Highly Efficient OT-Based Multiplication Protocols

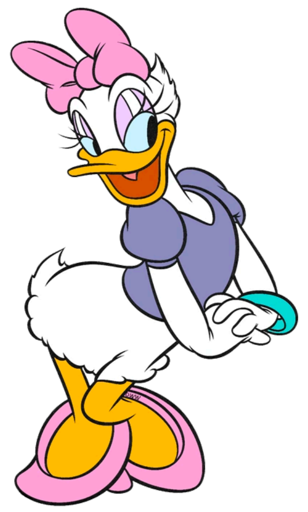
Nikolaos Makriyannis (Fireblocks)

Joint work w/ **Iftach Haitner**, **Samuel Ranellucci** & **Eliad Tsfadia**

Problem Statement (Two-Party Multiplication)

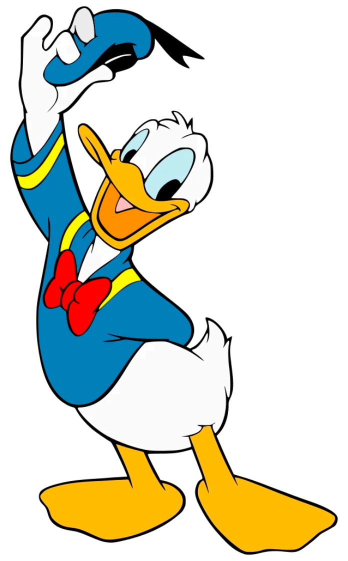
$\mathsf{Mult}(x, y) \mapsto (s_1, s_2 = x \cdot y - s_1)$

