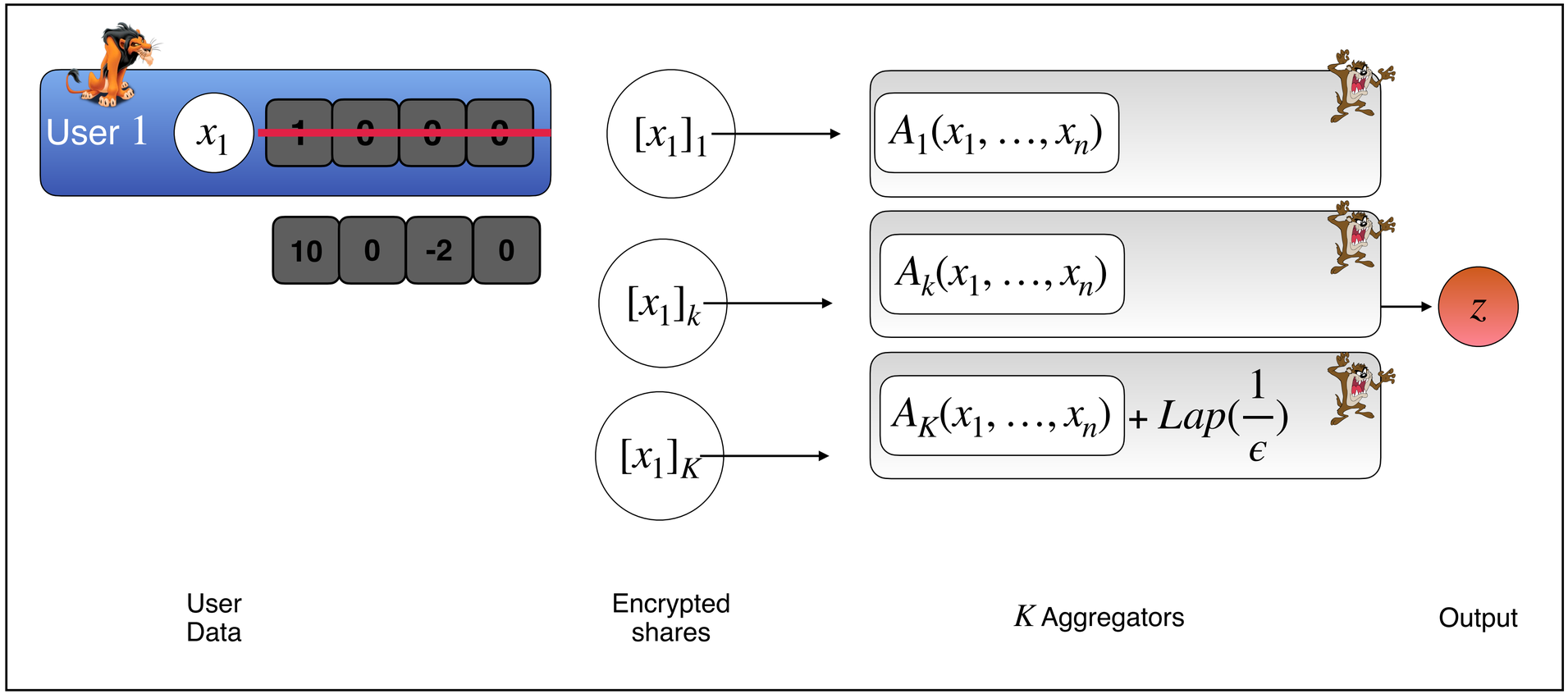


Each server computes aggregate over shares and broadcasts these aggregates to each other.

PRIO - Boneh *et al. (Semi-Honest Servers)*



Ballot Stuffing



Now a single client can bias the output of the entire protocol arbitrarily — VERY BAD!

Defending Against Malicious Clients

$[x]_1 \in \mathbb{Z}_q^M$

$[x]_3 \in \mathbb{Z}_q^M$

$[x]_2 \in \mathbb{Z}_q^M$

Sketching protocol from work in 2016 on function secret sharing

Boyle, Gilboa, and Ishai, ‘Function Secret Sharing’.

Defending Against Malicious Clients

$r_1, \ldots, r_M$

$[x]_1 \in \mathbb{Z}_q^M$

$[x]_3 \in \mathbb{Z}_q^M$

$r_1, \ldots, r_M$

$r_1, \ldots, r_M$

$[x]_2 \in \mathbb{Z}_q^M$

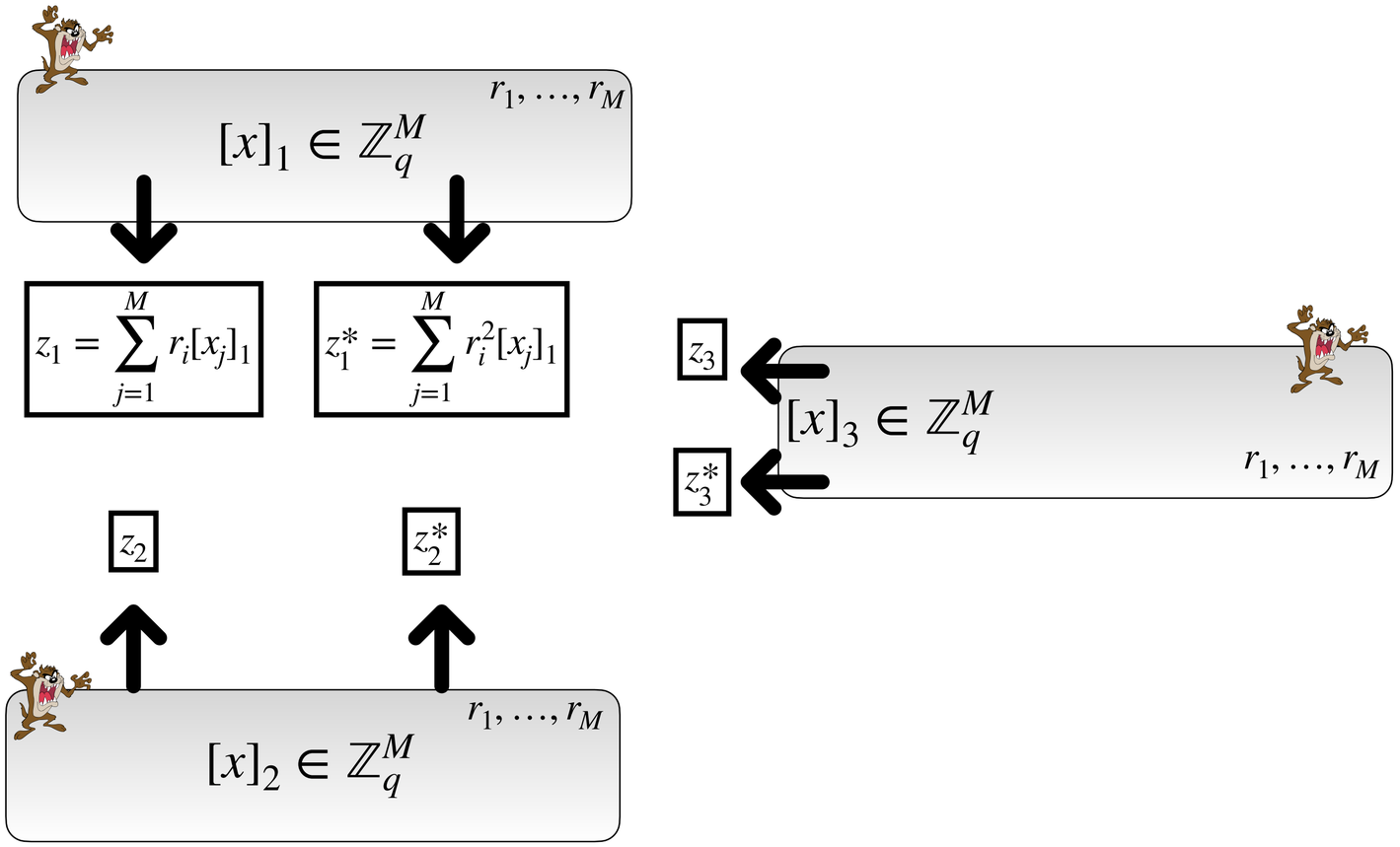
1. Server 1 samples $r_1, \ldots, r_M$ where $r_i \xleftarrow{R} \mathbb{Z}_q$ independently and broadcasts it to other servers

INTUITION:

We will create a degree two M variate degree 2 polynomial $p(\vec{r})$ such that if $\vec{r}$ is not a root, then the only way to 0 out this polynomial is to have a single non zero entry equal to 1.

