Multi-user Verifiable/Secure Computation

Ko Dokmai, Saba Eskandarian

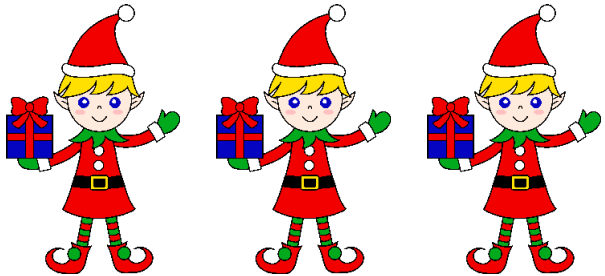
Goals

Extend cryptographic protocols with applications to cloud computing from a two-party to a multi-party setting:

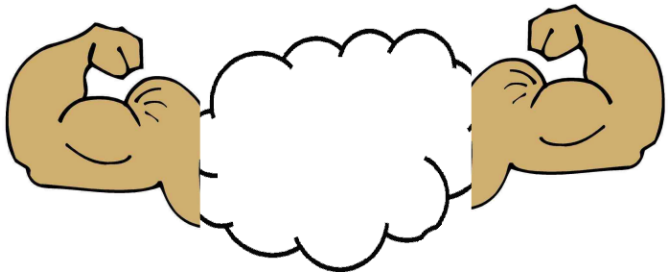
- Verifiable Computation
- Secure Computation

Common Setting

Weak Clients



Powerful Cloud

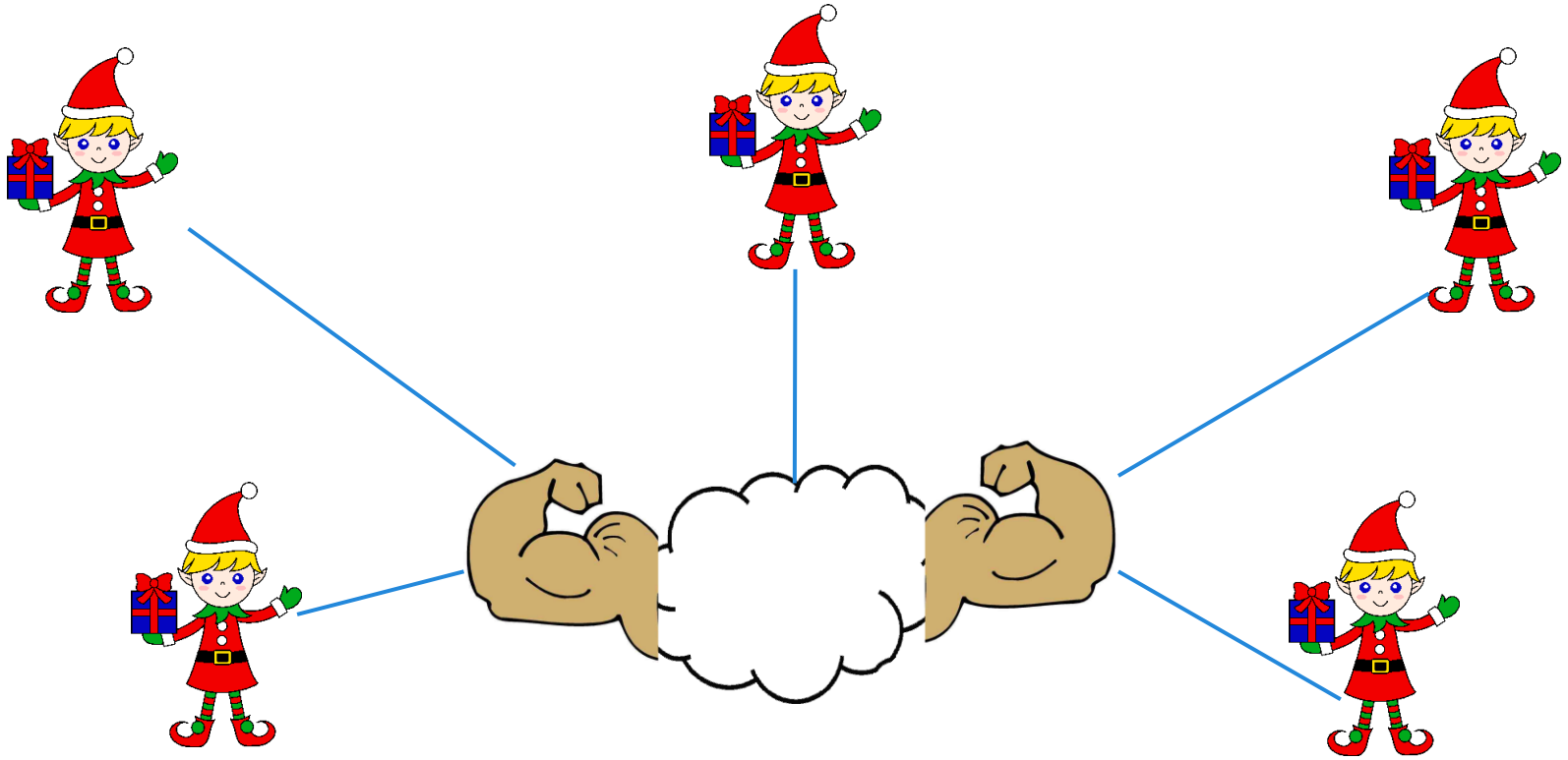


Common Elements

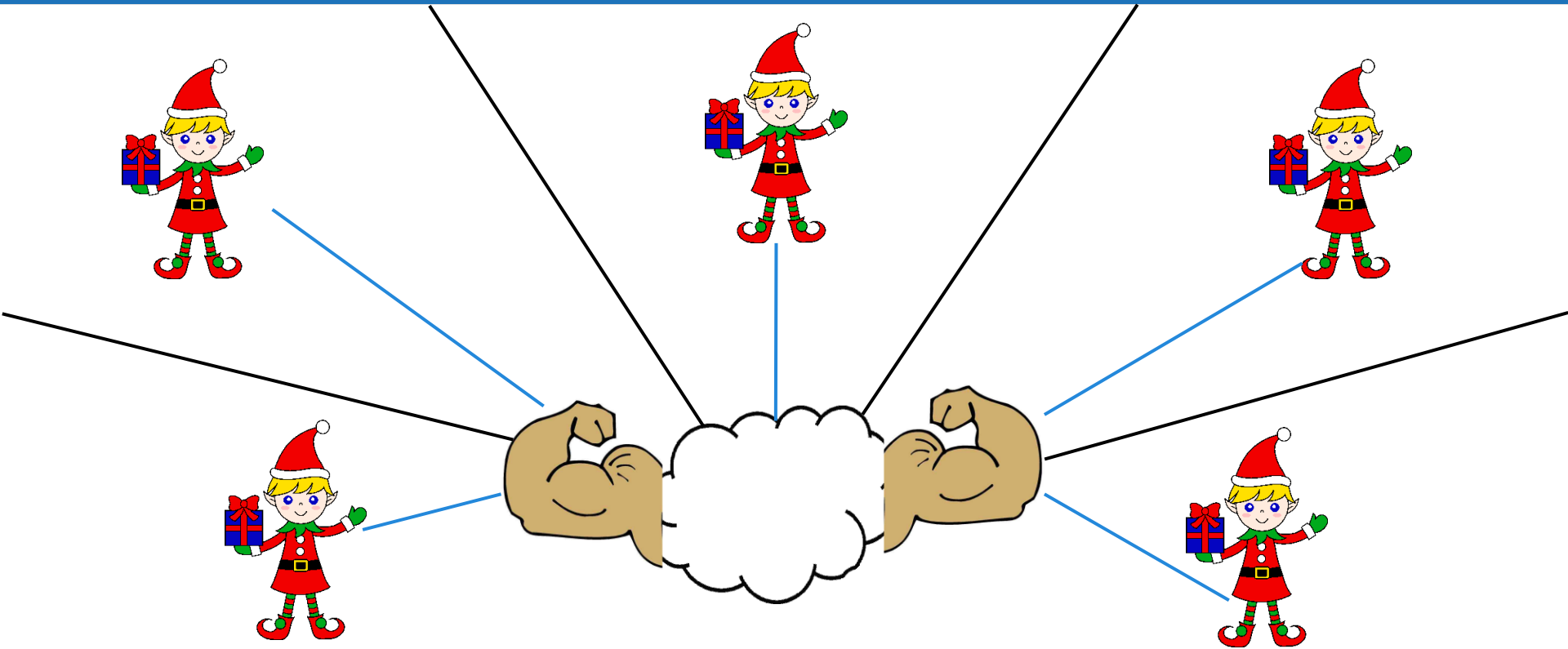
- Multiclient setting
- Outsourced computation
- Best definitions = VBB Obfuscation