MULT



$\mathscr{P}_1$

$x \in \mathbb{Z}_q$

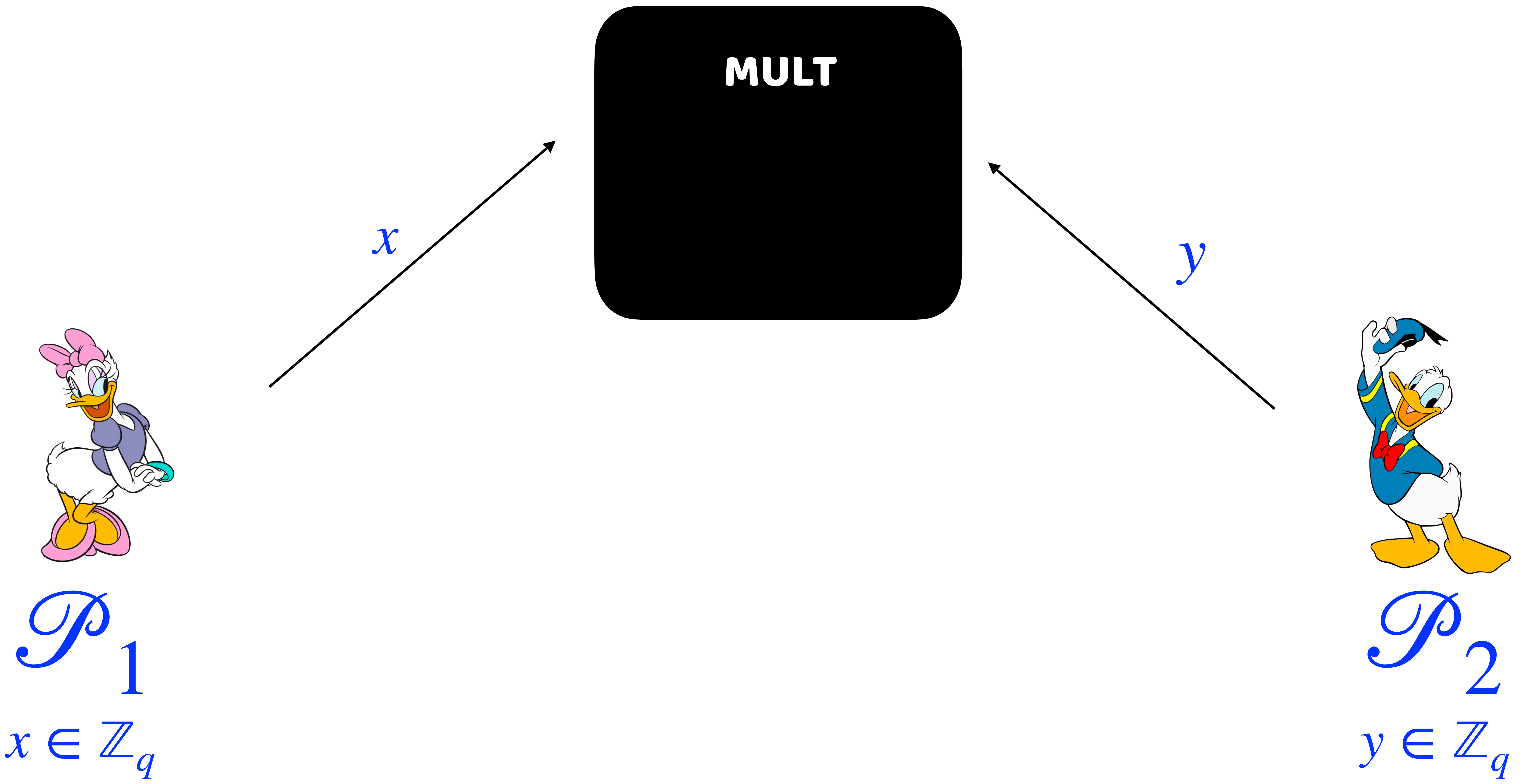


$\mathscr{P}_2$

$y \in \mathbb{Z}_q$

Problem Statement (Two-Party Multiplication)

$\mathsf{Mult}(x, y) \mapsto (s_1, s_2 = x \cdot y - s_1)$



Problem Statement (Two-Party Multiplication)