The cheating client does not know the values of $\vec{r}$ thus with probability $\frac{2}{q}$ fails to pass to the test

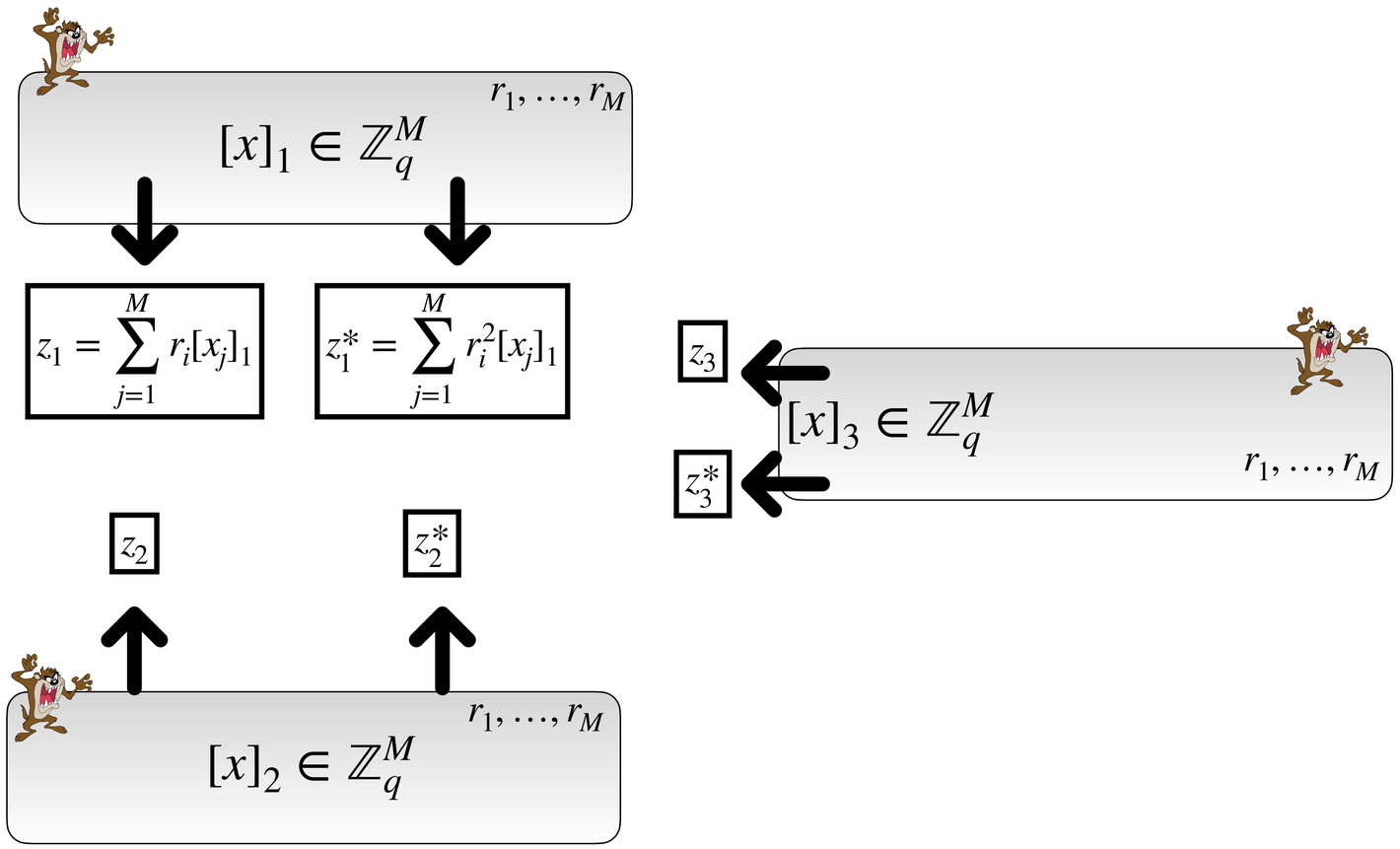
Sketching Protocol



1. Server 1 samples $r_1, \ldots, r_M$ where $r_i \xleftarrow{R} \mathbb{Z}_q$ independently and broadcasts it to other servers
2. Server k broadcasts $z_k = \sum_{j=1}^{M} r_i[x_j]_k$ and

   $z_k^* = \sum_{j=1}^{M} r_i^2[x_j]_k$

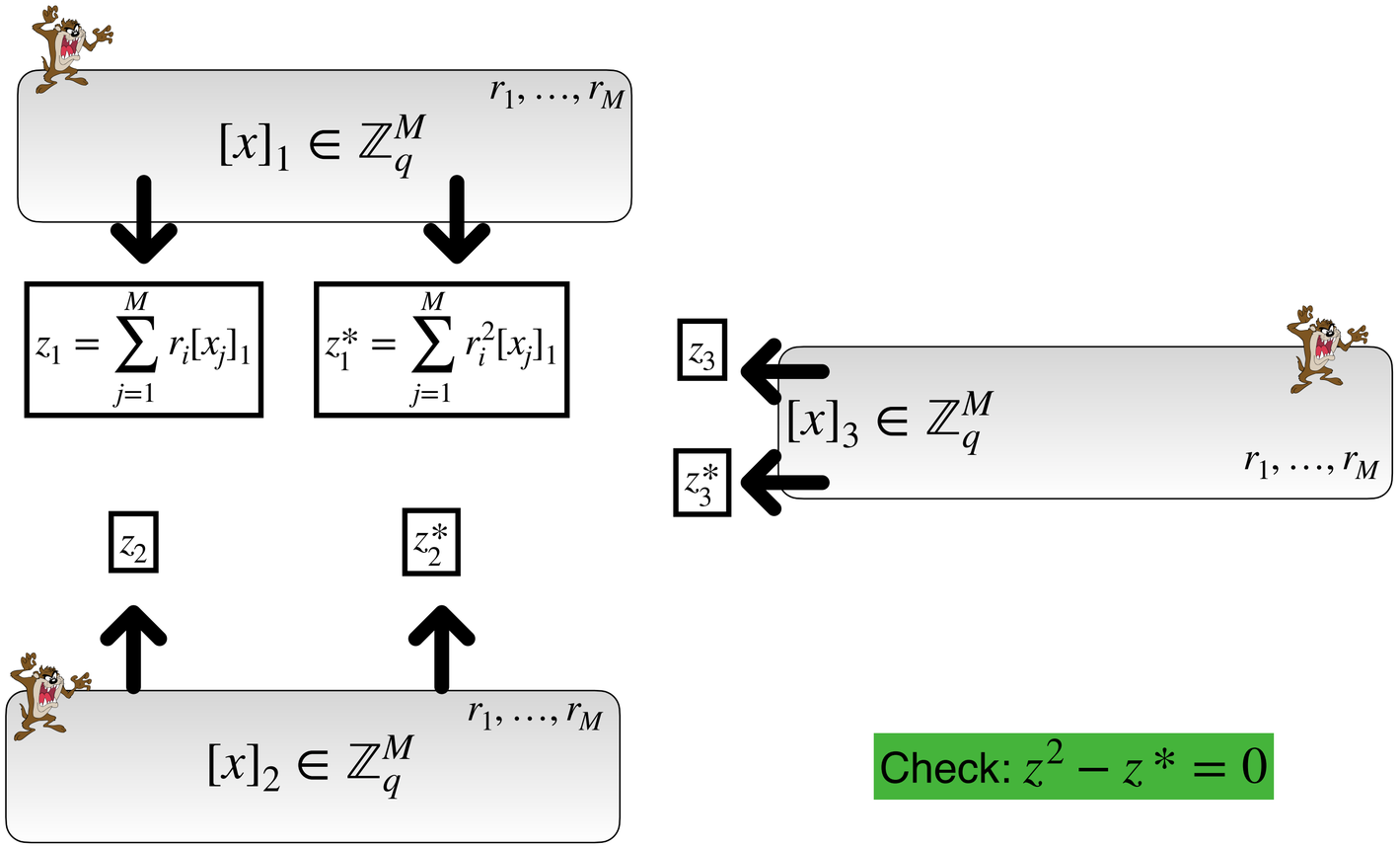
Sketching Protocol



1. Server 1 samples $r_1, \ldots, r_M$ where $r_i \xleftarrow{R} \mathbb{Z}_q$ independently and broadcasts it to other servers
2. Server k broadcasts $z_k = \sum_{j=1}^{M} r_i[x_j]_k$ and

   $z_k^* = \sum_{j=1}^{M} r_i^2[x_j]_k$
3. Each server computes $z = \sum_{i=1}^{3} z_i$ and $z^* = \sum_{i=1}^{3} z_i^*$