Multi-user Verifiable Computation

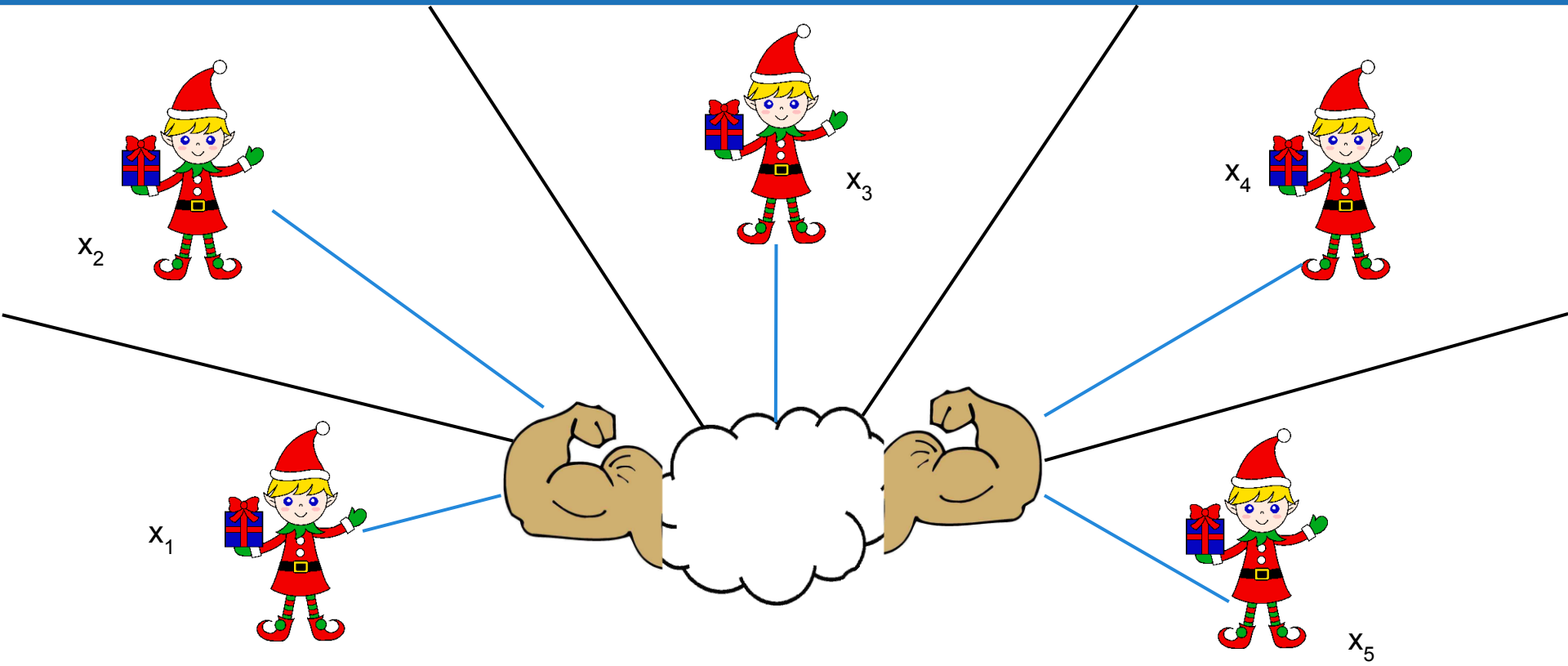
Setting



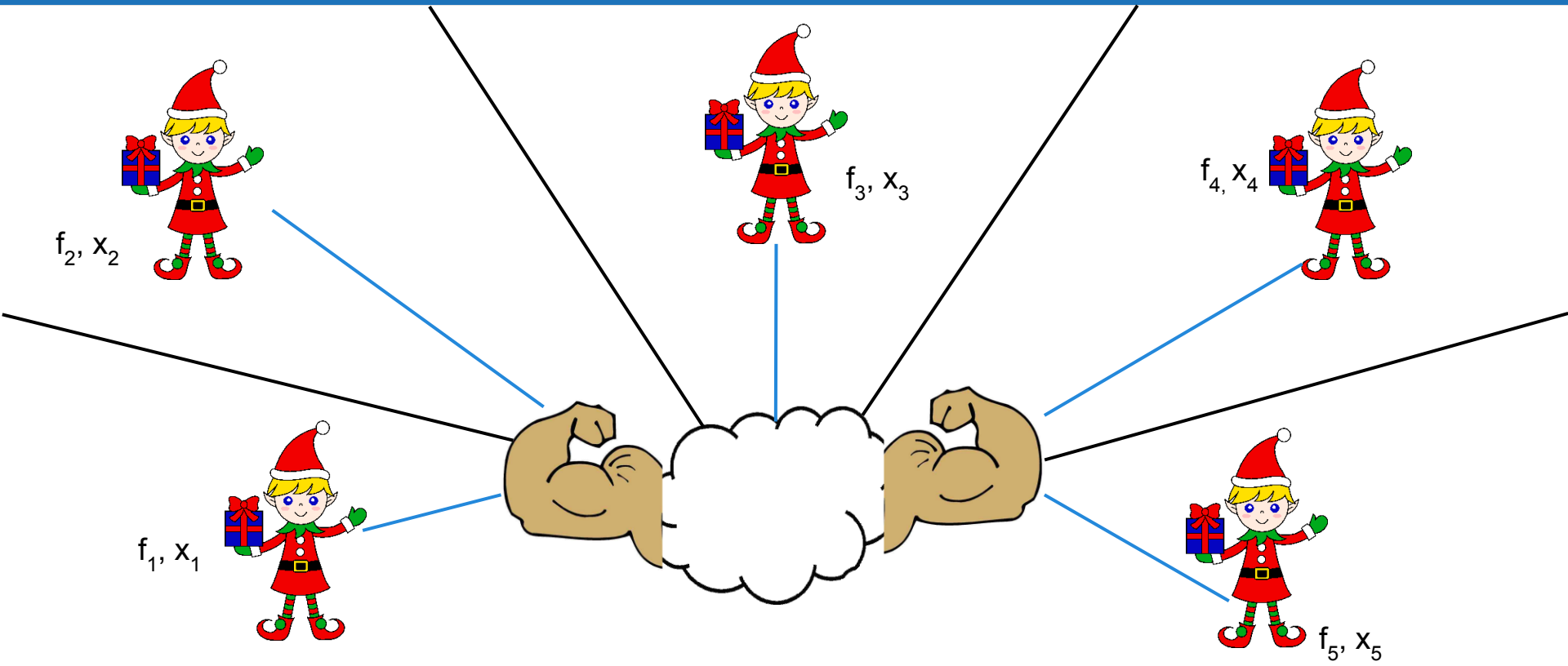
Setting



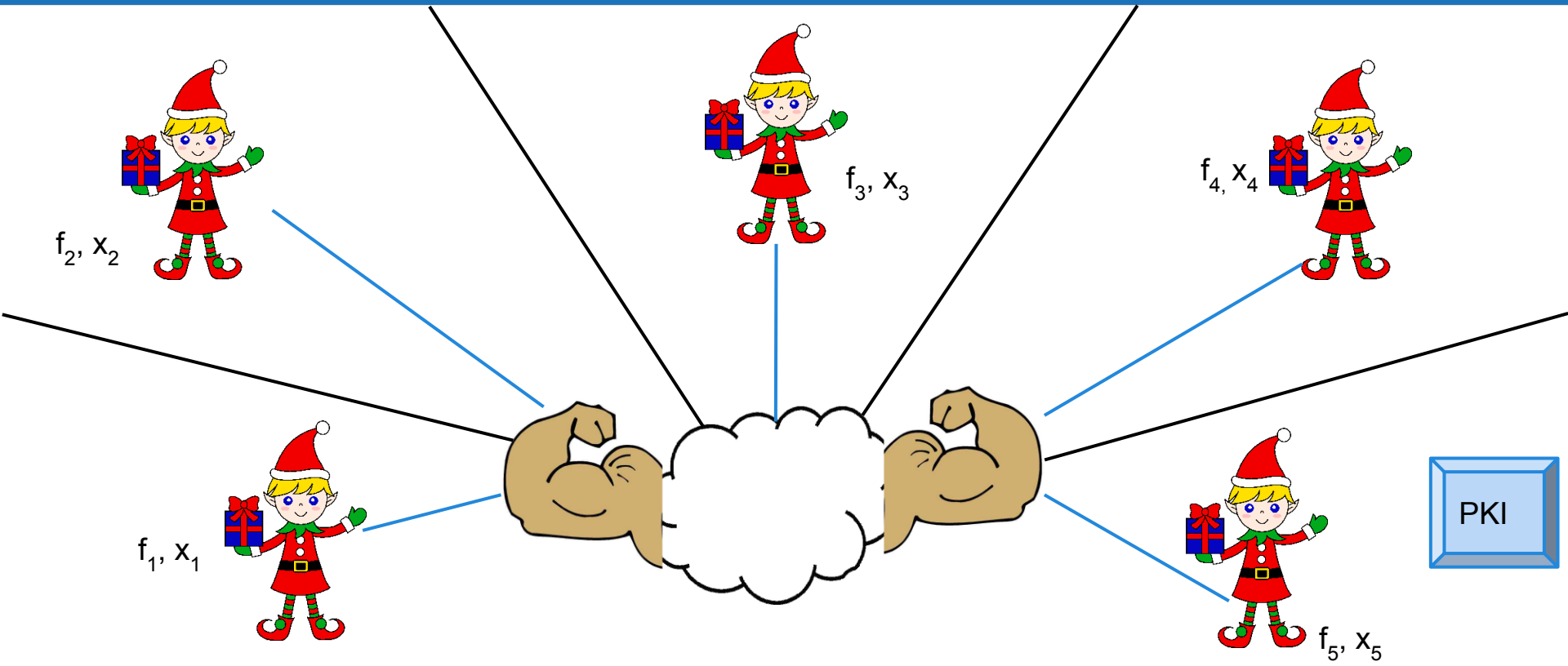
Setting



Setting



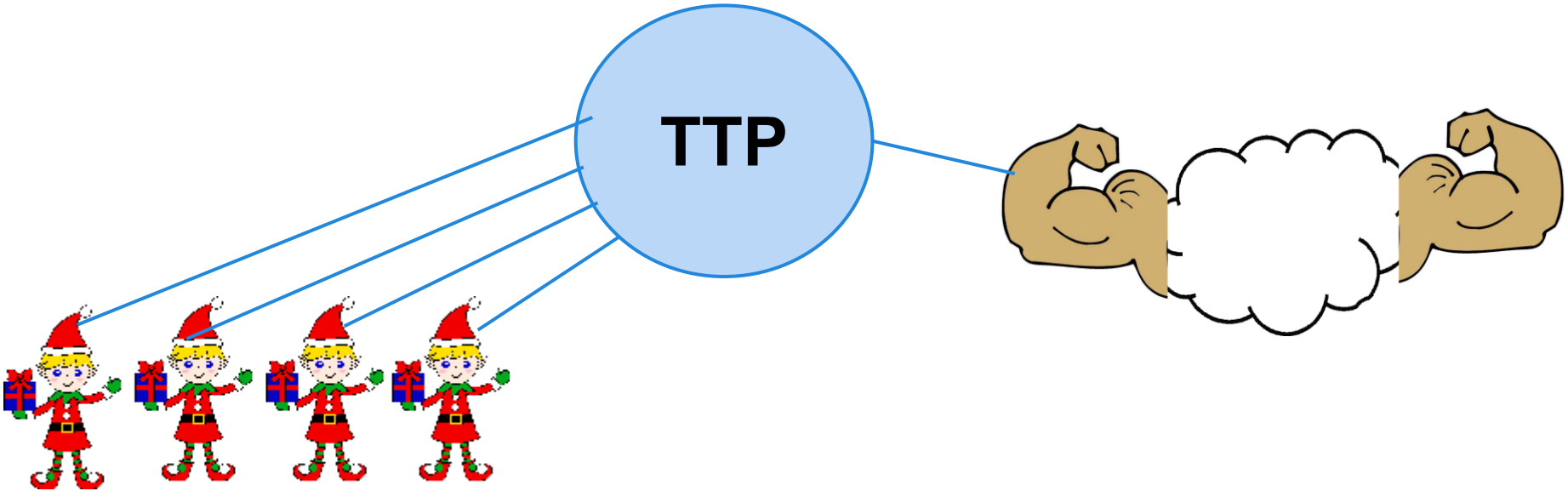
Setting



Security

- Real/Ideal paradigm
- Most desirable definition is unattainable (implies VBB)
- Ideal definition behavior differs based on who is corrupted
- we will assume semi-honest cloud and clients

Ideal Situation



Primitives Used

- Proxy oblivious transfer (POT)
- Garbled Circuits
- Fully homomorphic encryption (FHE)

Garbled Circuits

- Garble(1^k, C) → pk, sk, Γ
- Genc(pk, x) → c
- Geval(Γ, c) → Y
- Gdec(sk, Y) → y

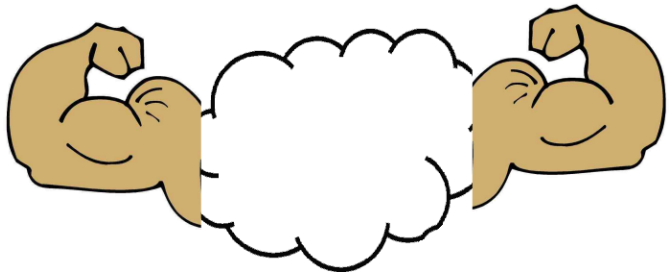
- y=C(x)
- Extra requirements

1st try: single-client, single-use

Intuition: P_1 Garbles function and input,
Server evaluates and sends back results.

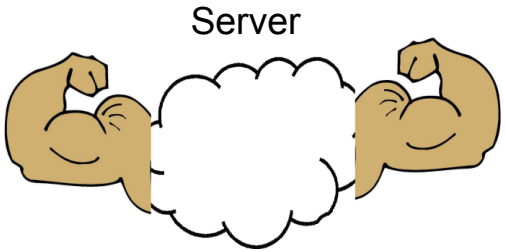
P_1

Server



1st try: single-client, single-use

f, x



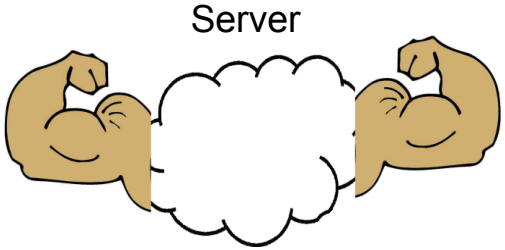
Garble(1^k, f) → pk, sk, Γ

Genc(pk, x) → c

Γ, c

1st try: single-client, single-use

f, x



Garble(1^k, f) → pk, sk, Γ

Genc(pk, x) → c

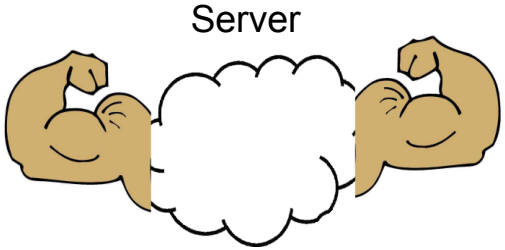
Γ, c →

Geval(Γ, c) → Y

← Y

1st try: single-client, single-use

f, x



Garble(1^k, f) → pk, sk, Γ

Genc(pk, x) → c

Γ, c →

Geval(Γ, c) → Y

← Y

Gdec(sk, Y) → y

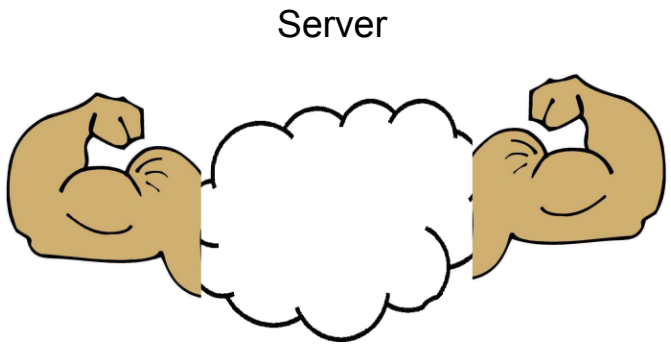
Problem: 1st try only works once.

How can we allow for multiple function evaluations?

Why not just keep sending more inputs?

2nd try: single-client, multi-use

Intuition: garbled inputs encrypted with FHE



Fully Homomorphic Encryption

- $Fgen(1^k) \rightarrow (pk_{FHE}, sk_{FHE})$
- $Fenc(pk_{FHE}, m) \rightarrow c$
- $Fdec(sk_{FHE}, c') \rightarrow m'$
- $Feval(pk_{FHE}, f, c_1,...,c_n)$

2nd try: single-client, multi-use

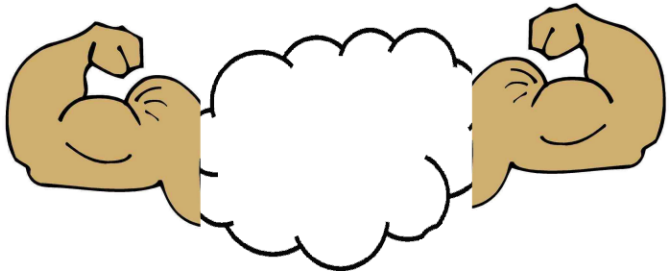
Setup Phase

P_1

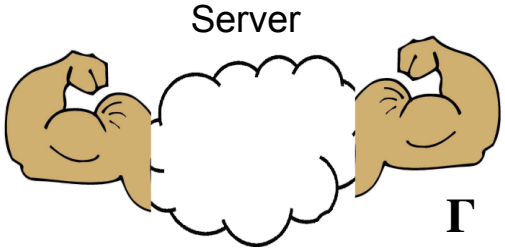
Garble(1^k, f) → pk, sk, Γ

Γ

Server



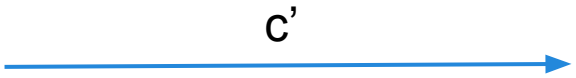
2nd try: single-client, multi-use



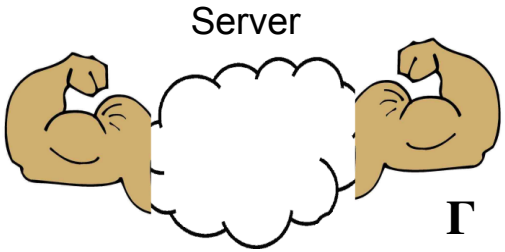
$Fgen(1^k) \rightarrow (pk_{FHE}, sk_{FHE})$

$Genc(pk, x) \rightarrow c$

$Fenc(pk_{FHE}, c) \rightarrow c'$



2nd try: single-client, multi-use



$Fgen(1^k) \rightarrow (pk_{FHE}, sk_{FHE})$

$Genc(pk, x) \rightarrow c$

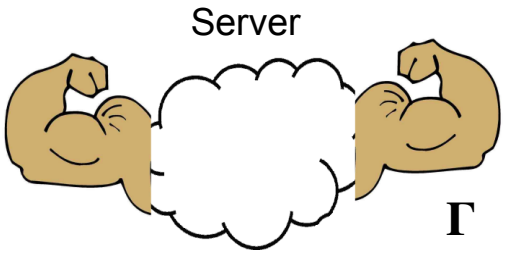
$Fenc(pk_{FHE}, c) \rightarrow c'$

c'

$Feval(pk_{FHE}, Geval(\Gamma, \cdot), c') \rightarrow Y'$

Y'

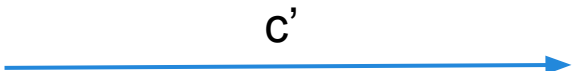
2nd try: single-client, multi-use



$Fgen(1^k) \rightarrow (pk_{FHE}, sk_{FHE})$

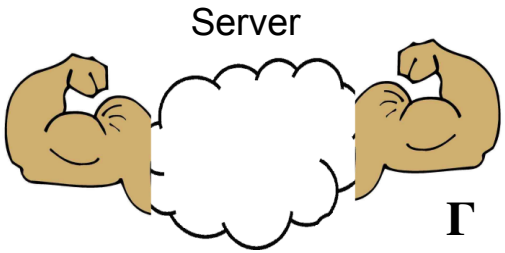
$Genc(pk, x) \rightarrow c$

$Fenc(pk_{FHE}, c) \rightarrow c'$



$Feval(pk_{FHE}, Geval(\Gamma, \cdot), c') \rightarrow Y'$

2nd try: single-client, multi-use



$Fgen(1^k) \rightarrow (pk_{FHE}, sk_{FHE})$

$Genc(pk, x) \rightarrow c$

$Fenc(pk_{FHE}, c) \rightarrow c'$

c' →

$Feval(pk_{FHE}, Geval(\Gamma, \cdot), c') \rightarrow Y'$

← Y'

$Fdec(sk_{FHE}, Y') \rightarrow Y$

$Gdec(sk, Y) \rightarrow y$

How to go multiparty?

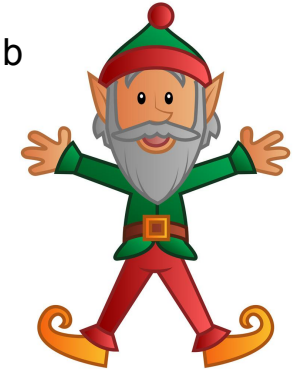
We'll need some new tools.

What prevents the 2nd try from generalizing to multiparty setting?