$\mathsf{Mult}(x, y) \mapsto (s_1, s_2 = x \cdot y - s_1)$

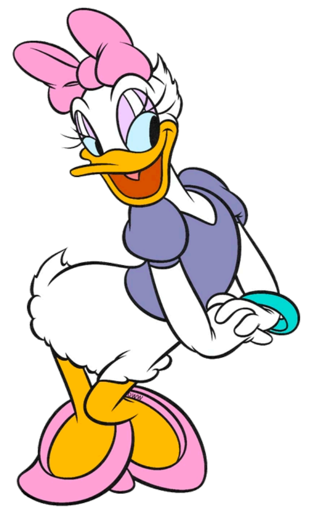
MULT

$s_2 = x \cdot y - s_1$

$s_1 \leftarrow \mathbb{Z}_q$

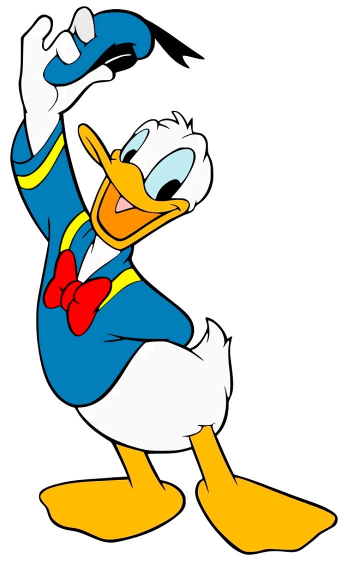
x

y



$\mathcal{P}_1$

$x \in \mathbb{Z}_q$

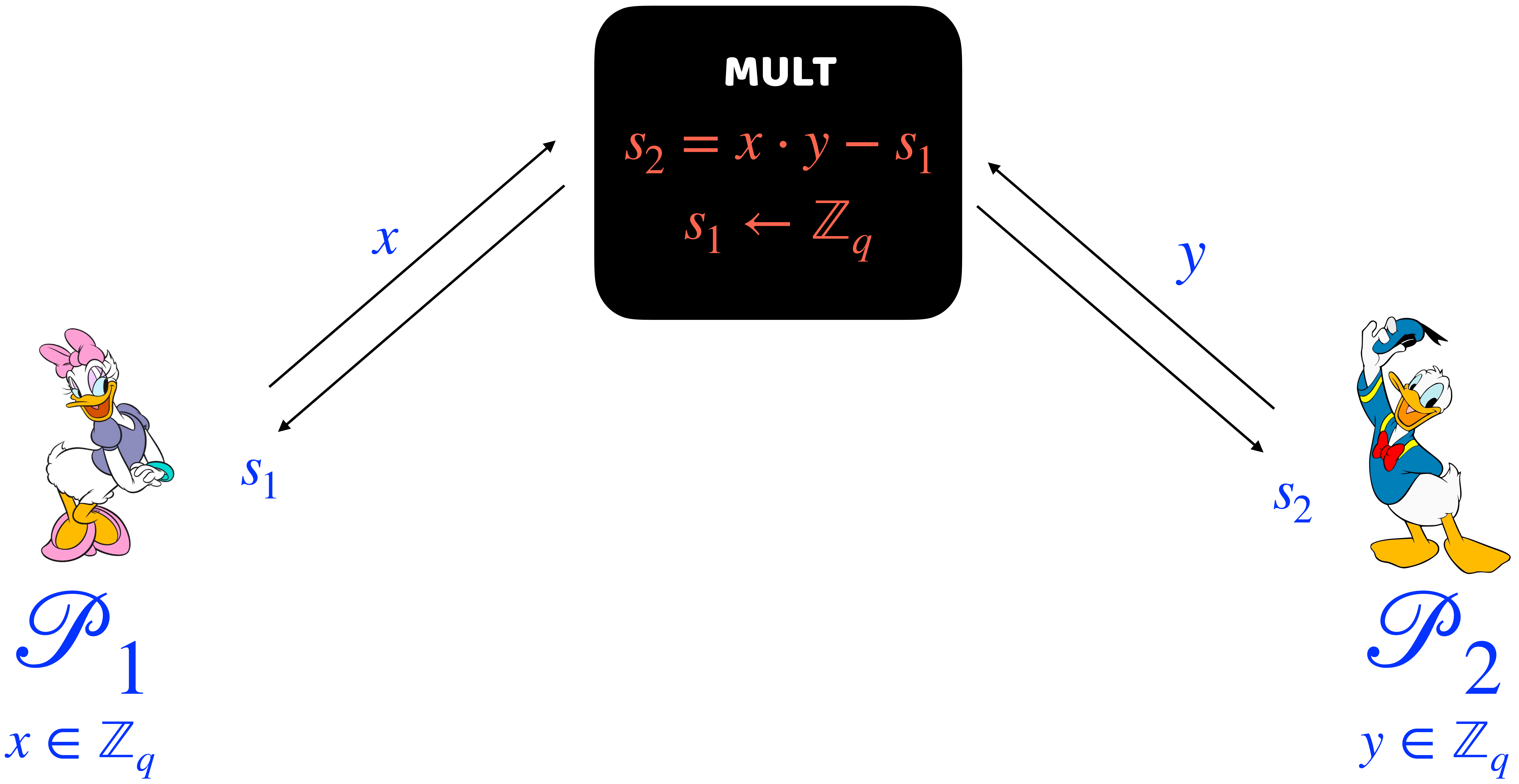


$\mathcal{P}_2$

$y \in \mathbb{Z}_q$

Problem Statement (Two-Party Multiplication)