Sketching Protocol



1. Server 1 samples $r_1, \ldots, r_M$ where $r_i \xleftarrow{R} \mathbb{Z}_q$ independently and broadcasts it to other servers
2. Server k broadcasts $z_k = \sum_{j=1}^{M} r_i[x_j]_k$ and
   $z_k^* = \sum_{j=1}^{M} r_i^2[x_j]_k$
3. Each server computes $z = \sum_{i=1}^{3} z_i$ and $z^* = \sum_{i=1}^{3} z_i^*$

Check: $z^2 - z^* = 0$

$$p(r_1, \ldots, r_M) = \sum_{i \in [M]} r_i^2[x]_i([x]_i - 1) + 2 \sum_{i \neq j} r_i r_j [x]_i [x]_j$$

Protocol Overview

Voting:

Clients send input shares to servers

Verifying:

Servers collaboratively verify that the input is legal

Each server aggregates

$$\sum_{i=1}^{n} [x_i]_k$$

The analysers are all semi-honest, next we will see how to deal with actively adversarial servers

Protocol Overview

Voting:

Clients send input shares to servers

Verifying:

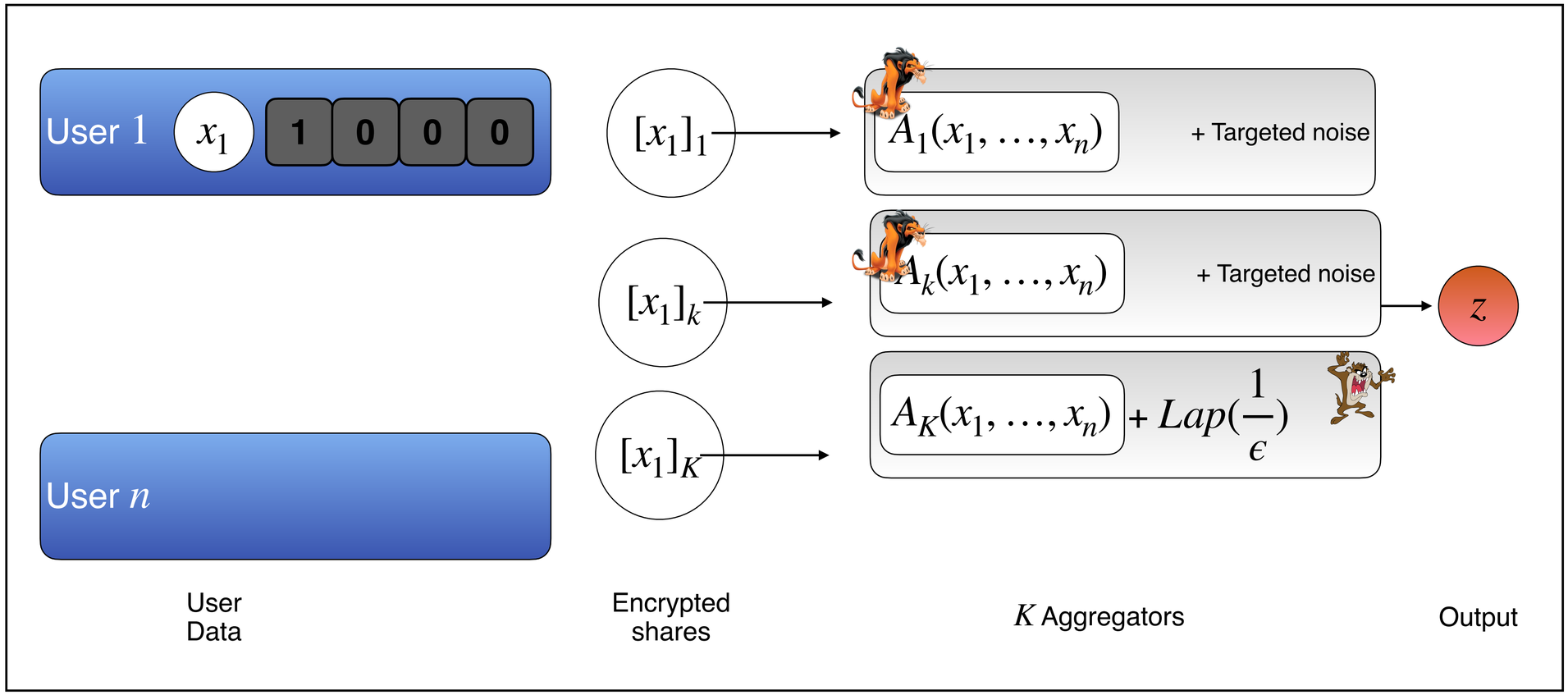
Servers collaboratively verify that the input is legal

Broadcast

$$\mathrm{Lap}(\frac{1}{\epsilon}) + \sum_{k=1}^{K}\sum_{i=1}^{n}[x_i]_k = \sum_{i=1}^{n} x_i + \mathrm{Lap}(\frac{1}{\epsilon})$$

The analysers are all semi-honest, next we will see how to deal with actively adversarial servers

Upto K-1 Corrupt Servers



Is sketching still secure ?

Sliding Attack On Honest Client

Adds +1 at some index and subtracts -1

$[x]_1 \in \mathbb{Z}_q^M$ $r_1, \ldots, r_M$

$z_1 = \sum_{j=1}^{M} r_i[x_j]_1$ $z_1^* = \sum_{j=1}^{M} r_i^2[x_j]_1$

z_3

$[x]_3 \in \mathbb{Z}_q^M$ $r_1, \ldots, r_M$

z_2 z_2^* z_3^*

$[x]_2 \in \mathbb{Z}_q^M$ $r_1, \ldots, r_M$

Check: $z^2 - z^* = 0$

$x =$ 0 1 0 0

Leaks 1 bit of information.

Malicious Sketching

$[\kappa x]_1, [x]_1 \in \mathbb{Z}_q^M$

Only the honest client knows κ

Servers now also broadcast

$$z_k^{**} = \sum_{i=1}^{M} r_i([\kappa x_i]_k)$$

$[\kappa x]_3, [x]_3 \in \mathbb{Z}_q^M$

Check: $(z^2 - z^*) + (\kappa z - z^{**}) = 0$

$[\kappa x]_2, [x]_2 \in \mathbb{Z}_q^M$

Show that the protocol is zero knowledge and a dishonest server does not learn any new information

Boneh et al., 'Lightweight Techniques for Private Heavy Hitters'.

The secrecy of κ prevents a sliding attack.
We are abstracting details of implementation: In reality the client also has to supply beaver triples or Shares of κ