Oblivious Transfer (OT)

m_0
m_1

b



Oblivious Transfer (OT)

Sender

m_0
m_1

OT

Receiver

b

Oblivious Transfer (OT)

Sender

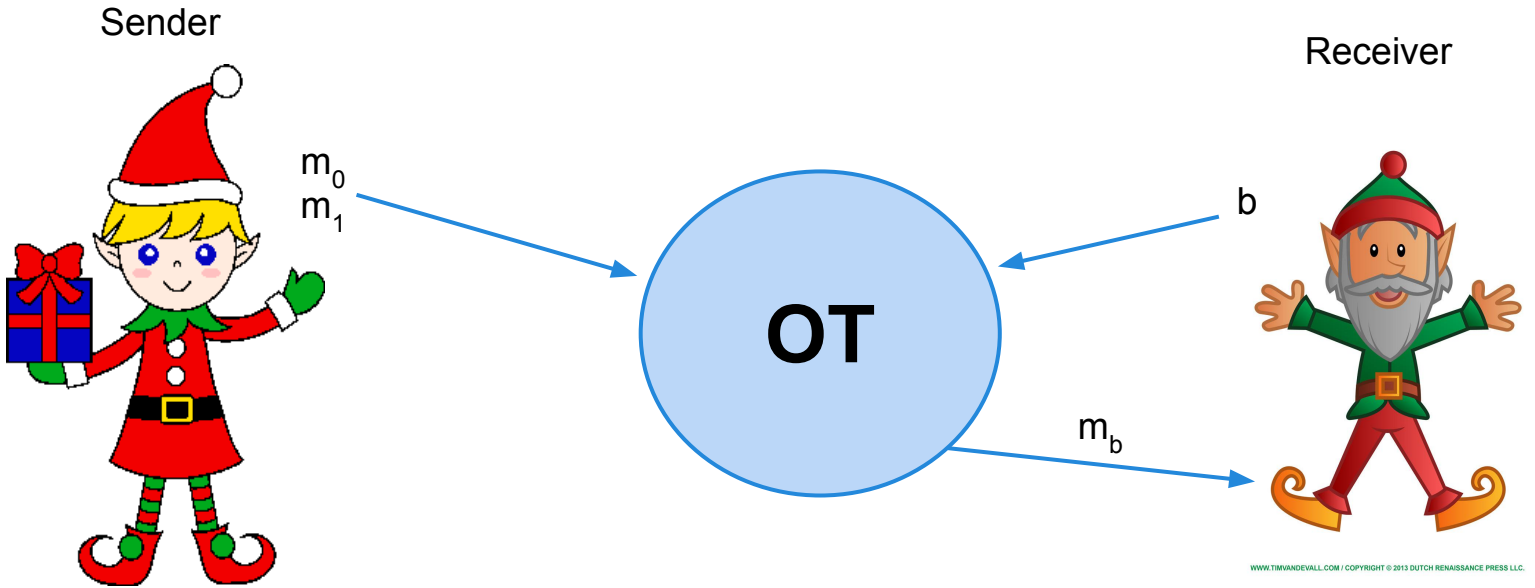
m_0
m_1

OT

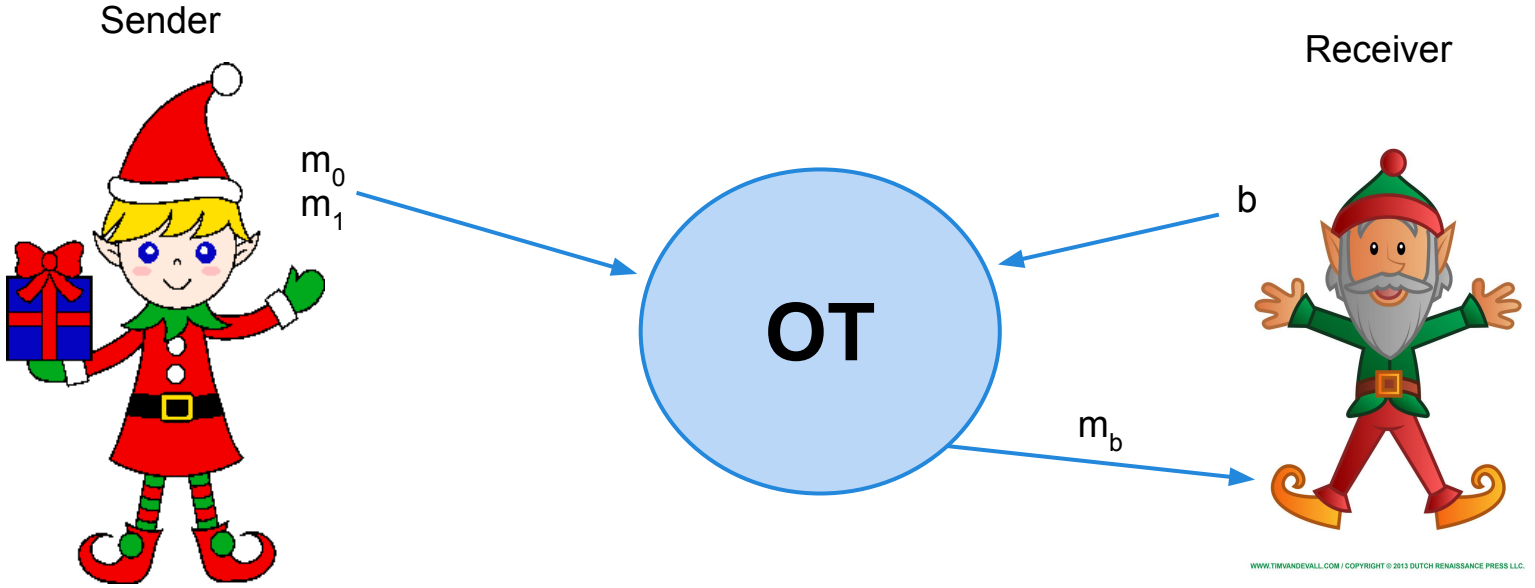
b

Receiver

Oblivious Transfer (OT)



Oblivious Transfer (OT)



- does not learn b

- learns m_b
- does not learn m_{1-b}

Proxy Oblivious Transfer (POT)

b

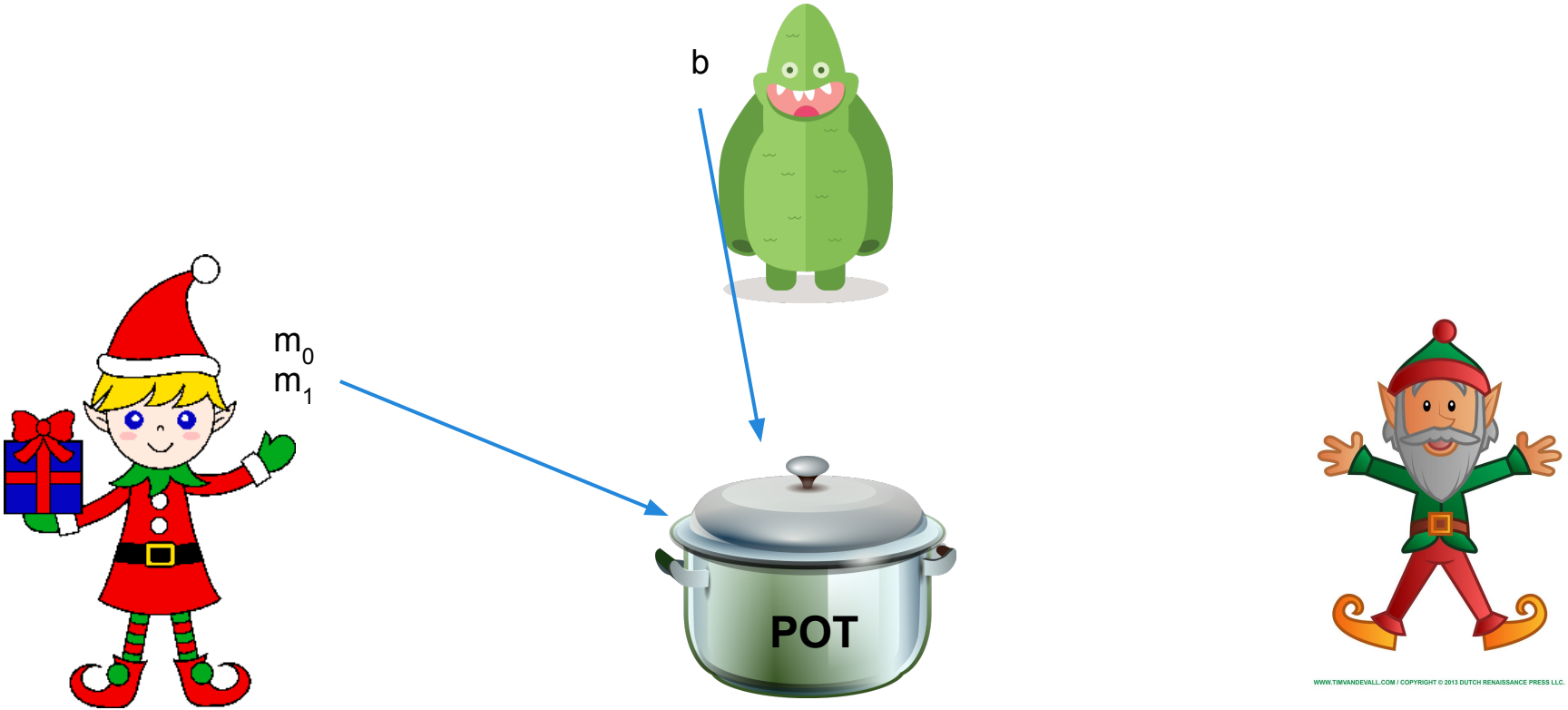
m_0
m_1

Proxy Oblivious Transfer (POT)

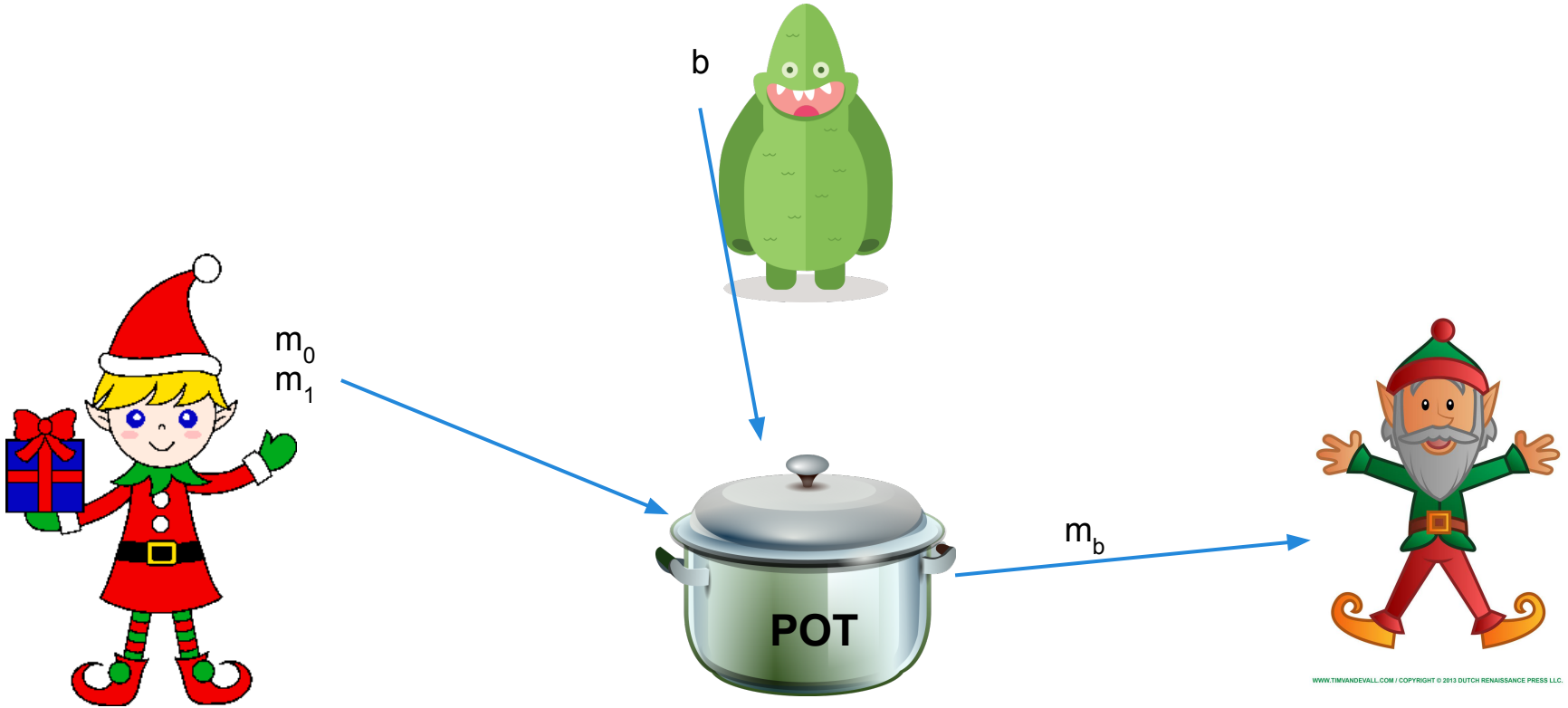
b

m_0
m_1

Proxy Oblivious Transfer (POT)



Proxy Oblivious Transfer (POT)



Proxy Oblivious Transfer (POT)

- does not learn b

m_0
m_1

b

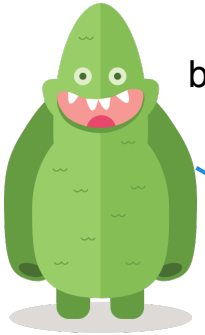
- does not learn m_0
- does not learn m_1

m_b

- learns m_b
- does not learn m_{1-b}
- does not learn b

Proxy Oblivious Transfer (POT)

What it really looks like:



b

β

m_0
m_1

α

m_b

Proxy Oblivious Transfer

- $\mathrm{SetupS}(1^k) \rightarrow (pk_s, sk_s)$
- $\mathrm{SetupC}(1^k) \rightarrow (pk_c, sk_c)$
- $\mathrm{Snd}(pk_c, sk_s, m_0, m_1) \rightarrow \alpha$
- $\mathrm{Chs}(pk_s, sk_c, b) \rightarrow \beta$
- $\mathrm{Prx}(pk_s, pk_c, \alpha, \beta) \rightarrow y$

- $y=m_b$

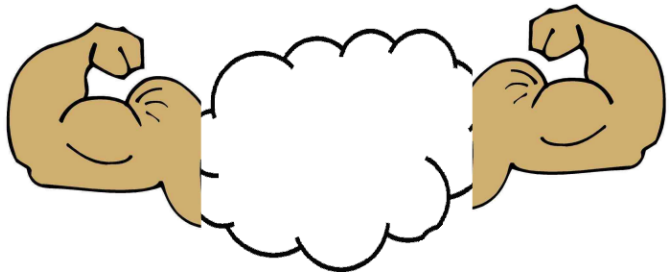
3rd try: multi-user, multi-use

Intuition: use POT to allow $P_2 ... P_n$ to pick their inputs for the Garbled Circuit

P_1

$P_{2...n}$

Server



3rd try: multi-user, multi-use

Setup Phase

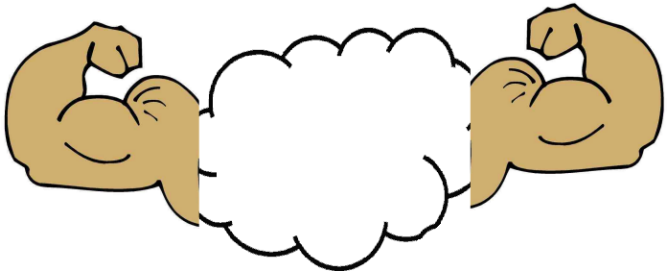
P_1

$\text{Garble}(1^k, f) \rightarrow pk, sk, \Gamma$

$\text{SetupS}(1^k) \rightarrow pk_s, sk_s$

Γ

Server



$P_{2...n}$

$\text{SetupC}(1^k) \rightarrow pk_c, sk_c$

3rd try: multi-user, multi-use

P_1

$Fgen(1^k) \rightarrow (pk_{FHE}, sk_{FHE})$

$Genc(pk, x_1) \rightarrow c_1$

$Fenc(pk_{FHE}, c_1) \rightarrow c_1'$

for i from 2 to n and for j from 1 to l
(l = length of input):

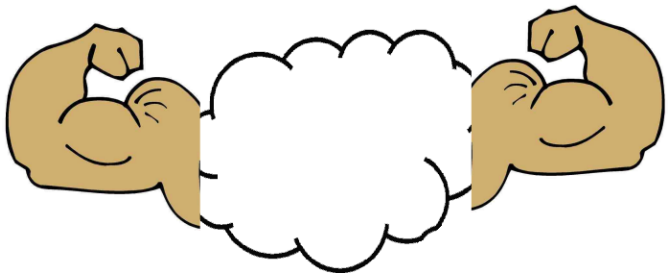
$x_0 = Fenc(sk_{FHE}, Genc(pk, 0))$

$x_1 = Fenc(sk_{FHE}, Genc(pk, 1))$

$Snd(pk_{ci}, sk_s, x_0, x_1) \rightarrow \alpha_{ij}$

c_1', α

Server



3rd try: multi-user, multi-use

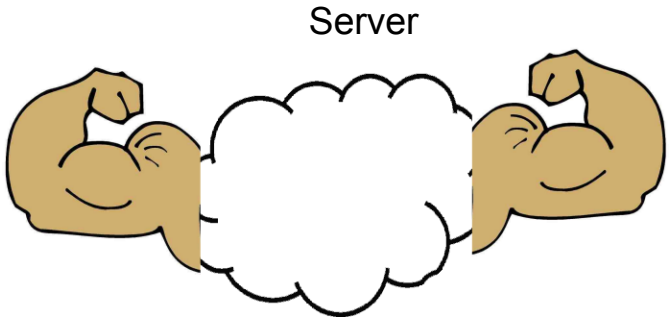
$P_{2...n}$

for j from 1 to l:

$Chs(pk_s, sk_{ci}, x_{ij}) \rightarrow \beta_{ij}$

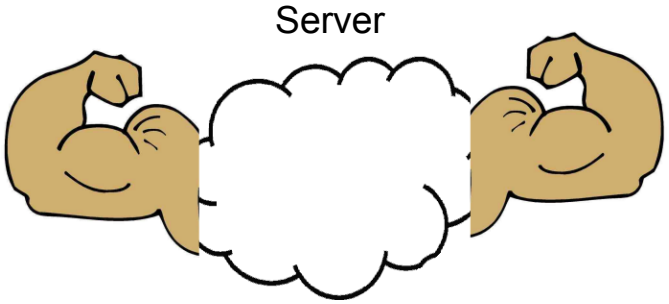
Send all β_{ij} to Server

β_i



3rd try: multi-user, multi-use

Y'