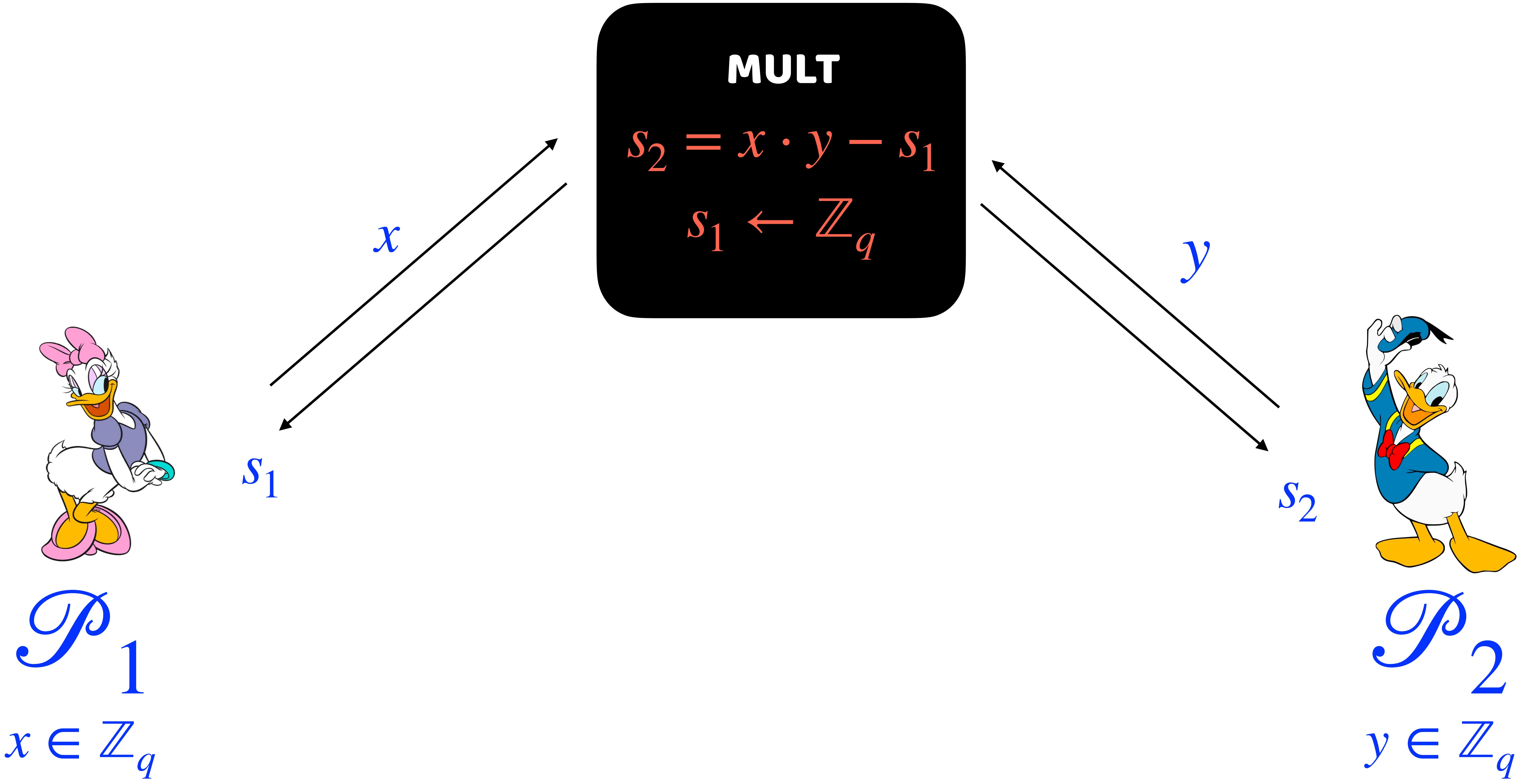
$\mathsf{Mult}(x, y) \mapsto (s_1, s_2 = x \cdot y - s_1)$



Problem Statement (Two-Party Multiplication)