Collusions Break Sketching Protocols

$[\kappa x]_1, [x]_1 \in \mathbb{Z}_q^M$

A dishonest client can just tell colluding servers the value for κ

$[\kappa x]_3, [x]_3 \in \mathbb{Z}_q^M$

Check: $(z^2 - z^*) + (\kappa z - z^{**}) = 0$

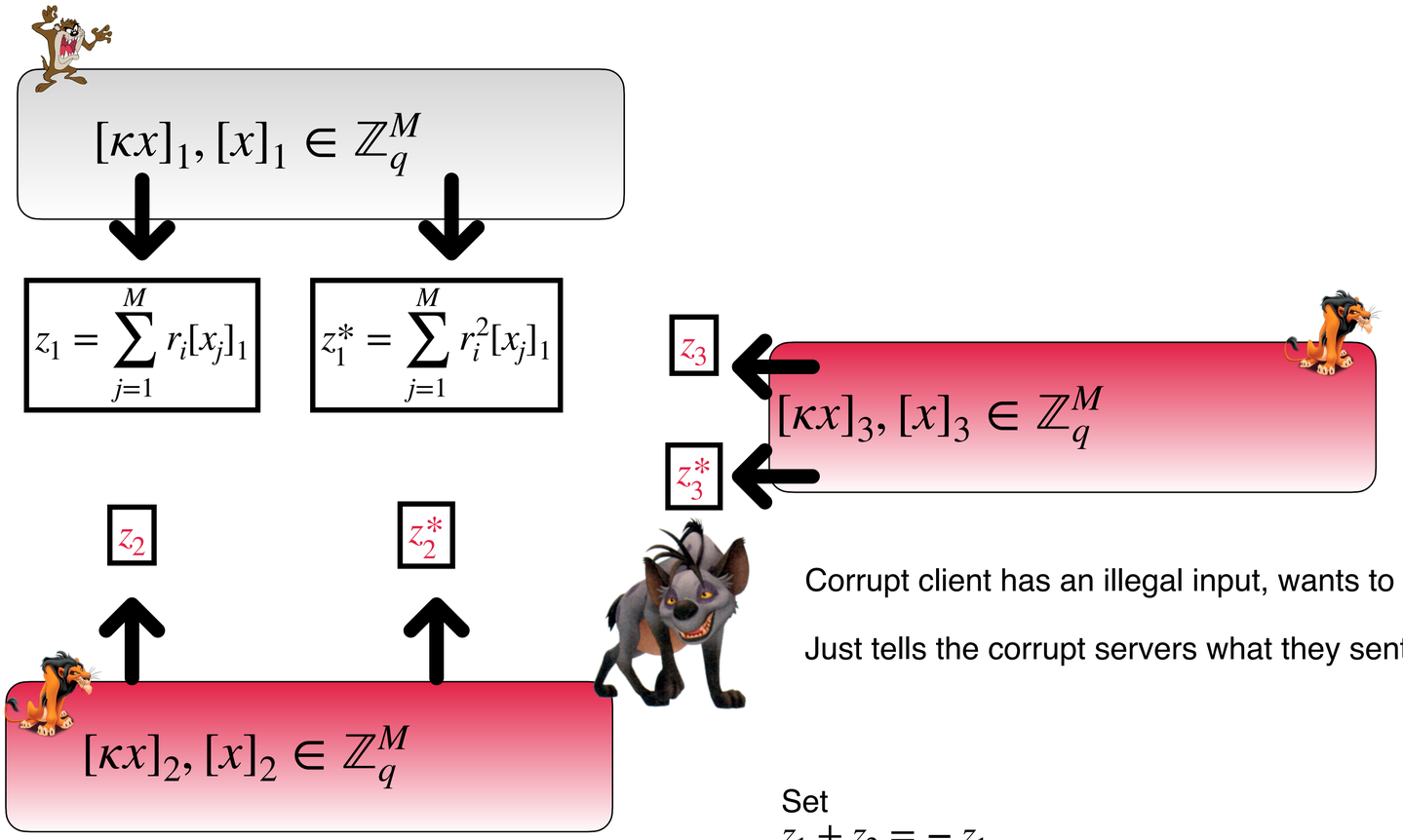
10	0	-2	0

Corrupt client has an illegal input, wants to be included in the protocol.

Just tells the corrupt servers what they sent to honest server 1

$[\kappa x]_2, [x]_2 \in \mathbb{Z}_q^M$

Collusions Break Sketching Protocols



A dishonest client can just tell colluding servers the value for κ

Corrupt client has an illegal input, wants to be included in the protocol.

Just tells the corrupt servers what they sent to honest server 1

Set

$z_1 + z_2 = -z_1$

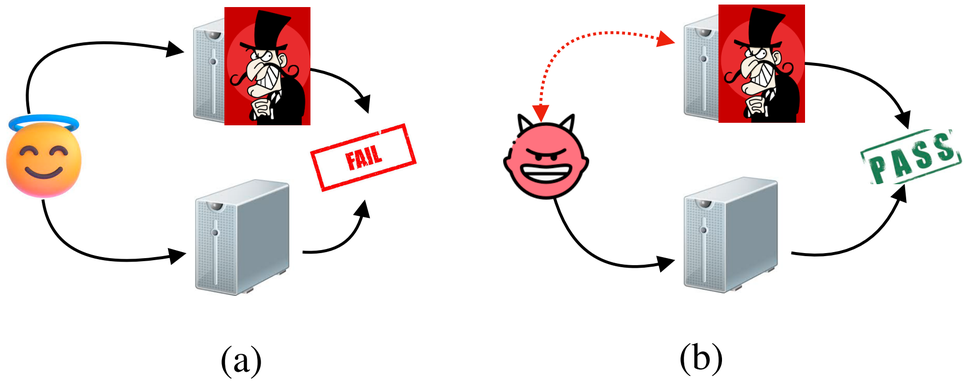
$z_1^* + z_2^* = -z_1^*$

$z_1^{**} + z_2^{**} = -z_1^{**}$

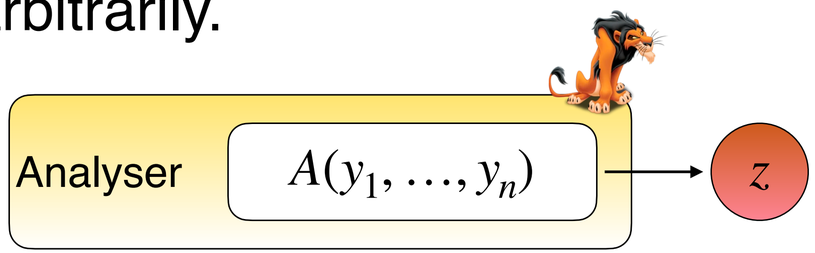
Check: $(z^2 - z^*) + (\kappa z - z^{**}) = 0$

Summary Of Malicious Activities

- Manipulate an honest clients input and exclude them from the protocol.
- Include a malicious input into the protocol.



- Manipulate the noise generation protocol to bias the output of the protocol arbitrarily.



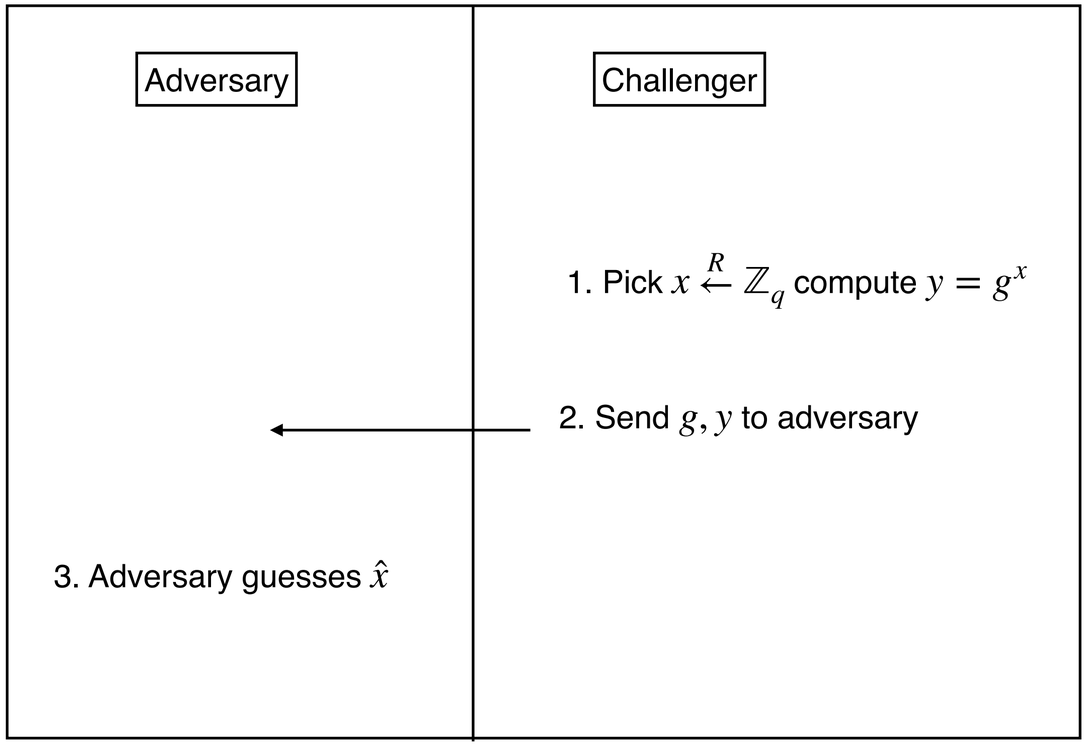
We want to keep all the nice properties of the protocols discussed so far but be able to thwart these attacks

ACT III: Accountability + Privacy

Key observation: If we ignore the noise generation procedure, histograms and the sketching algorithms are a linear function of client inputs.

Key assumption: The clients/servers have negligible advantage in solving the discrete log problem

Discrete Log Attack Game



$\text{Advantage}(\mathcal{A}, \mathbb{G}_q) := Pr[\hat{x} = x]$

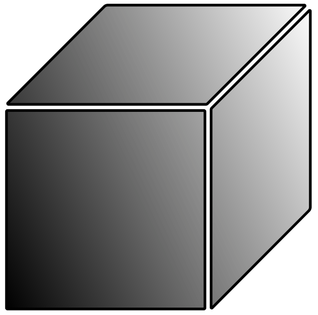
We do not know any PPT algorithm that has non negligible advantage in guessing x for large enough q.

Let $\mathbb{G}_q$ be a sub group of $\mathbb{Z}_p^*$ with order q where p and q are large primes such that $q \mid p - 1$ and g is a generator of $\mathbb{G}_q$

Pedersen Commitments

- Let $\mathbb{G}_q$ be a sub group of $\mathbb{Z}_p^*$ with order q where p and q are large primes such that $q \mid p - 1$

- **COMMIT:** We have a secret $s \in \mathbb{Z}_q$ and we want to commit to it. Then a Peterson commitment to s is given by $c = Com(s, t) = g^s h^t$ where $t \overset{R}{\leftarrow} \mathbb{Z}_q$ and g is randomly selected generators for $\mathbb{G}_q$ and $h = g^\alpha$ for some random $\alpha \overset{R}{\leftarrow} Z_q$.