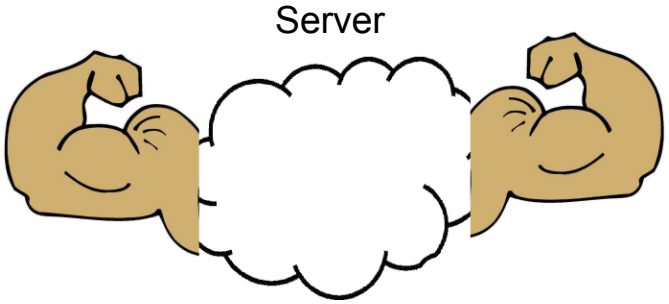


For each i,j: $Prx(pk_s, pk_{ci}, \alpha_{ij}, \beta_{ij}) \rightarrow c_{ij}'$

Reassemble c_1' and all c_{ij}' in order of subscript into c'

$Feval(pk_{FHE}, Geval(\Gamma, \cdot), c') \rightarrow Y'$

3rd try: multi-user, multi-use



Y'

For each i,j: Prx(pk_s,pk_{ci}, α_{ij}, β_{ij}) → c_{ij}'

Reassemble c_1' and all c_{ij}' in order of subscript into c'

Feval(pk_{FHE}, Geval(Γ, ·), c') → Y'

Fdec(sk_{FHE}, Y') → Y

Gdec(sk, Y) → y

Summary

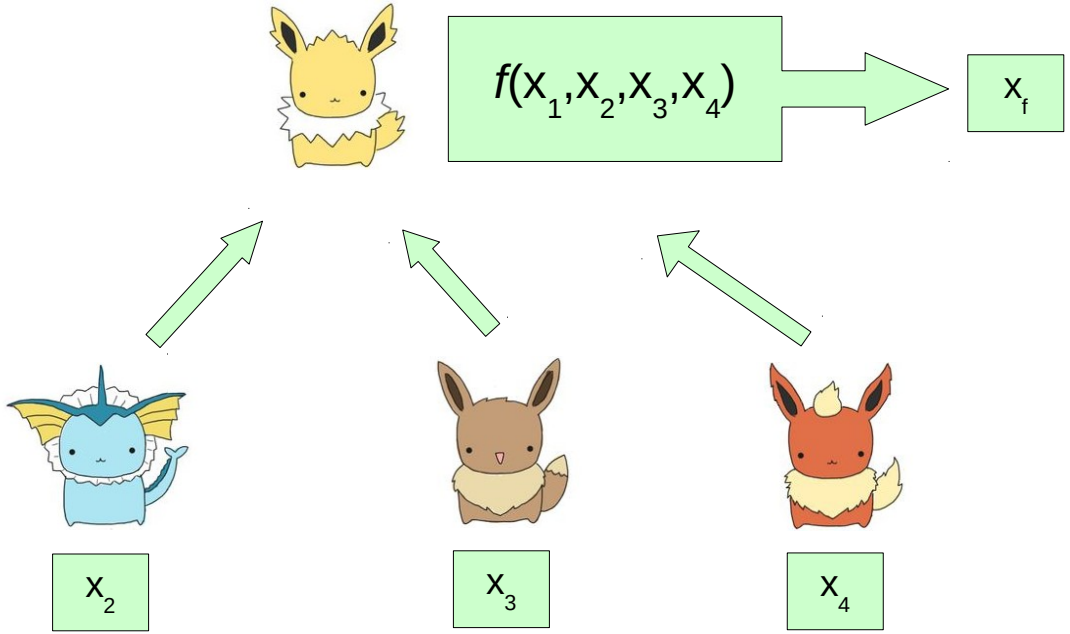
- Extended Verifiable Computation to multiparty setting
- Used POT, Garbled Circuits, and FHE
- Compute Garbled Circuit inside FHE
- Use POT so clients $P_{2...n}$ can pick from inputs sent by P_1

- Next: Secure Computation

On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encrptyion

Multiparty Computation

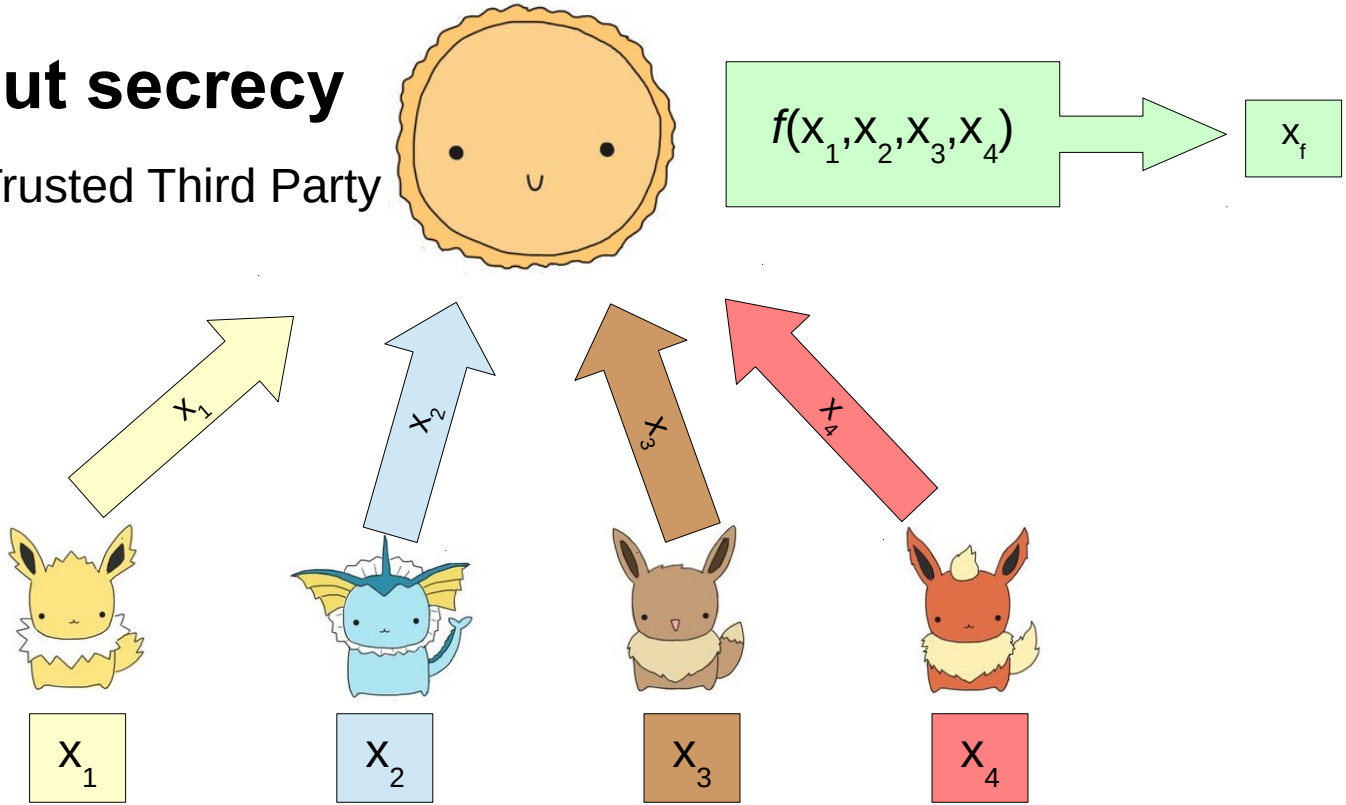
Without input secrecy



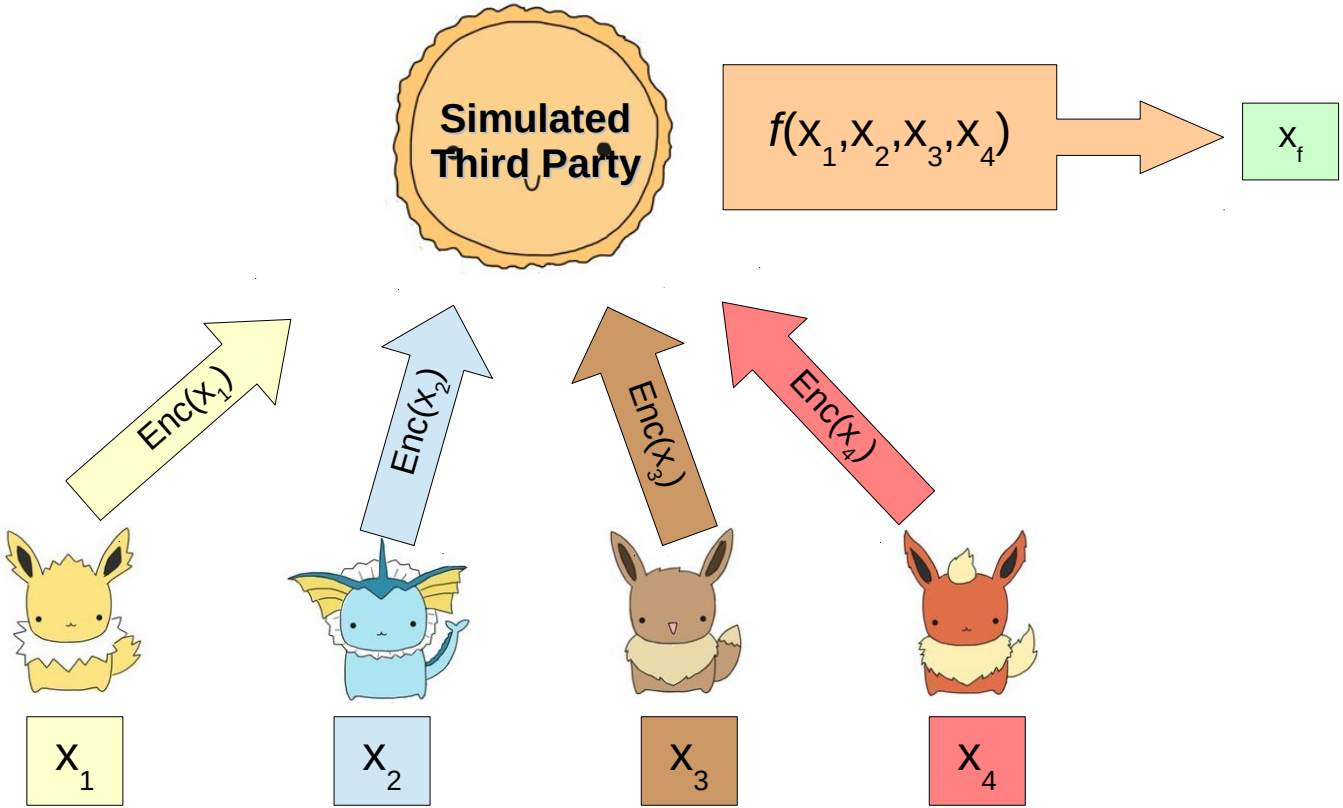
Multiparty Computation

With input secrecy

Trusted Third Party



Secure Multiparty Computation (MPC)



Secure Multiparty Computation (MPC)

- **Security Models**
  - **Semi-Honest Adversaries** follow the protocol as described but try to learn information about honest player's input
  - **Malicious Adversaries** can deviate from the protocol to learn honest player's input

Secure Multiparty Computation (MPC)

- **Security Models**
  - **Semi-Malicious Adversaries** are like semi-honest adversaries but can sample random elements from any arbitrary distribution

Secure Multiparty Computation (MPC)

- **Problem with regular MPC?**
  - Highly interactive
  - One-time use
  - Assume all parties are equally powerful