$\mathsf{Mult}(x, y) \mapsto (s_1, s_2 = x \cdot y - s_1)$

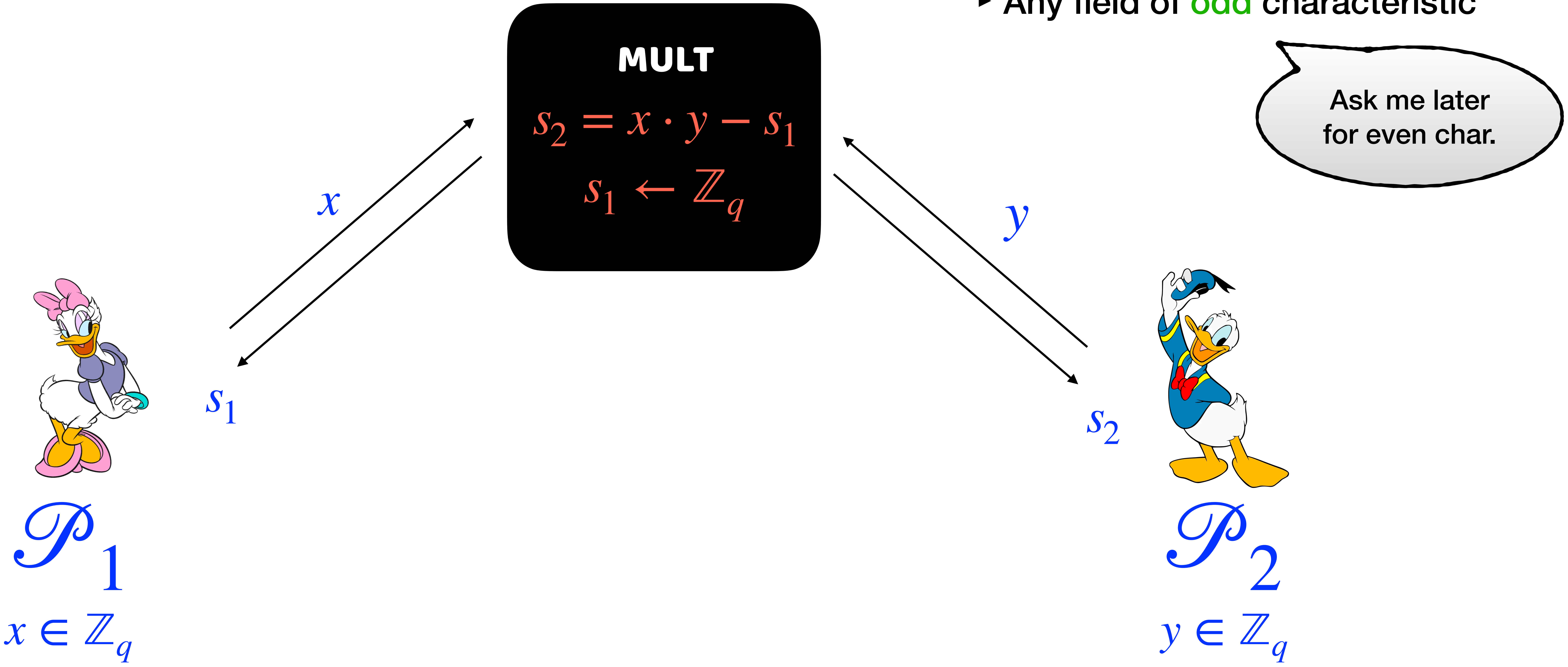
- q is a large prime



Problem Statement (Two-Party Multiplication)

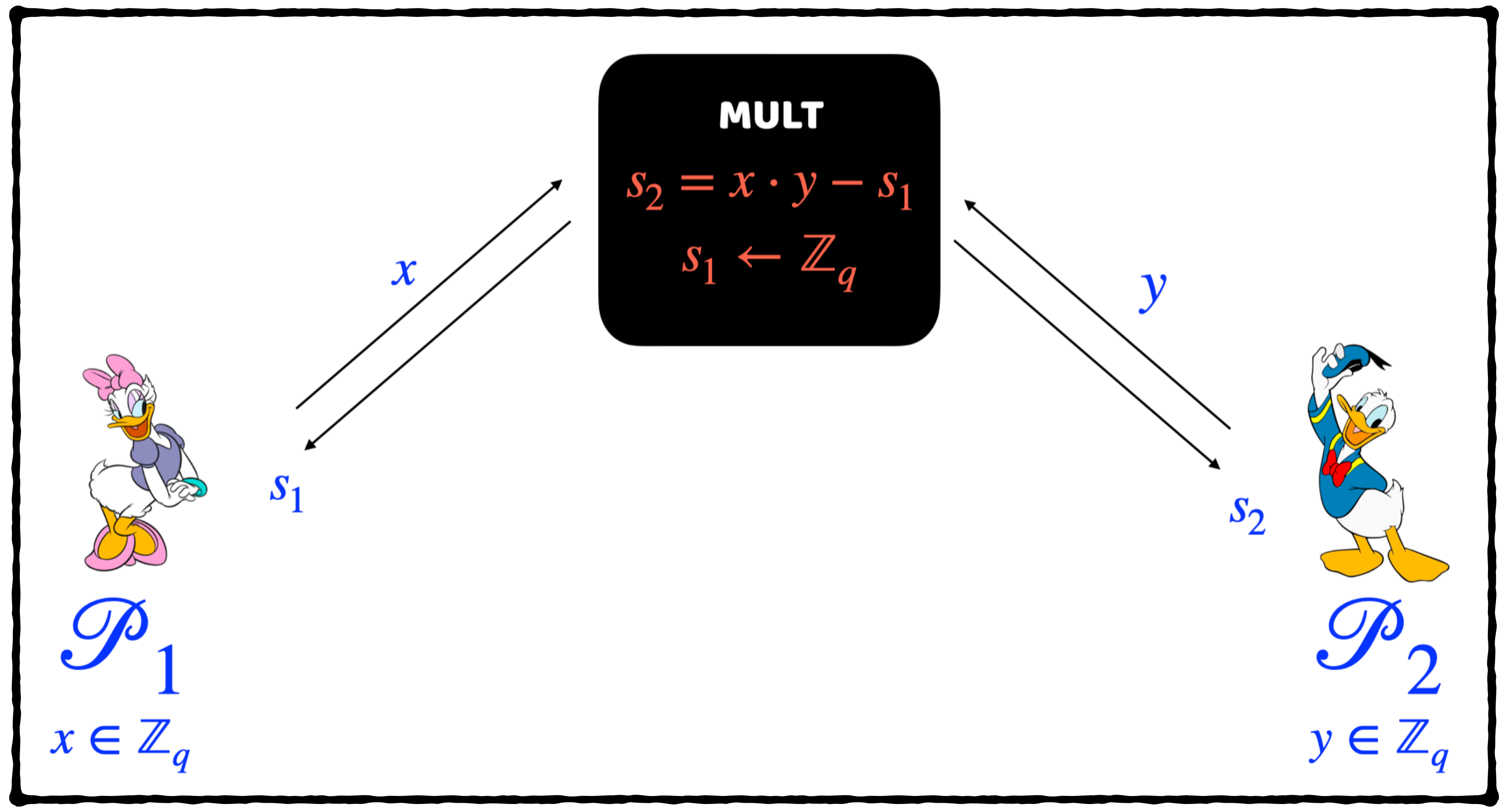
$\mathsf{Mult}(x, y) \mapsto (s_1, s_2 = x \cdot y - s_1)$