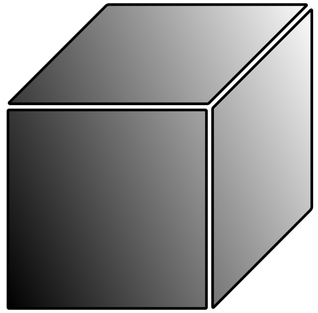
c: Message in a locked box.
Only way to open it is to have the key

s, t such that $\mathtt{Com}(s, t) = c$

Pedersen Commitments

- Given c, a computationally unbounded adversary $\mathscr{A}$ cannot infer any information about x **(Perfectly Hiding)**

$$c = Com(x, r)$$



$$\Pr[\hat{x} = 0 \mid c] = \Pr[\hat{x} = 1 \mid c]$$

- Given c, if adversary $\mathscr{A}$ can find $(s', t') \neq (s, t)$ such that $Com(s, t) = Com(s', t')$, then $\mathscr{A}$ can solve the DLOG attack game **(Computationally Binding)**

Hard to fake the key to the box.

Homomorphism

- Given $c_1 = Com(s_1, t_1)$ and $c_2 = Com(s_2, t_2)$, then
- $c_1 c_2 = Com(s_1 + s_2, t_1 + t_2)$

Addition in plaintext space is multiplication in ciphertext space.

- Important trick we will use a lot:
  - Let $a, b, c \in Z_q$ and $r_a, r_b, r_c \overset{R}{\leftarrow} Z_q^3$. Let c_a, c_b, c_c be the respective commitments.
  - Someone claims $x = a + b + c \pmod q$ and $y = r_a + r_b + r_c \pmod q$.
- Then $g^x h^y = c_a c_b c_c$

Informal defintion

Publicly Verifiable Covert Security

Participants in the protocol may deviate arbitrarily from the prescribed instructions.

This deviation might violate an honest party's privacy.

However, this violation is detected by an honest party with a constant probability $\rho > \frac{1}{2}$, and the honest party can prove who cheated.

In our case, $\rho = 1 - \mathtt{Adv}(DLog) \approx 1$, so the honest party will detect and abort the protocol in case of any violations.

Protocol Overview: Previously

Voting:

Clients send input shares to servers

Verifying:

Servers collaboratively verify that the input is legal

Aggregating:

$$\sum_{i=1}^{n} \sum_{k=1}^{K} [x_i]_k = \sum_{i=1}^{n} x_i$$

Protocol Overview: Now

Committed Voting:

Clients send input shares to servers

Active Verifying:

Servers collaboratively verify that the input is legal

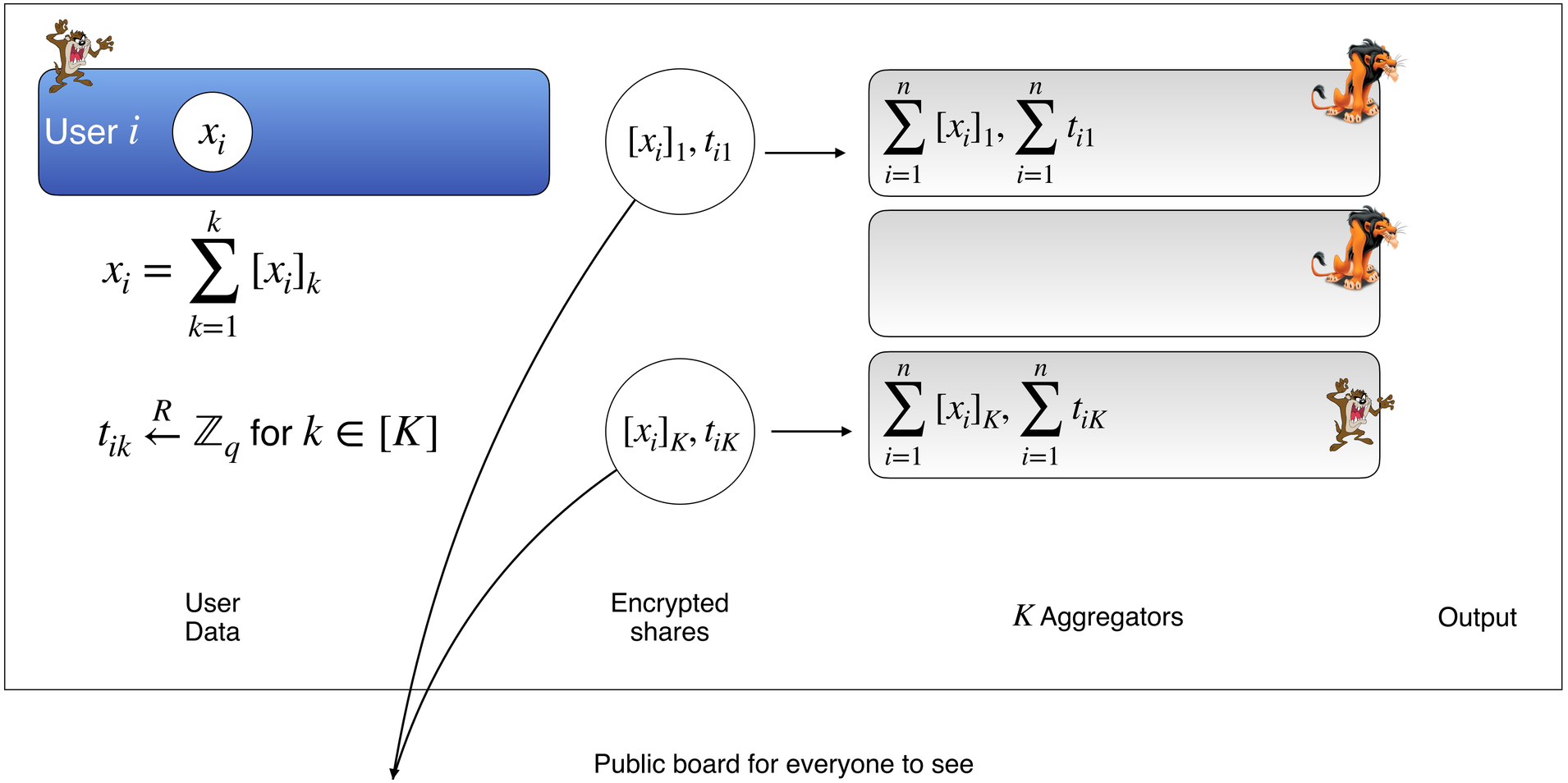
Noise Generation

Servers collaboratively verify that the output is differentially private.

Verifiable Aggregating

$$\sum_{i=1}^{n}\sum_{k=1}^{K}[x_i]_k = \sum_{i=1}^{n} x_i$$

Committed Voting



Defending Against Malicious Clients And Servers

$[x]_1 \in \mathbb{Z}_q^M$ $\quad r_1, \ldots, r_M$

$t_1 \in \mathbb{Z}_q^M$

1. Server 1 samples $r_1, \ldots, r_M$ where $r_i \xleftarrow{R} \mathbb{Z}_q$ independently and broadcasts it to other servers

$[x]_3 \in \mathbb{Z}_q^M$

$t_3 \in \mathbb{Z}_q^M$ $\quad r_1, \ldots, r_M$

INTUITION:

We will create a degree two polynomial $p(\vec{r})$ such that if $\vec{r}$ is not a root, then the only way to 0 out this polynomial is to have a single non zero entry equal to 1.