On-the-Fly MPC on the Cloud

Powerful Cloud

Weak Clients

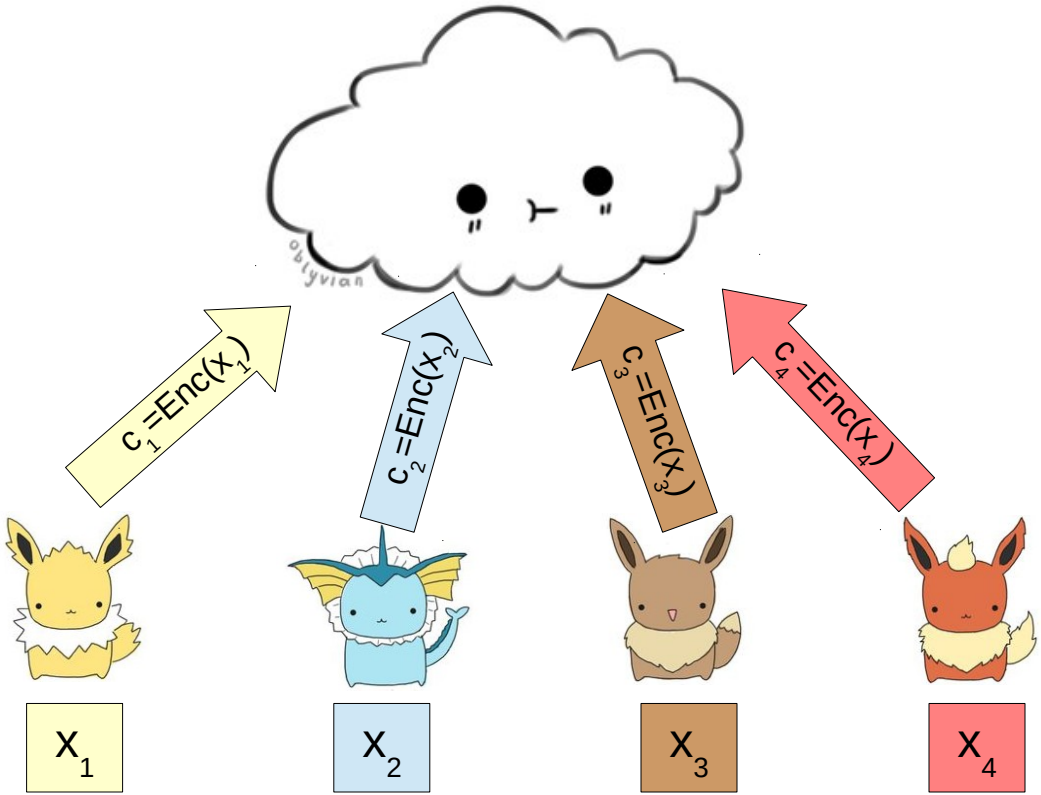
x_1

x_2

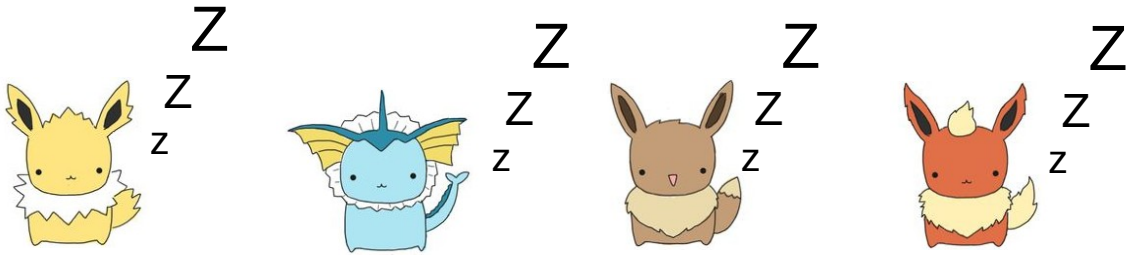
x_3

x_4

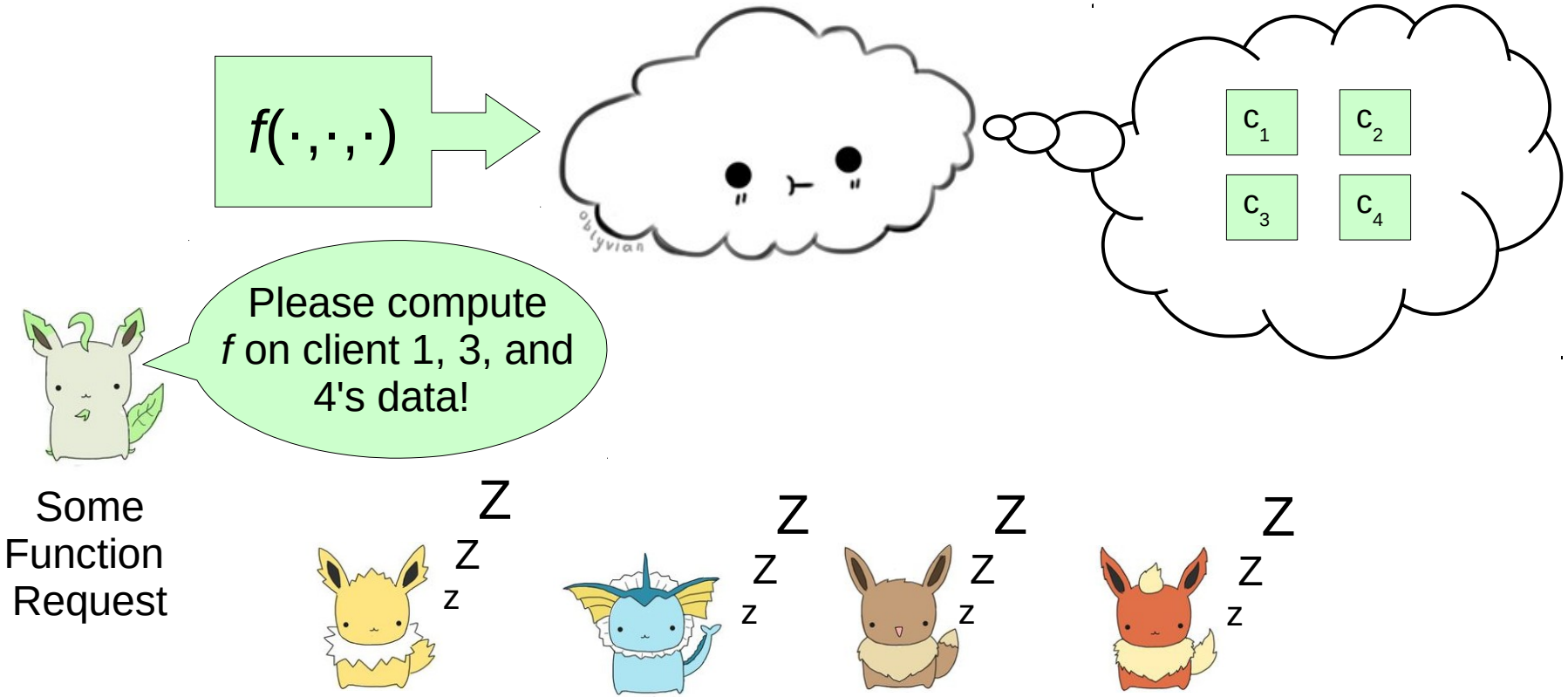
On-the-Fly MPC on the Cloud



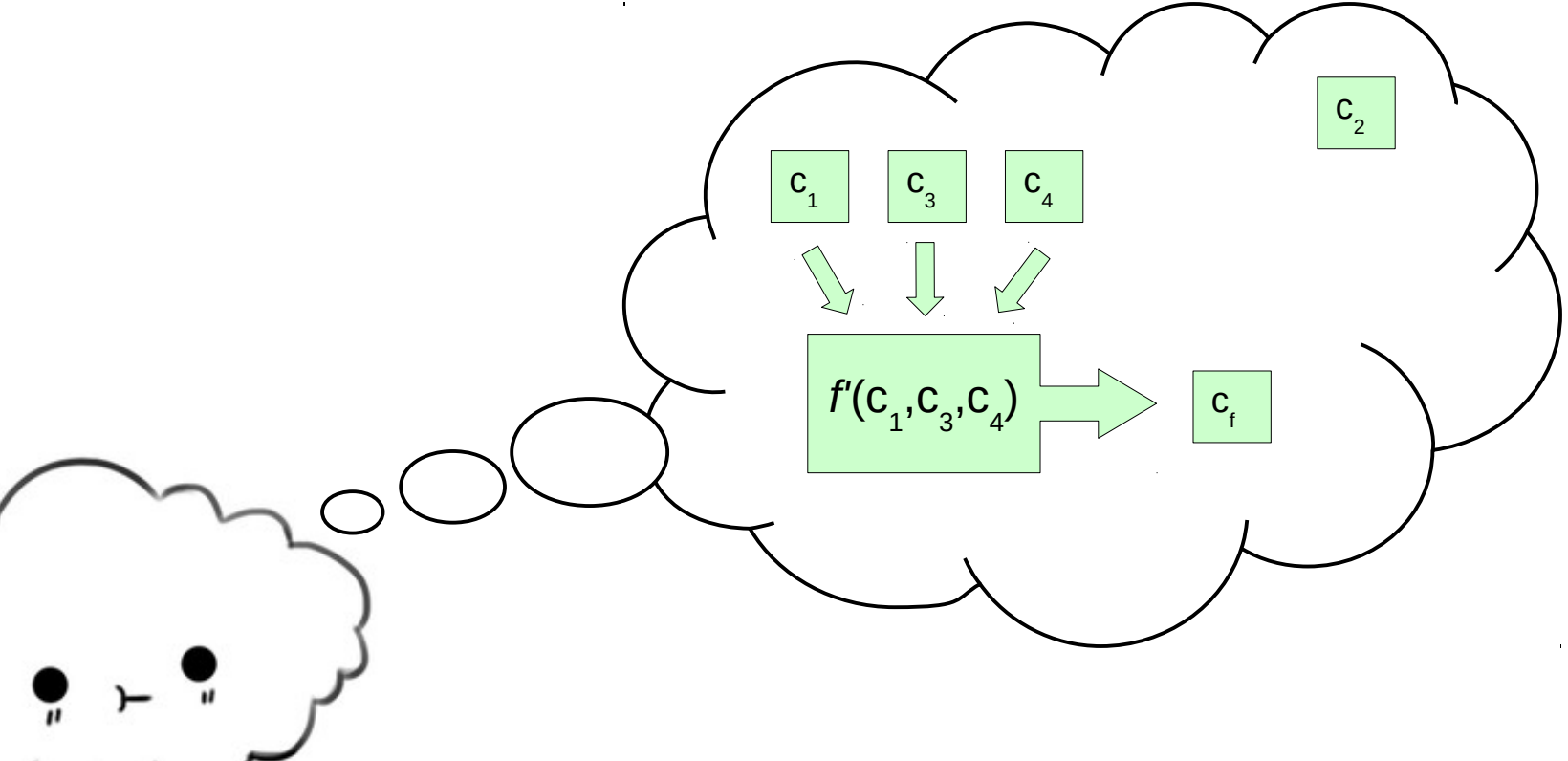
On-the-Fly MPC on the Cloud



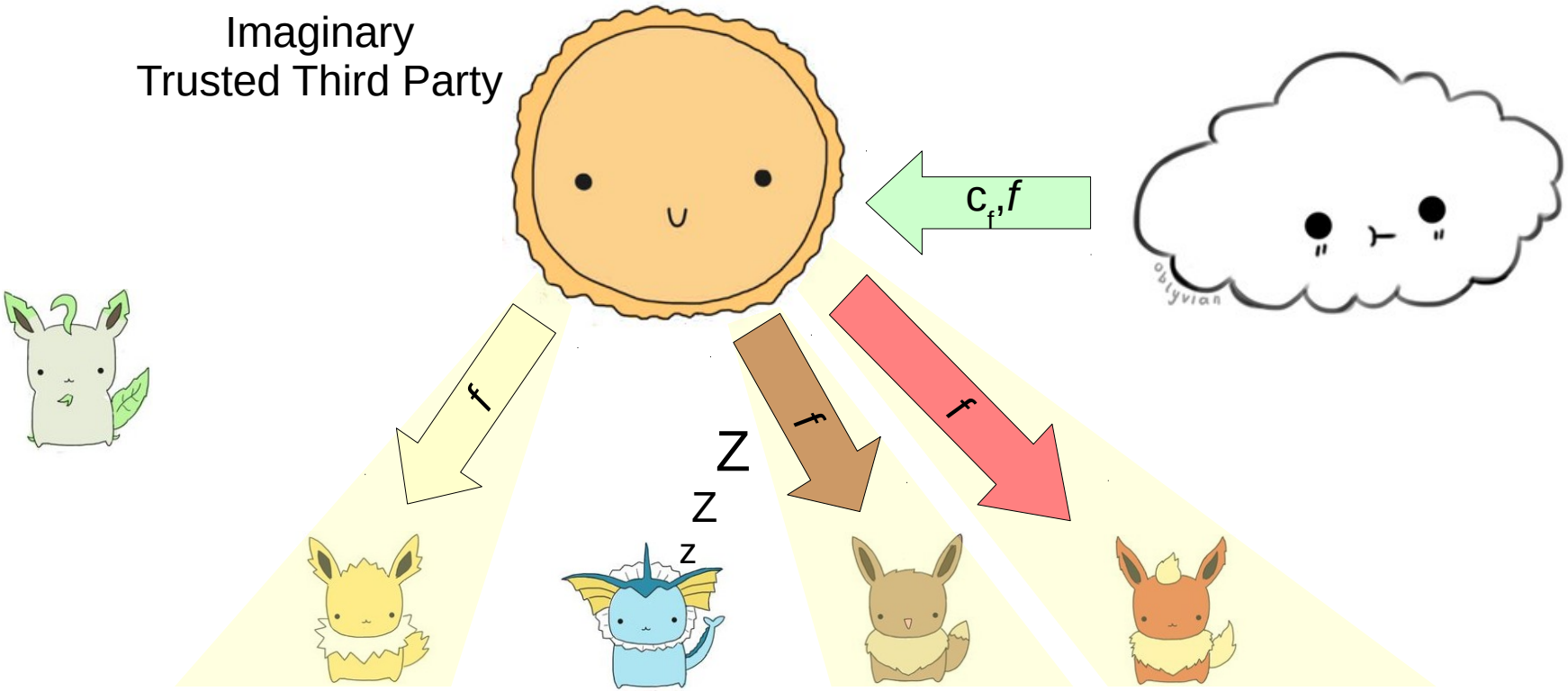
On-the-Fly MPC on the Cloud



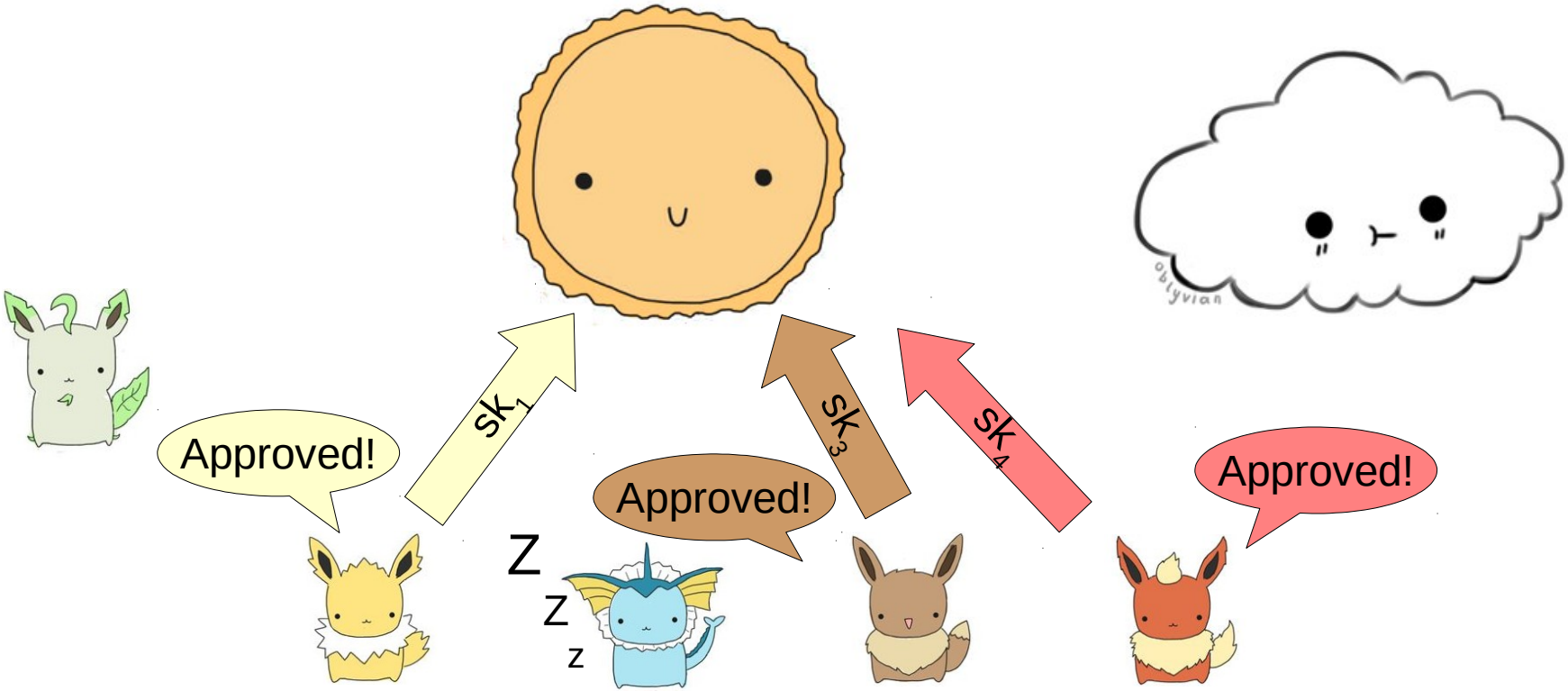
On-the-Fly MPC on the Cloud



On-the-Fly MPC on the Cloud

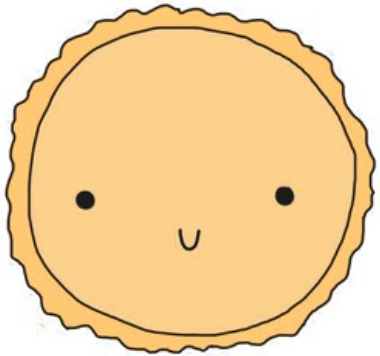


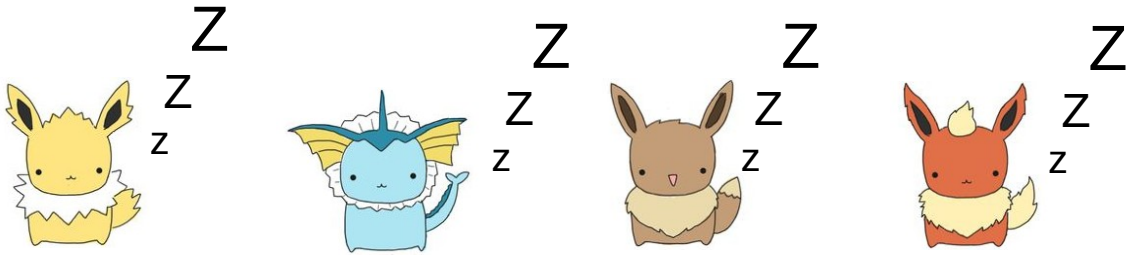
On-the-Fly MPC on the Cloud



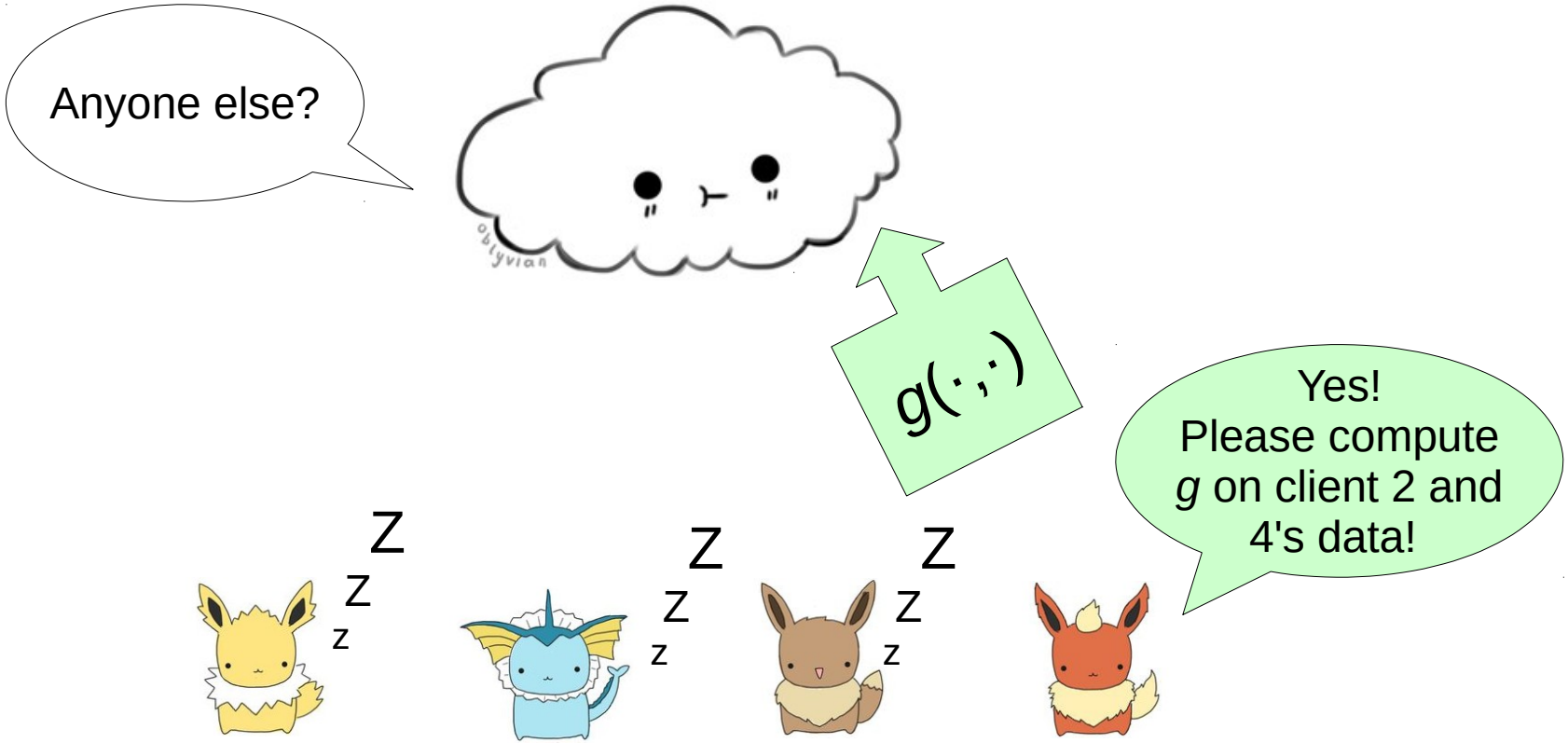
On-the-Fly MPC on the Cloud

$f(x_1,x_3,x_4)=\mathrm{Dec}(c_i)$





On-the-Fly MPC on the Cloud



Properties

We want:

- *Dynamically* chosen functions *on-the-fly*
- Functions compute on an *arbitrary* subset of data
- *Non-interactive* function evaluation *on the cloud*

Properties

We want:

- Clients are *unaware* of the identity or the number of other clients
- Relevant clients approve the choice of functions after evaluation and before decryption

Semi-Malicious Construction: Building Blocks

Multikey Fully Homomorphic Encryption

$\{\mathcal{E}^{(N)}=(\text{Keygen},\text{Enc},\text{Dec},\text{Eval})\}_{N>0}$

where N is the number of key pairs

- **Key Generation**
  - $(pk,sk,ek)\leftarrow\text{Keygen}(1^k)$

Semi-Malicious Construction: Building Blocks

Multikey Fully Homomorphic Encryption

- **Encryption**
  - $c \leftarrow Enc(pk,m)$
- **Decryption**
  - $m := Dec(sk_1,\ldots,sk_N,c)$

Semi-Malicious Construction: Building Blocks

Multikey Fully Homomorphic Encryption

- **Evaluation**
  - $c:=Eval(C,(c_1,pk_1,ek_1),\ldots,(c_l,pk_l,ek_l))$

    where each tuple of $\{(pk_i,sk_i,ek_i)\}_{i\in[l]}$ is in $\{(pk_j,sk_j,ek_j)\}_{j\in[N]}$

Semi-Malicious Construction: Building Blocks

Multikey Fully Homomorphic Encryption

- **Correctness**
  - $\mathrm{Dec}(sk_1,\ldots,sk_N,c)=C(m_1,\ldots,m_l)$
- **Compactness**
  - $|c|\leq P(k,N)$

Semi-Malicious Construction: Building Blocks

Multikey Fully Homomorphic Encryption