- q is a large prime
- Any field of odd characteristic



Motivation

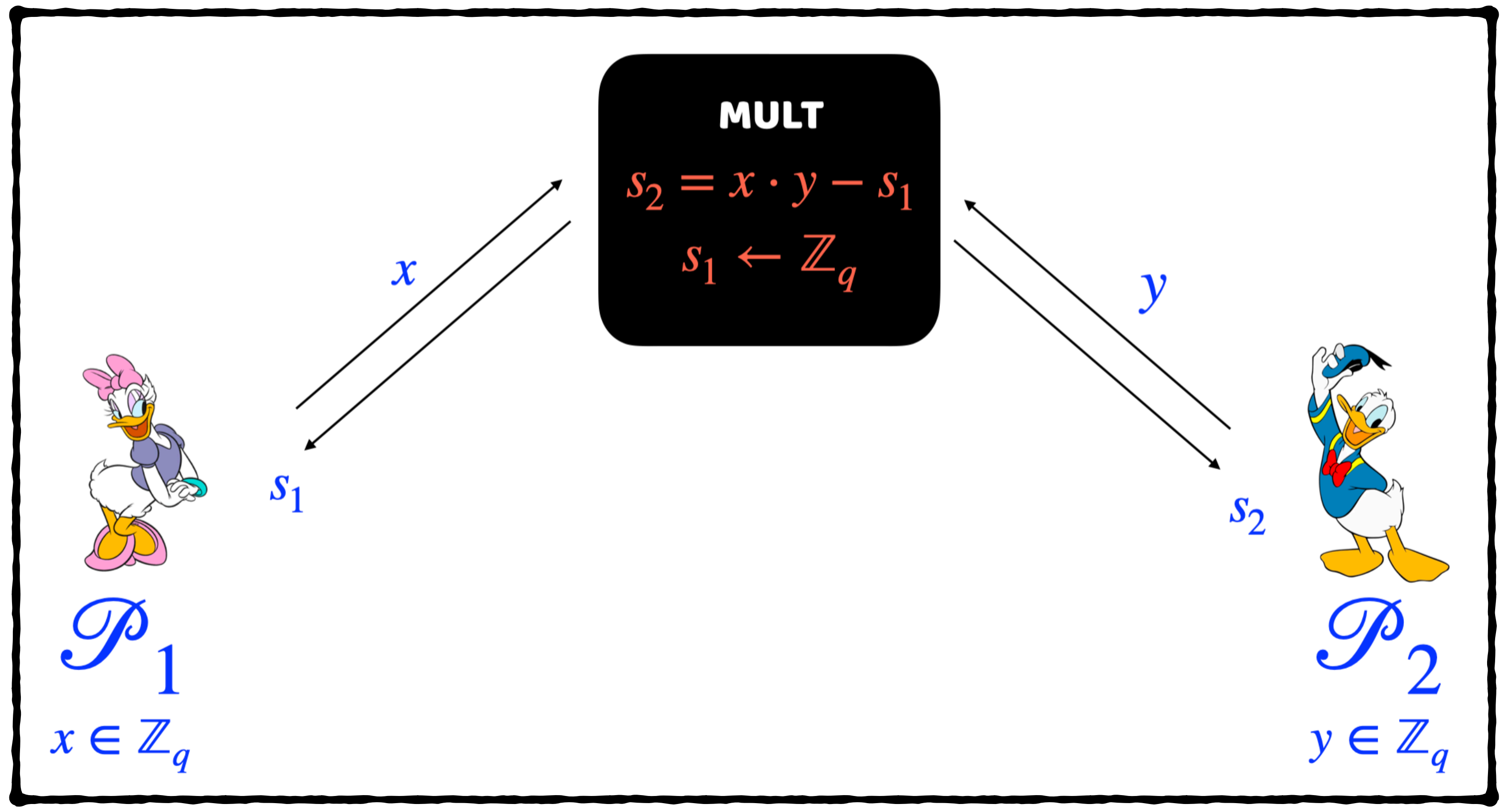
Why Multiplication Protocols?