The cheating client does not know the values of $\vec{r}$ thus with probability $\frac{2}{q}$ fails to pass to the test

$[x]_2 \in \mathbb{Z}_q^M$ $\quad r_1, \ldots, r_M$

$t_2 \in \mathbb{Z}_q^M$

Total of KM public commitments

$\alpha_{ij} = Com([x_j]_i, t_{ij})$ where $i \in [K]$ and $j \in [M]$

Defending Against Malicious Clients And Servers

$[x]_1 \in \mathbb{Z}_q^M$ $\quad r_1, \ldots, r_M$

$t_1 \in \mathbb{Z}_q^M$

$z_1 = \sum_{j=1}^{M} r_i [x_j]_1$ $\quad z_1^* = \sum_{j=1}^{M} r_i^2 [x_j]_1$ $\quad u_1 = \sum_{j=1}^{M} r_i t_{1j}$ $\quad w_1 = \sum_{j=1}^{M} r_i^2 t_{1j}$

New messages

z_3

z_3^*

u_3

w_3

$[x]_3 \in \mathbb{Z}_q^M$ $\quad r_1, \ldots, r_M$

$t_3 \in \mathbb{Z}_q^M$

z_2 $\quad z_2^*$ $\quad u_2$ $\quad w_2$

$[x]_2 \in \mathbb{Z}_q^M$ $\quad r_1, \ldots, r_M$

$t_2 \in \mathbb{Z}_q^M$

$\alpha_{ij} = Com([x_j]_i, t_{ij})$ where $i \in [K]$ and $j \in [M]$

Defending Against Malicious Clients

$[x]_1 \in \mathbb{Z}_q^M$ $r_1, \ldots, r_M$

$t_1 \in \mathbb{Z}_q^M$

$z_1 = \sum_{j=1}^{M} r_i [x_j]_1$ $z_1^* = \sum_{j=1}^{M} r_i^2 [x_j]_1$ $u_1 = \sum_{j=1}^{M} r_i t_{1j}$ $w_1 = \sum_{j=1}^{M} r_i^2 t_{1j}$

BROADCAST

z_3 z_3^* u_3 w_3

z_2 z_2^* u_2 w_2

$[x]_2 \in \mathbb{Z}_q^M$ $r_1, \ldots, r_M$

$t_2 \in \mathbb{Z}_q^M$

$[x]_3 \in \mathbb{Z}_q^M$

$t_3 \in \mathbb{Z}_q^M$ $r_1, \ldots, r_M$

$\alpha_{ij} = Com([x_j]_i, t_{ij})$ where $i \in [K]$ and $j \in [M]$

Defending Against Malicious Clients And Servers

Each server receives from Server k : z_k, z_k^*, u_k, w_k

$$\prod_{i=1}^{M} \left(\alpha_{ik}\right)^{r_i} = g^{z_k} h^{u_k}$$

$$\prod_{i=1}^{M} \left(\alpha_{ik}\right)^{r_i^2} = g^{z_k^*} h^{w_k}$$

If this test fails — Server aborts the protocol and declares that the protocol has been tampered with

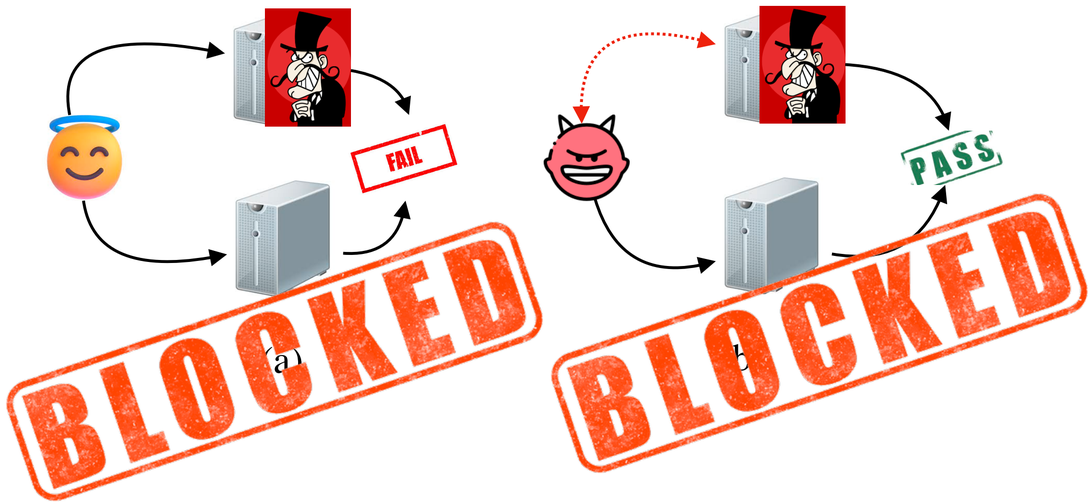
Check if $z^2 - z^* = 0$ where $z = \sum_{k=1}^{K} z_k$ and $z^* = \sum_{k=1}^{K} z_k^*$

If this test passes — Servers have not cheated so far. Next, check if client input is well formed

Theorem

Let $\overrightarrow{v}$ represent an honest client's input and f_{verify} denote the deterministic function that checks if $\overrightarrow{v} \in V$. Let π_{sketch} refer to the protocol described earlier

Then π_{sketch} securely computes f with abort in the presence of static covert adversary with ρ-deterrent, where $\rho = 1 - \texttt{Adv}_{\mathcal{A}}(\text{DLog}) \approx 1$.



Proof Sketch

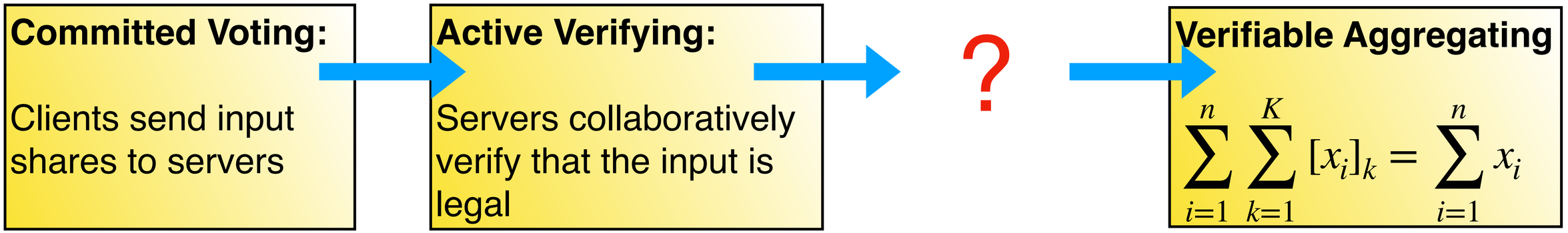
The issue with the original sketching proof was that the server could tamper with user inputs, and therefore infer information about users inputs.

1. So if we could just detect if they have tampered with the inputs we would be good. Furthermore, we don’t want lying clients saying their inputs were tampered with when in fact they were not.

2. By committing to their inputs, the clients cannot later change their mind about their inputs.

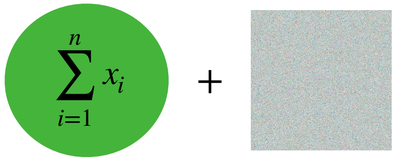
3. The homomorphic property of our commitment scheme ensures that when the servers broadcast their linear sketch output, the only way to get the authentication test to align — is to break the binding property of the commitment scheme.

Verifiable Noise Generation



The only inputs included are honest inputs. Note if we did not need DP. We are done! We can use the commitments to verify that the servers aggregate their shares correctly.

Verifiable But Private Noise



The actual random value of the noise should remain private to everyone.

Challenge:

As the total noise is secret, any server could just use adversarial noise and the other servers have no way of verifying what the other malicious server did.

Desired Outcome

So we must design a protocol that still preserves the secrecy of noise while ensuring that any party that adds malicious noise gets caught

Binomial Mechanism

The binomial distribution is (ϵ, δ, k) smooth

Ghazi, B., Golowich, N., Kumar, R., Pagh, R. and Velingker, A., 2021, October. On the power of multiple anonymous messages: Frequency estimation and selection in the shuffle model of differential privacy.

As a direct consequence of the results of Ghazi *et al.*

Fix $\eta > 30$ and $0 \leq \delta \leq o(\frac{1}{\eta})$, then, the additive mechanism using $\texttt{Binomial}(\eta, \frac{1}{2})$ results in (ϵ, δ)-differentially private M-bin histograms where $\epsilon = 10\sqrt{\frac{1}{\eta} \ln \frac{2}{\delta}}$.

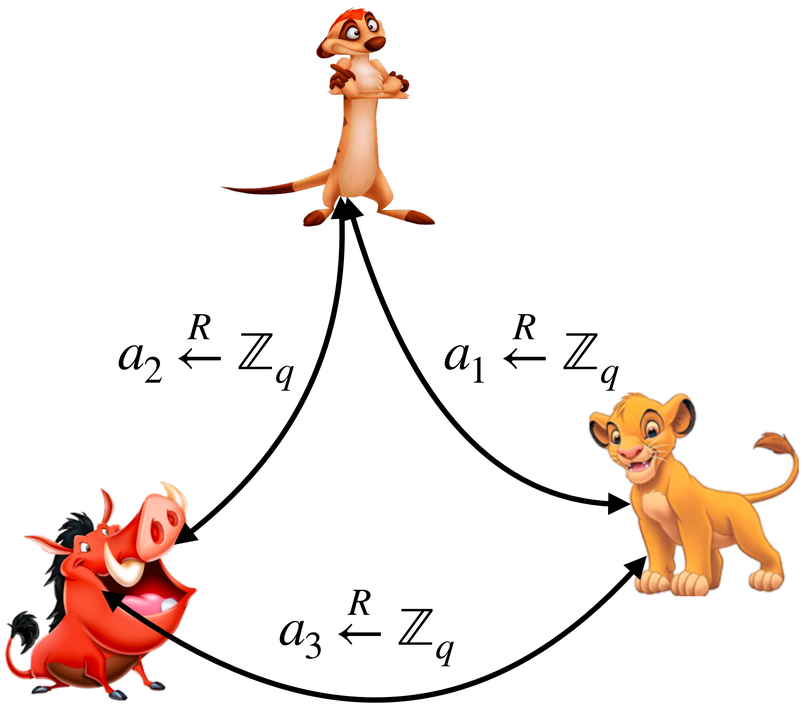
Thus if we could guarantee that the servers added $\texttt{Binomial}(\eta, \frac{1}{2})$ noise to the output of the protocol we would get DP.