- **Compactness** is important, since we don't want the running time of decryption to depend on the size of circuits

Semi-Malicious Construction: Building Blocks

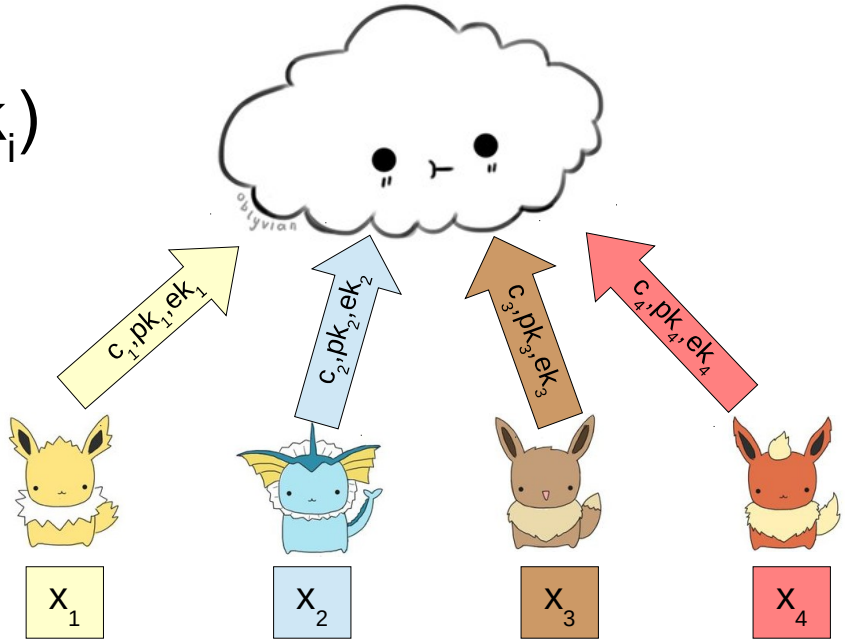
Semi-Malicious Secure MPC

- **Server-Aided Secure MPC**
  - Powerful server and weak clients
  - Most computation are carried out by the server
  - Clients only communicate with the server

Semi-Malicious Construction: Where the Building Blocks fit

Offline Phase

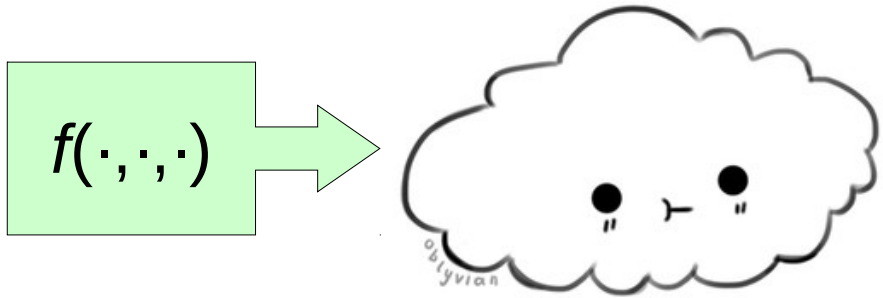
1) Clients samples (pk_i, sk_i, ek_i)
2) $c_i \leftarrow Enc(pk_i, x_i)$
3) Send c_i, pk_i, ek_i to the server



Semi-Malicious Construction: Where the Building Blocks fit

Online Phase

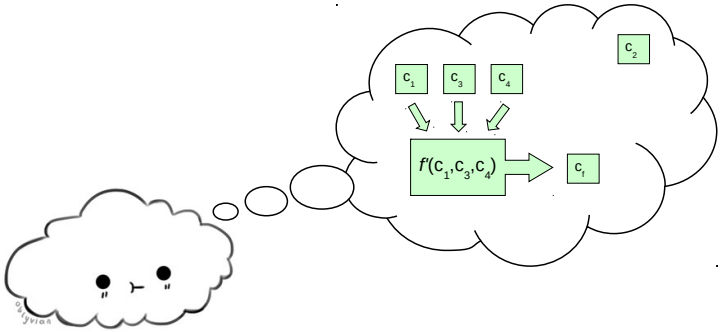
- Choose a function



Semi-Malicious Construction: Where the Building Blocks fit

Online Phase: Step 1

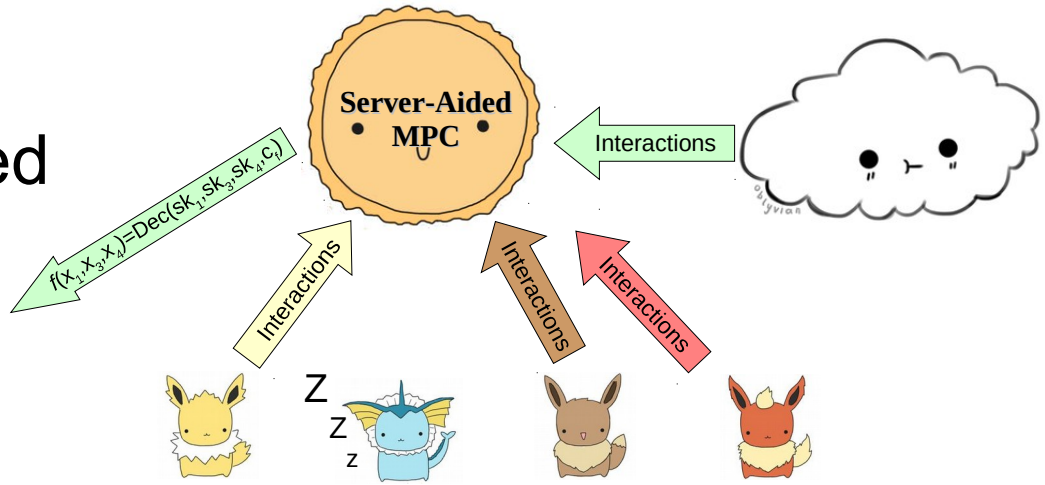
1) The server performs multikey FHE
2) The server broadcasts the evaluated ciphertext to all computing parties



Semi-Malicious Construction: Where the Building Blocks fit

Online Phase: Step 2

- Run server-aided MPC protocol to decrypt the evaluated ciphertext



Problems with the Semi-Malicious Construction

1) Clients may encrypt data *incorrectly*
2) Server may compute functions *incorrectly*
3) Clients may cheat in semi-malicious MPC
4) Clients may use an invalid secret key to decrypt the resulting ciphertext

Malicious Construction

Clients may encrypt data incorrectly

Fix: Clients must prove this NP relation to server:

$\mathbf{R^{ENC}=\{ ((pk,c),(x,s)) \mid c = Enc(pk,x; s) \}}$

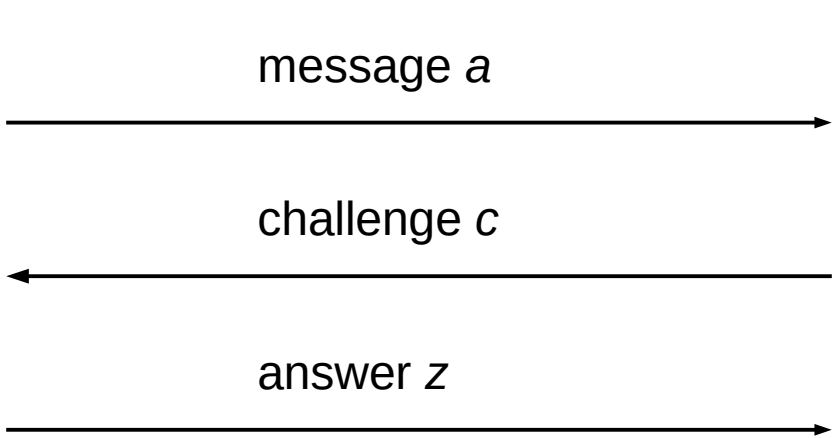
where **s** is some random string

How? **Zero-knowledge Proof**

Malicious Construction

Non-Interactive Zero Knowledge (NIZK)

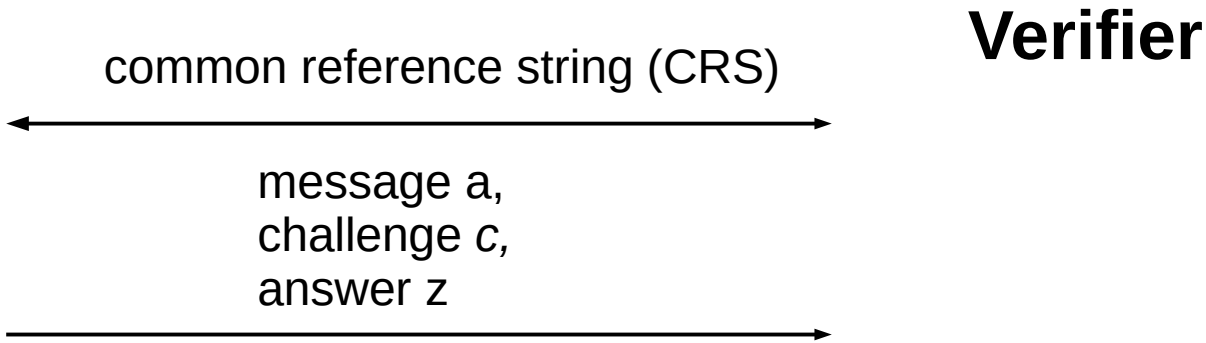
- (Interactive) Zero-Knowledge Proof



Malicious Construction

Non-Interactive Zero Knowledge (NIZK)

- NIZK



Malicious Construction

Server may compute functions incorrectly

Fix: Server verifies its computation to clients

- Use **succinct non-interactive arguments of knowledge (SNARK)**
- Similar to PCP theorem, but non-interactive