Motivation

Why Multiplication Protocols?

- Building block in *arithmetic* MPC

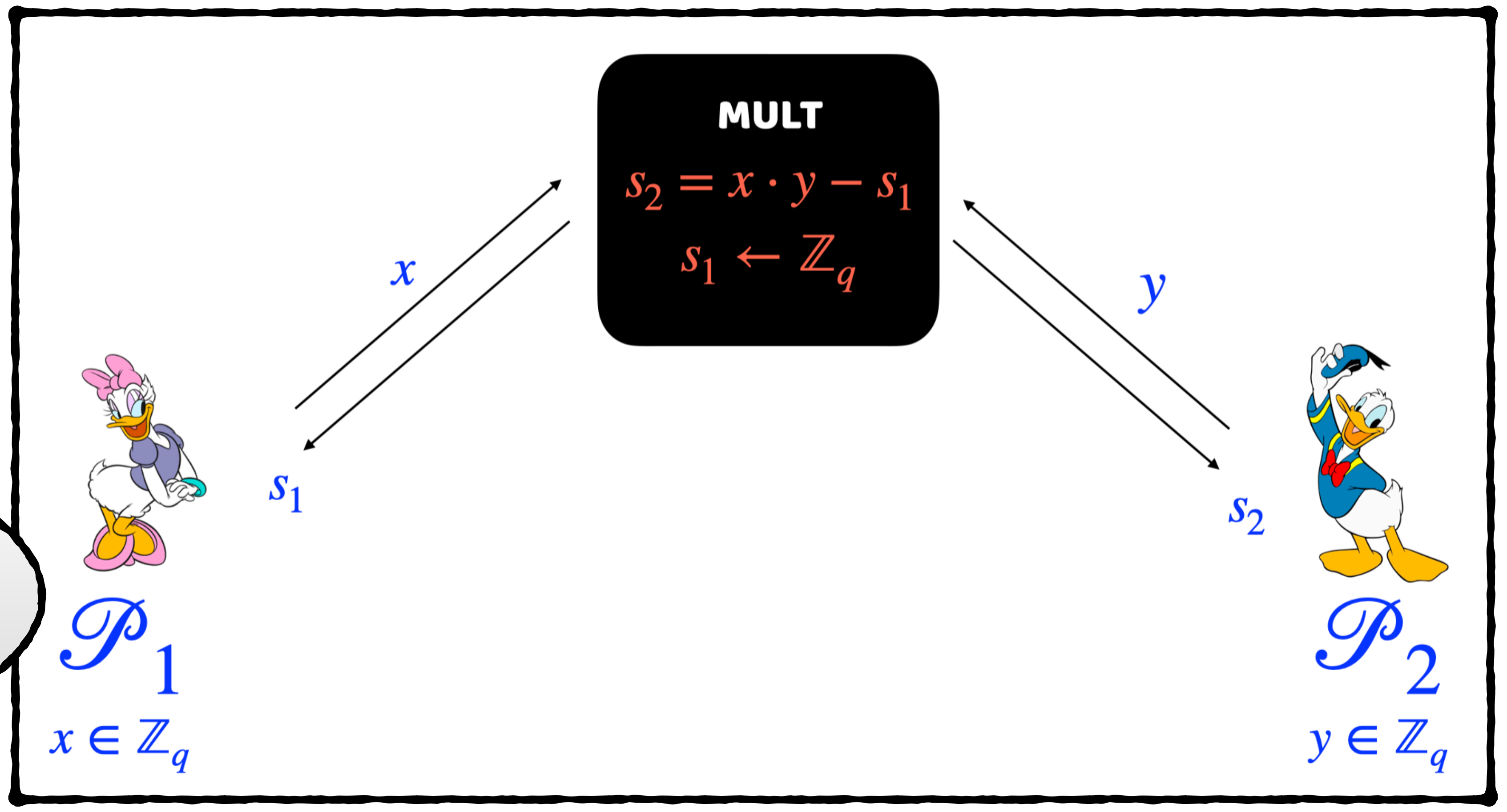


Motivation

Why Multiplication Protocols?