A Well Known Fact

- Given a bit $b_i \in \{0,1\}$ sampled from any arbitrary $\texttt{Bernoulli(p)}$
- And given $c_i \sim \texttt{Bernoulli}(1/2)$
- $z_i = c_i \oplus b_i$ is guaranteed to be $z_i \sim \texttt{Bernoulli}(1/2)$

IDEA: If we could generate verifiable public randomness, then we could ensure that regardless of what the an actor does, the final distribution is DP suitable.

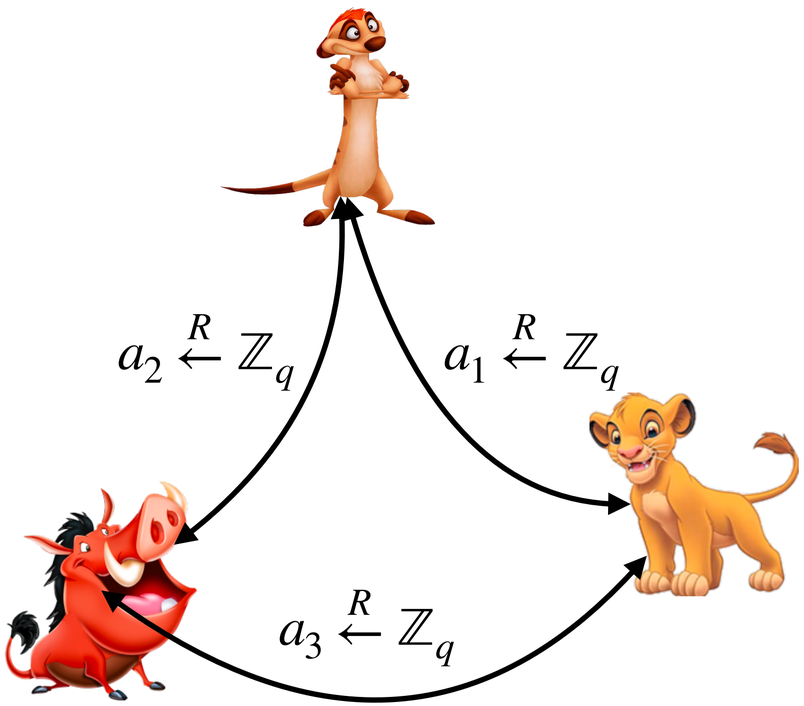
Ideal Morra



- Each party k **simultaneously** samples a value from $a_k \xleftarrow{R} \mathbb{Z}_q$ uniformly at random and then broadcast their sampled values to each other.
- **As long as a single party does this sampling honestly,** the sum of their values $(\sum_{k=1}^{K} a_k) \mod q$ is also a uniformly random value sampled from $\mathbb{Z}_q$

We now have a way to generate public randomness even in presence of malicious actors.

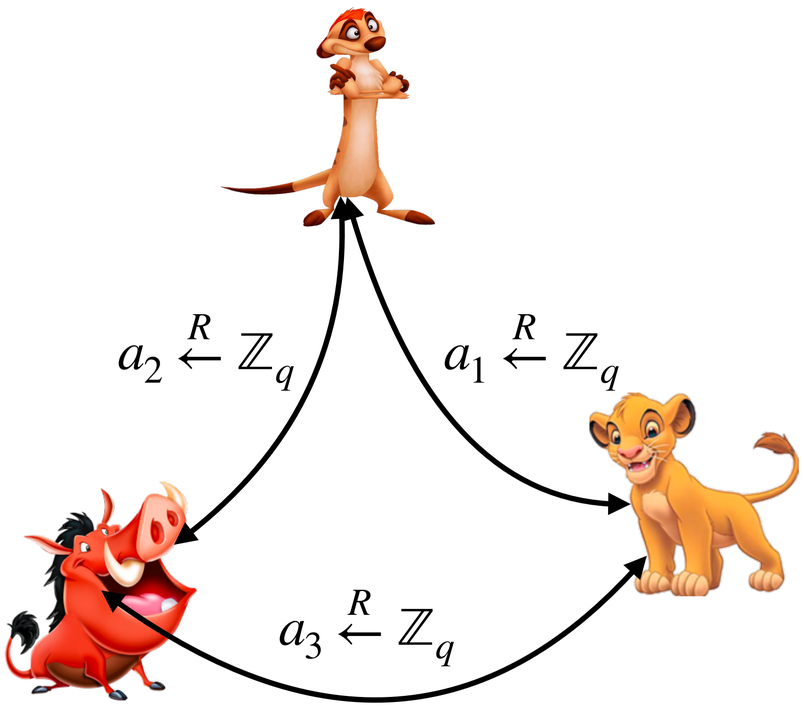
Ideal Morra



- Each party k **simultaneously** samples a value from $a_k \xleftarrow{R} \mathbb{Z}_q$ uniformly at random and then broadcast their sampled values to each other.
- **As long as a single party does this sampling honestly,** the sum of their values $(\sum_{k=1}^{K} a_k) \mod q$ is also a uniformly random value sampled from $\mathbb{Z}_q$

There is no practical way to guarantee that they will all sample these values at exactly the same time. If a malicious party sees the output of an honest party, they can adjust their value.

Real Morra: First Commit Then Reveal



$Com([a_k, t_k)$ for all $k \in [k]$

- Each party k samples a value from $a_k \xleftarrow{R} \mathbb{Z}_q$ uniformly at random.
- They then broadcast a commitment of this value to all other parties.
- Once all commitments have been revealed, the parties reveal their random values in any order.
- **With each reveal, each party checks whether the released values open to the same commitment.**

Formal simulator proof of security can be found in

Blum M., 1983, Coin Flipping Over A Telephone A Protocol For Solving Impossible Problems

Distributed Noise Generation Protocol

Server 1

$$\sum_{i=1}^{n} [x_i]_1 \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{i1} \in \mathbb{Z}_q^M$$

$$b_1 \xleftarrow{R} V$$

....

$$b_\eta \xleftarrow{R} V$$

Server 2

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

- The generating server samples η valid votes uniformly randomly.

Server K

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Distributed Noise Generation Protocol

Server 1

$$\sum_{i=1}^{n} [x_i]_1 \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{i1} \in \mathbb{Z}_q^M$$

$[b_1]_1, \ldots, [b_\eta]_1$

$[b_1]_2, \ldots, [b_\eta]_2$

Server 2

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Shares are guaranteed to be well formed!

- Samples η valid votes uniformly randomly.
- Secret share them using verifiable committed secret sharing.

$[b_1]_K, \ldots, [b_\eta]_K$

Server K

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Distributed Noise Generation Protocol

Server 1

$$\sum_{i=1}^{n} [x_i]_1 \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{i1} \in \mathbb{Z}_q^M$$

$[b_1]_1, \ldots, [b_\eta]_1$

$[b_1]_2, \ldots, [b_\eta]_2$

Server 2

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

All of them play Morra to Generate public randomness

$c_1, \ldots, c_\eta$ where

$c_{ij} \sim \texttt{Bernoulli}(1/2)$
for all $i \in [\eta]$ and $j \in [M]$

$[b_1]_K, \ldots, [b_\eta]_K$

Server K

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Distributed Noise Generation Protocol

Server 1

$$\sum_{i=1}^{n} [x_i]_1 \in \mathbb{Z}_q^M$$
$$\sum_{i=1}^{n} t_{i1} \in \mathbb{Z}_q^M$$

$[b_1]_1, \ldots, [b_\eta]_1$

$[b_1]_2, \ldots, [b_\eta]_2$

Server 2

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$
$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

$c_1, \ldots, c_\eta$

We know for certain that $b_{ij} \in \{0,1\}$ for all $i \in [\eta]$ and $j \in [M]$:

The boolean circuit $z_{ij} = b_{ij} \oplus c_{ij}$ is

$z_{ij} = b_{ij}$ if $c_{ij} = 0$
$z_{ij} = 1 - b_{ij}$ if $c_{ij} = 1$

$[b_1]_K, \ldots, [b_\eta]_K$

Server K

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$
$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Distributed Noise Generation Protocol