Malicious Construction

Clients may cheat in semi-malicious MPC

Fix: Replace semi-malicious MPC with malicious MPC

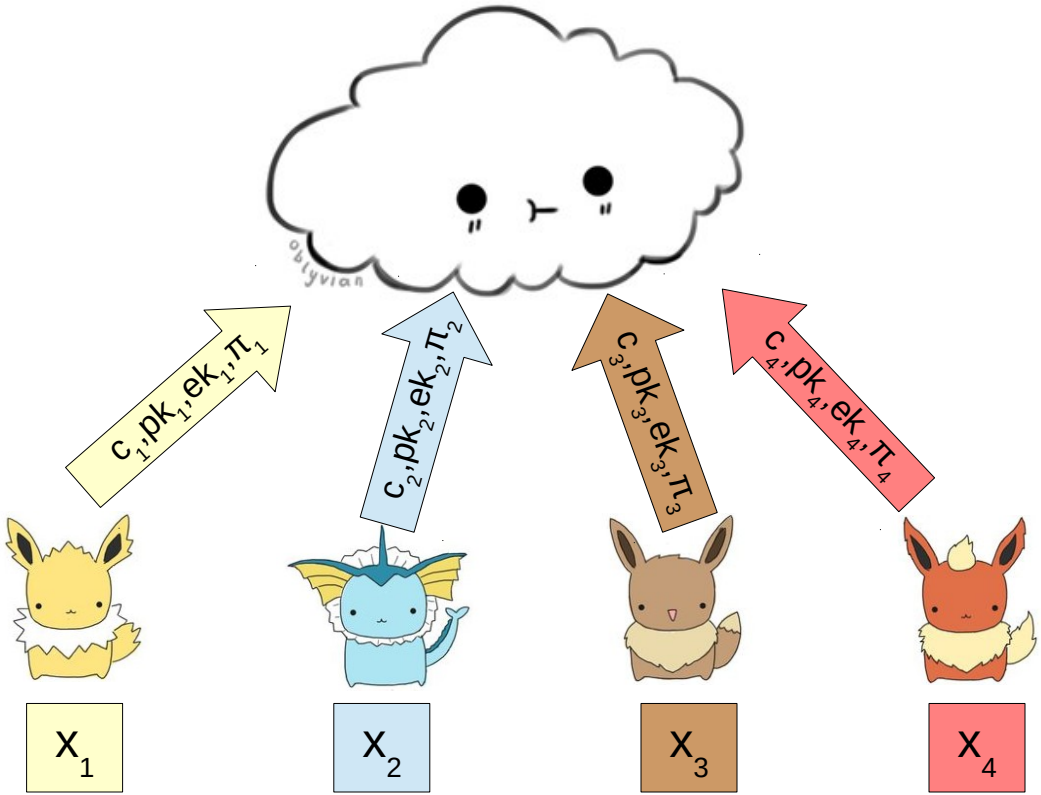
Malicious Construction

Clients may use an invalid secret key to decrypt the resulting ciphertext

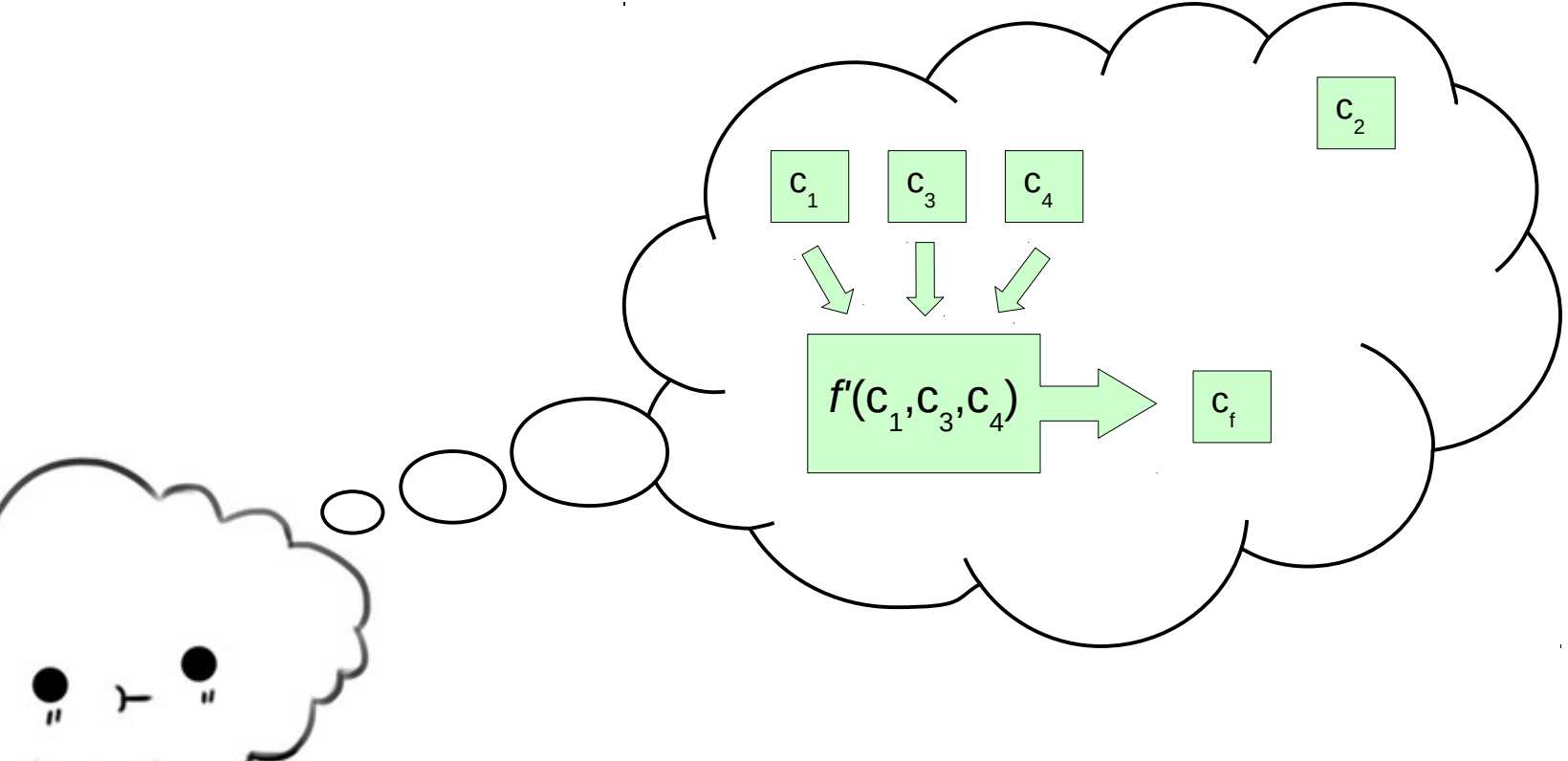
Fix: Change the decryption function to this:

$$g_{c,\mathsf{pk}_1,\mathsf{ek}_1,\ldots,\mathsf{pk}_N,\mathsf{ek}_N}((\mathsf{sk}_1, r_1)\ldots,(\mathsf{sk}_N, r_N))$$
$$\stackrel{\text{def}}{=} \begin{cases} \mathsf{Dec}(\mathsf{sk}_1,\ldots,\mathsf{sk}_N, c) & \text{if } (\mathsf{pk}_i,\mathsf{sk}_i,\mathsf{ek}_i) = \mathsf{Keygen}(1^\kappa; r_i) \quad \forall i \in [N] \\ \bot & \text{otherwise} \end{cases}$$

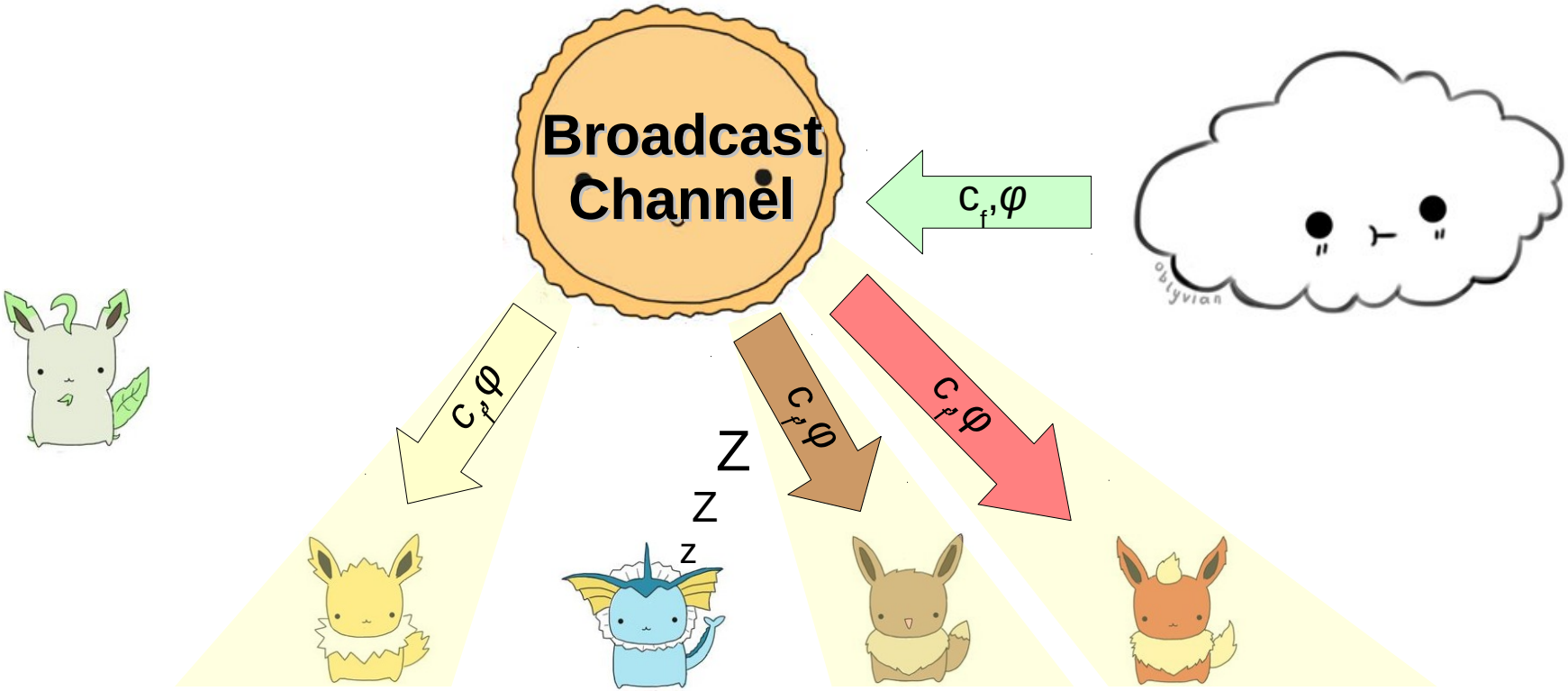
Malicious Construction: Summary



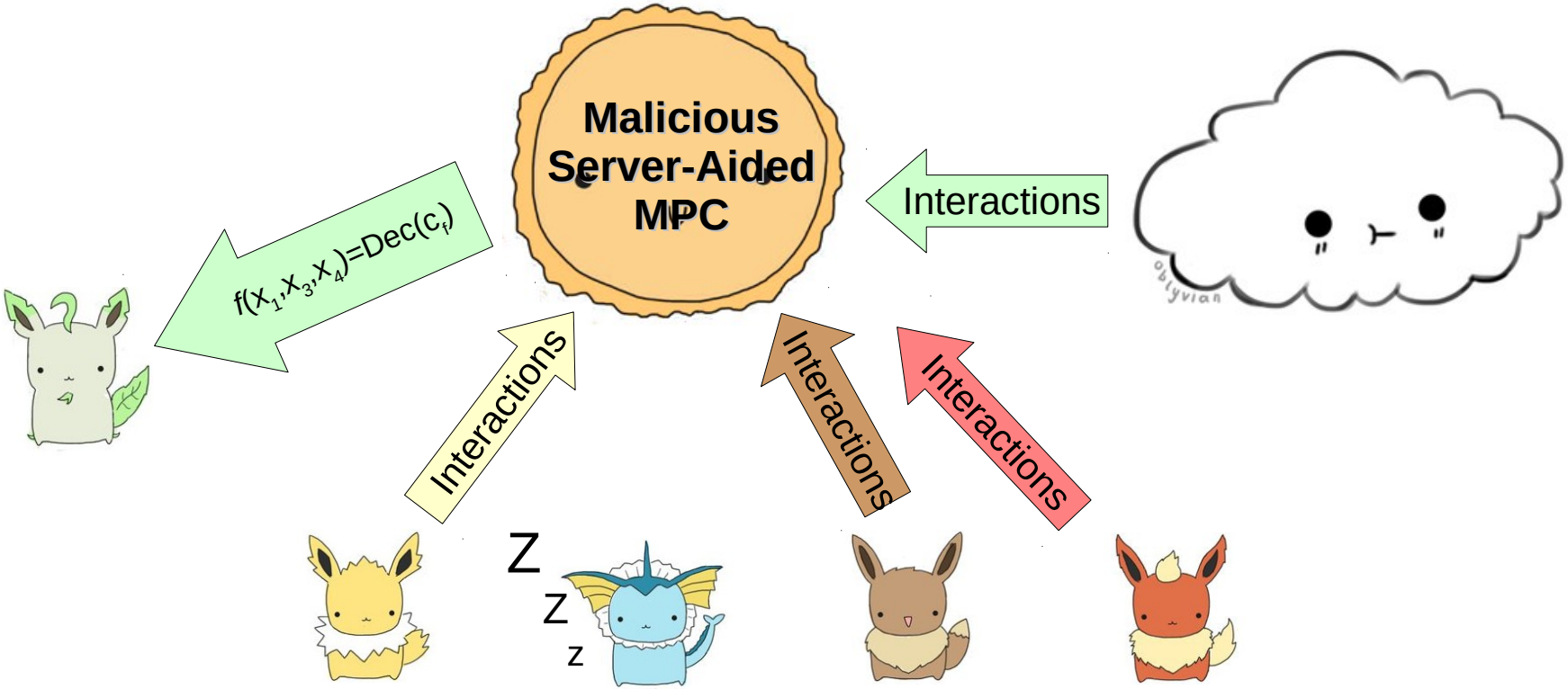
Malicious Construction: Summary



Malicious Construction: Summary

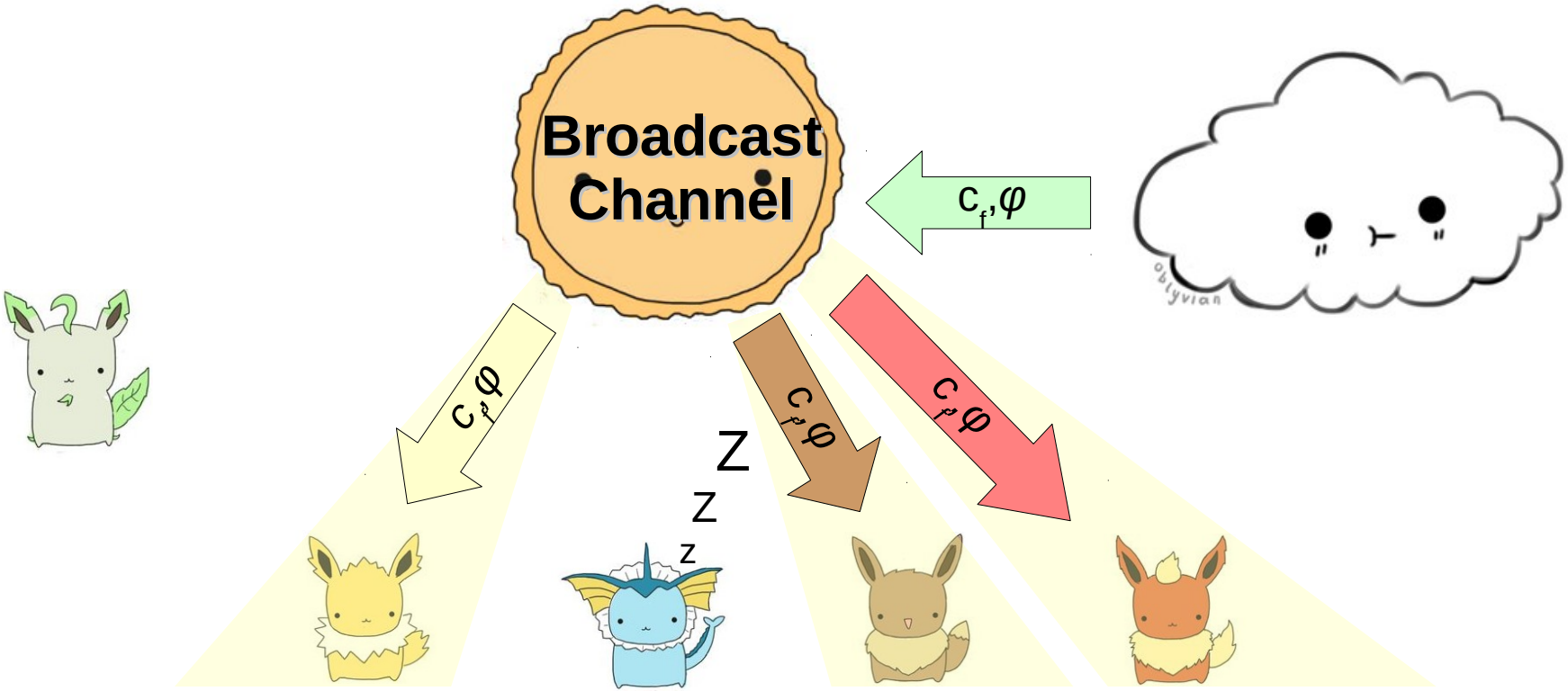


Malicious Construction: Summary

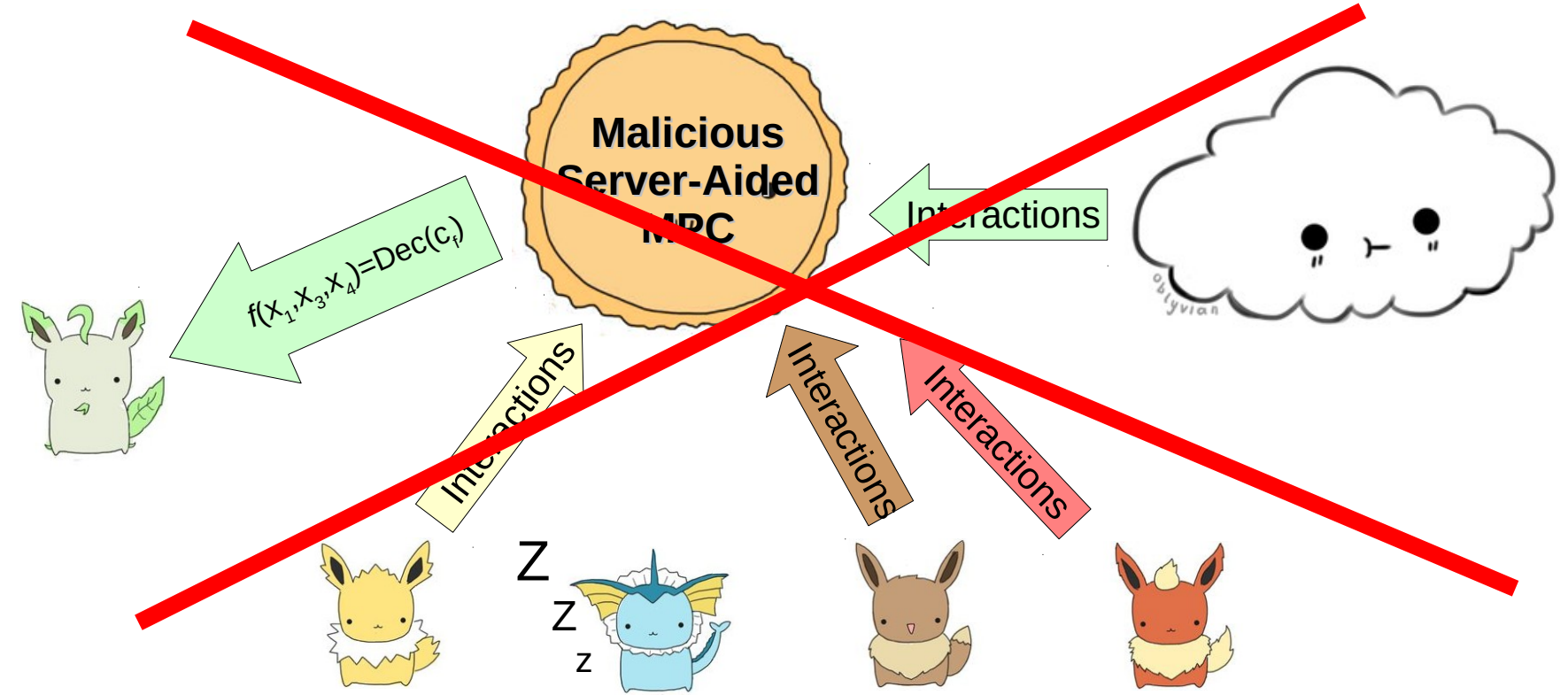


How about a protocol with fewer interactions?

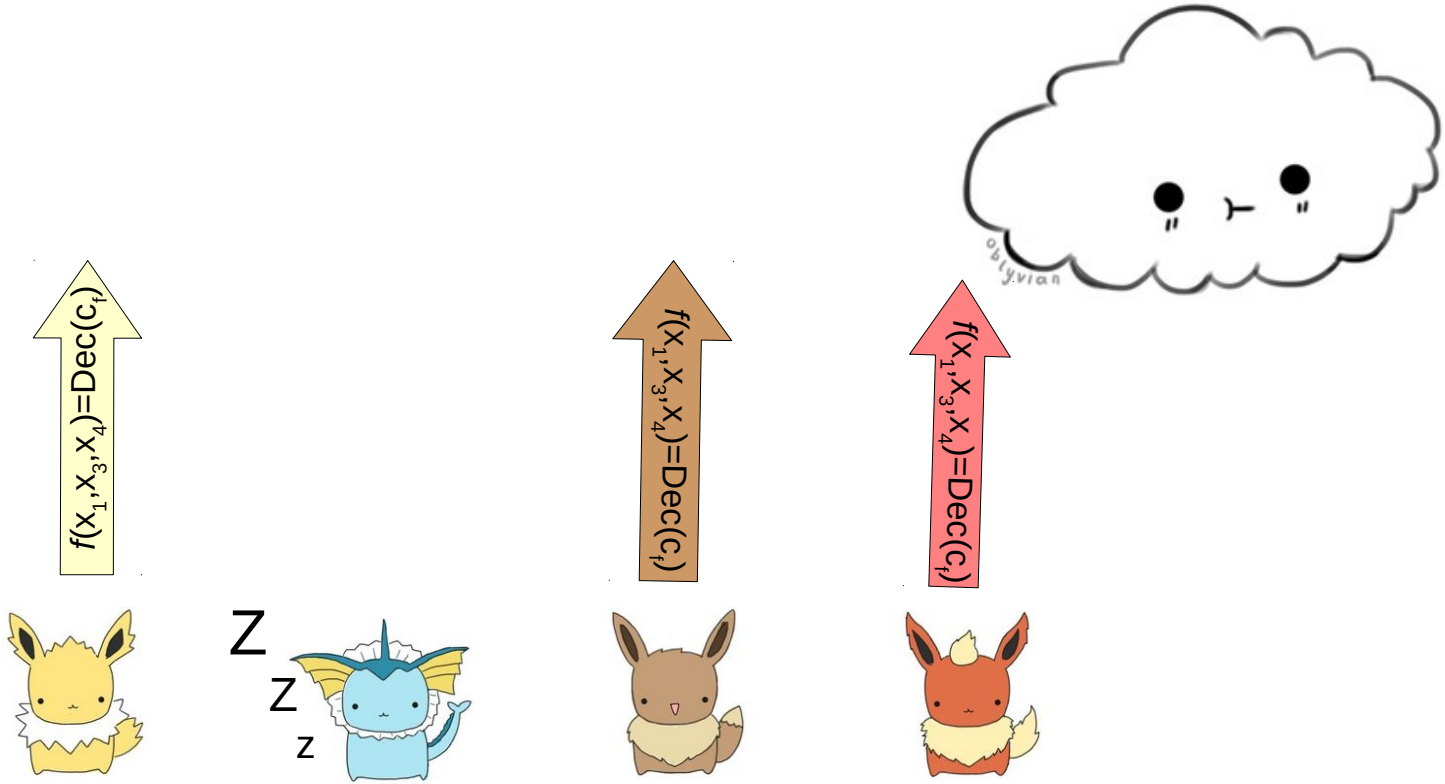
Protocol with Fewer Interactions



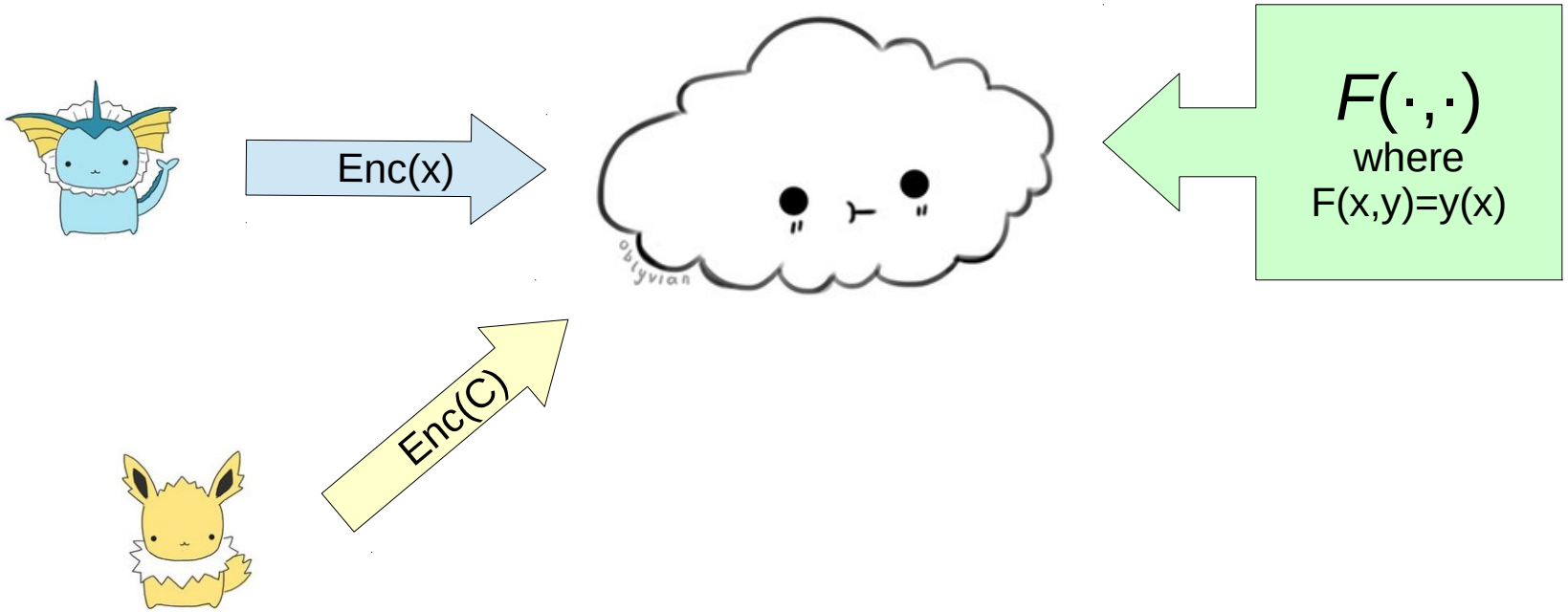
Protocol with Fewer Interactions



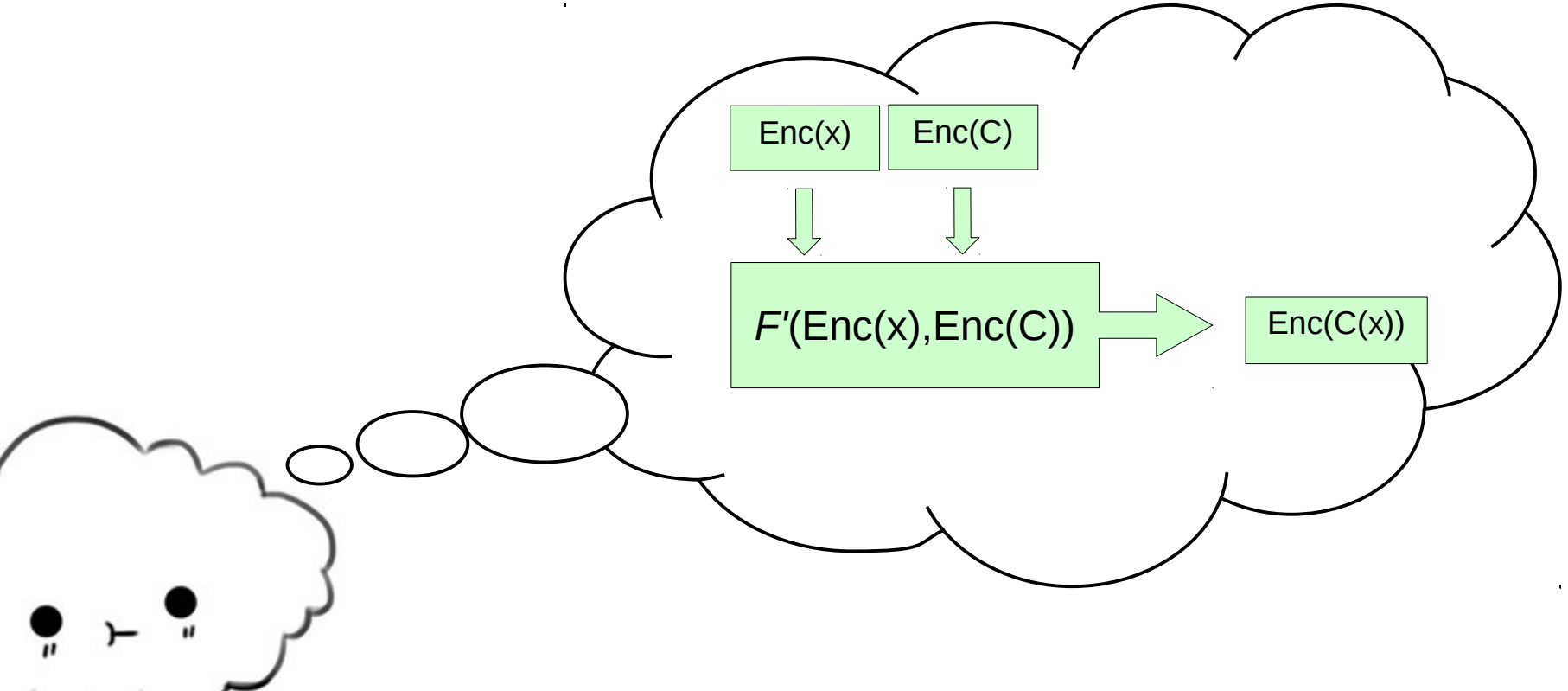
Protocol with Fewer Interactions



Reduction to VBB Obfuscation



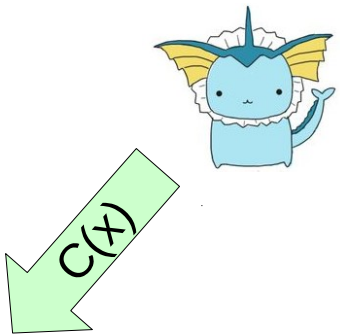
Reduction to VBB Obfuscation



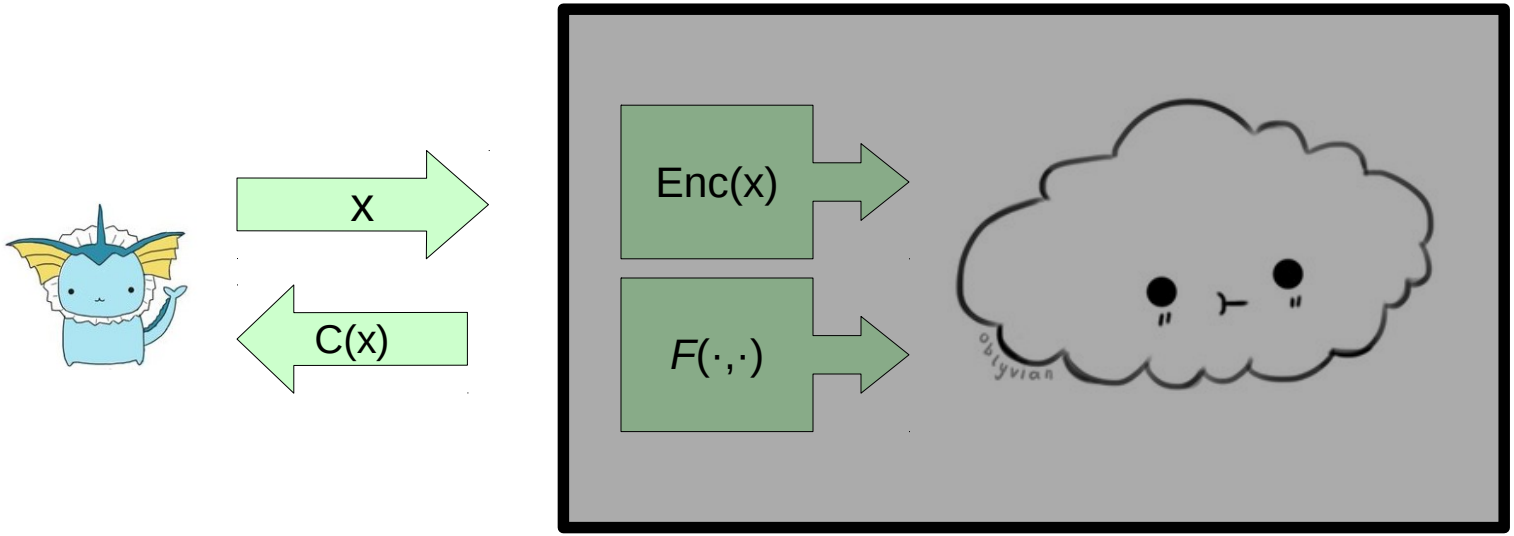
Reduction to VBB Obfuscation

Enc(C(x))

Reduction to VBB Obfuscation



Reduction to VBB Obfuscation



Questions?