Server 1

$$\sum_{i=1}^{n} [x_i]_1 \in \mathbb{Z}_q^M$$
$$\sum_{i=1}^{n} t_{i1} \in \mathbb{Z}_q^M$$

$[b_1]_1, \ldots, [b_\eta]_1$

Server 2

$[b_1]_2, \ldots, [b_\eta]_2$

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$
$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

$c_1, \ldots, c_\eta$

We know for certain that $b_{ij} \in \{0,1\}$ for all $i \in [\eta]$ and $j \in [M]$:

The boolean circuit $z_{ij} = b_{ij} \oplus c_{ij}$ is

$z_{ij} = b_{ij}$ if $c_{ij} = 0$
$z_{ij} = 1 - b_{ij}$ if $c_{ij} = 1$

c_{ij} is public! Thus each server can adjust their shares accordingly

Server K

$[b_1]_K, \ldots, [b_\eta]_K$

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$
$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Distributed Noise Generation Protocol

Server 1

$$\sum_{i=1}^{n} [x_i]_1 \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{i1} \in \mathbb{Z}_q^M$$

$[b_1]_1, \ldots, [b_\eta]_1$

$[b_1]_2, \ldots, [b_\eta]_2$

Server 2

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

$c_1, \ldots, c_\eta$

Operations for Server k

$z_{ij} = b_{ij}$ if $c_{ij} = 0$

$[z_{ij}]_k = 1 - [b_{ij}]_k$ if $c_{ij} = 1$

But HANG ON! If I change the shares, the commitments won't align

$[b_1]_K, \ldots, [b_\eta]_K$

Server K

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Distributed Noise Generation Protocol

Server 1

$$\sum_{i=1}^{n} [x_i]_1 \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{i1} \in \mathbb{Z}_q^M$$

$[b_1]_1, \ldots, [b_\eta]_1$

$[b_1]_2, \ldots, [b_\eta]_2$

Server 2

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Operations for Server k

$z_{ij} = b_{ij}$ if $c_{ij} = 0$

$[z_{ij}]_k = 1 - [b_{ij}]_k$ if $c_{ij} = 1$

Let $\alpha = Com(x, r)$, then $\alpha^{-1} \cdot (gh) = Com(1 - x, 1 - r)$

But HANG ON! If I change the shares, the commitments won't align

Updates are linear — So commitments can be adjusted accordingly in ciphertext space.

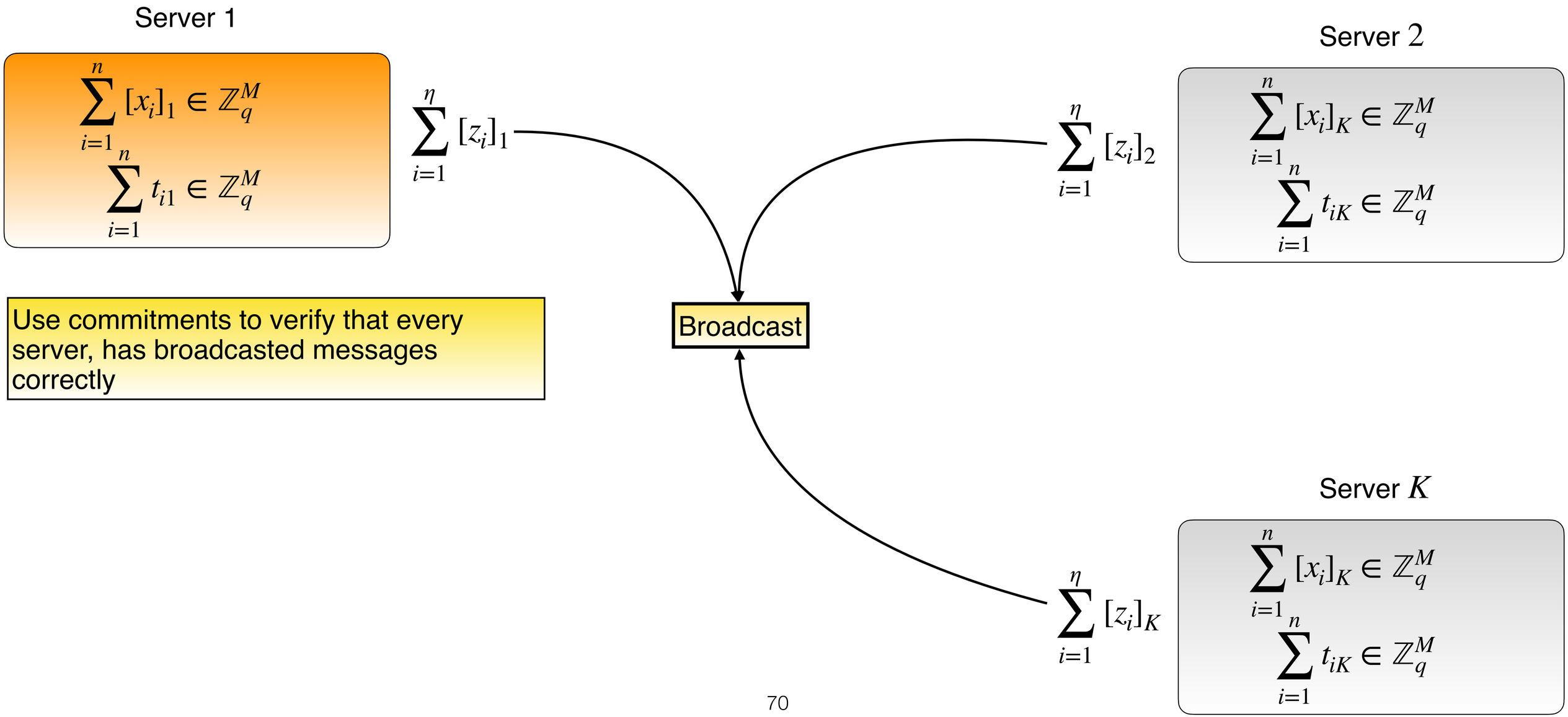
$[b_1]_K, \ldots, [b_\eta]_K$

Server K

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Servers Broadcast Their shares



Aggregate

Server 1

$$\sum_{i=1}^{n} [x_i]_1 \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{i1} \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{\eta} [z_i]_1$$

$$\sum_{i=1}^{\eta} [z_i]_2$$

Server 2

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Each coordinate of $Y_1 \sim \texttt{Binomial}(\eta, \frac{1}{2})$

$$Y_1 = (\sum_{i=1}^{\eta} \sum_{k=1}^{k} [z_i]_k)$$

Attention

The generating server is the only server that knows $b_1, \ldots, b_\eta$. All other servers operate on information theoretically private shares.

But the generating server still knows the value for randomness — so the protocol is not private.

$$\sum_{i=1}^{\eta} [z_i]_K$$

Server K

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Each Server Takes A Turn To Be Generating Server

Server k

$$\sum_{i=1}^{n} [x_i]_1 \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{i1} \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{\eta} [z_i]_k$$

$$Y_k = \left(\sum_{i=1}^{\eta} \sum_{k=1}^{k} [z_i]_k\right)$$

Server 1

$$\sum_{i=1}^{\eta} [z_i]_1$$

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

Each server has a go at being the generating server, before broadcasting their shares.
As 1 server is guaranteed to be semi-honest, the final sum of all the noise remains private

Total noise: $Y = Y_1 + \ldots + Y_K$

Variance: $\frac{k\eta}{4}$ instead of $\frac{\eta}{4}$ but k is a small number, so can be treated as a constant.

Server K

$$\sum_{i=1}^{\eta} [z_i]_K$$

$$\sum_{i=1}^{n} [x_i]_K \in \mathbb{Z}_q^M$$

$$\sum_{i=1}^{n} t_{iK} \in \mathbb{Z}_q^M$$

In Summary: Verifiable DP Histograms

$$(Y_1 + \ldots + Y_K) + \sum_{i=1}^{n} \sum_{k=1}^{K} [x_i]_k = \sum_{i=1}^{n} x_i + Y$$

Noise and inputs are committed. Each broadcast is verifiable.

As long as 1 server is semi-honest, no server knows the value of Y

Guaranteed to have k copies of $\texttt{Binomial}(\eta, 0.5)$

Conclusion/Open Questions

We have a protocol by which we get verified histogram protocols.

Open questions:

- Can we make variance independent of the number of servers K

- Committments worked out because the protocol and the noise generation was linear, are there efficient methods for non-linear protocols.

Information-Theoretic Privacy

If we could guarantee that number of bad servers was strictly less than $\frac{K}{3}$,

then via robust secret sharing, using Reed-Solomon Codes, we could get rid of the need for commitments.