Analogous to OT in *Boolean* MPC

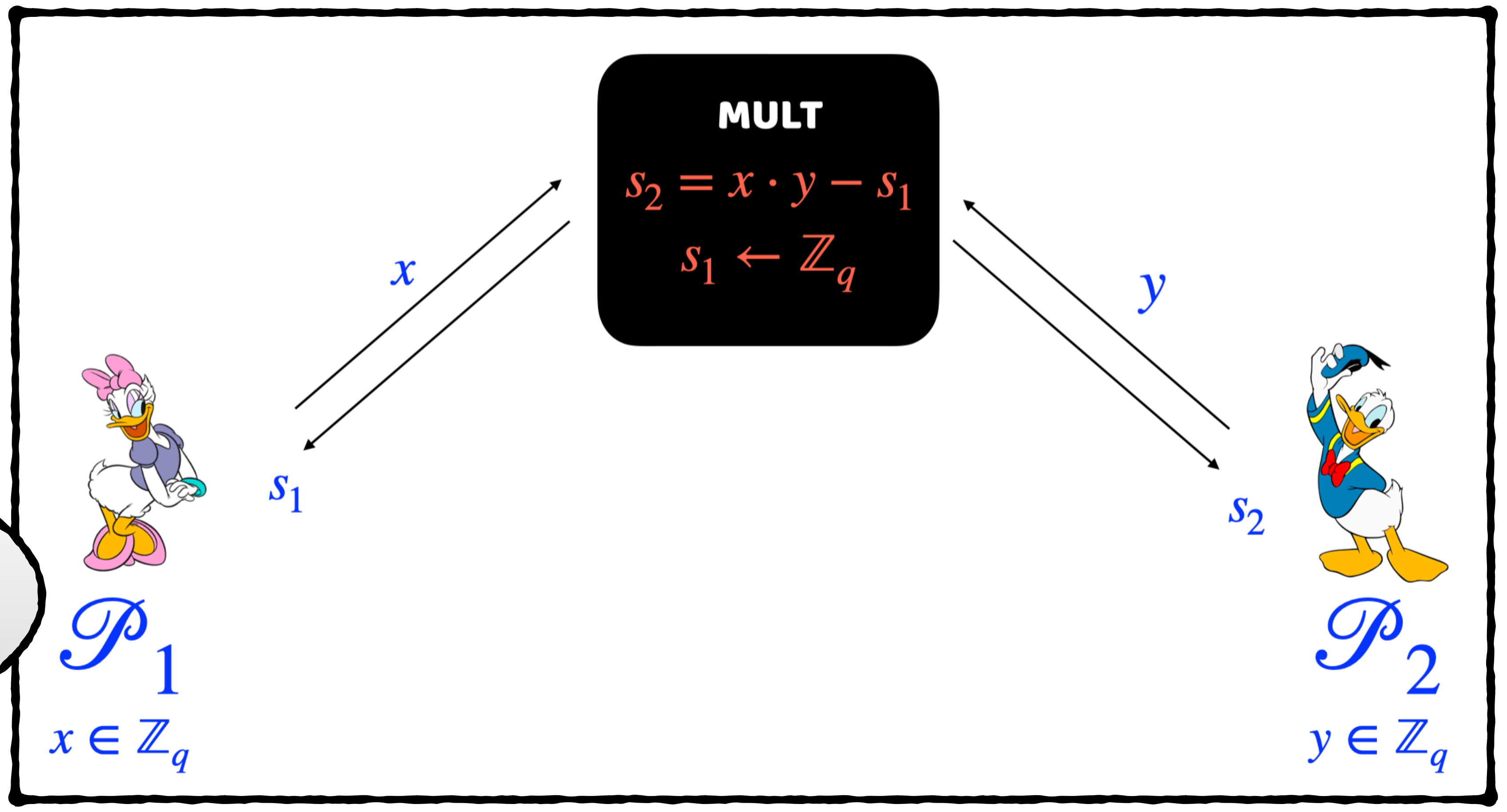
- Building block in *arithmetic* MPC



Motivation

Why Multiplication Protocols?

Analogous to OT in *Boolean* MPC



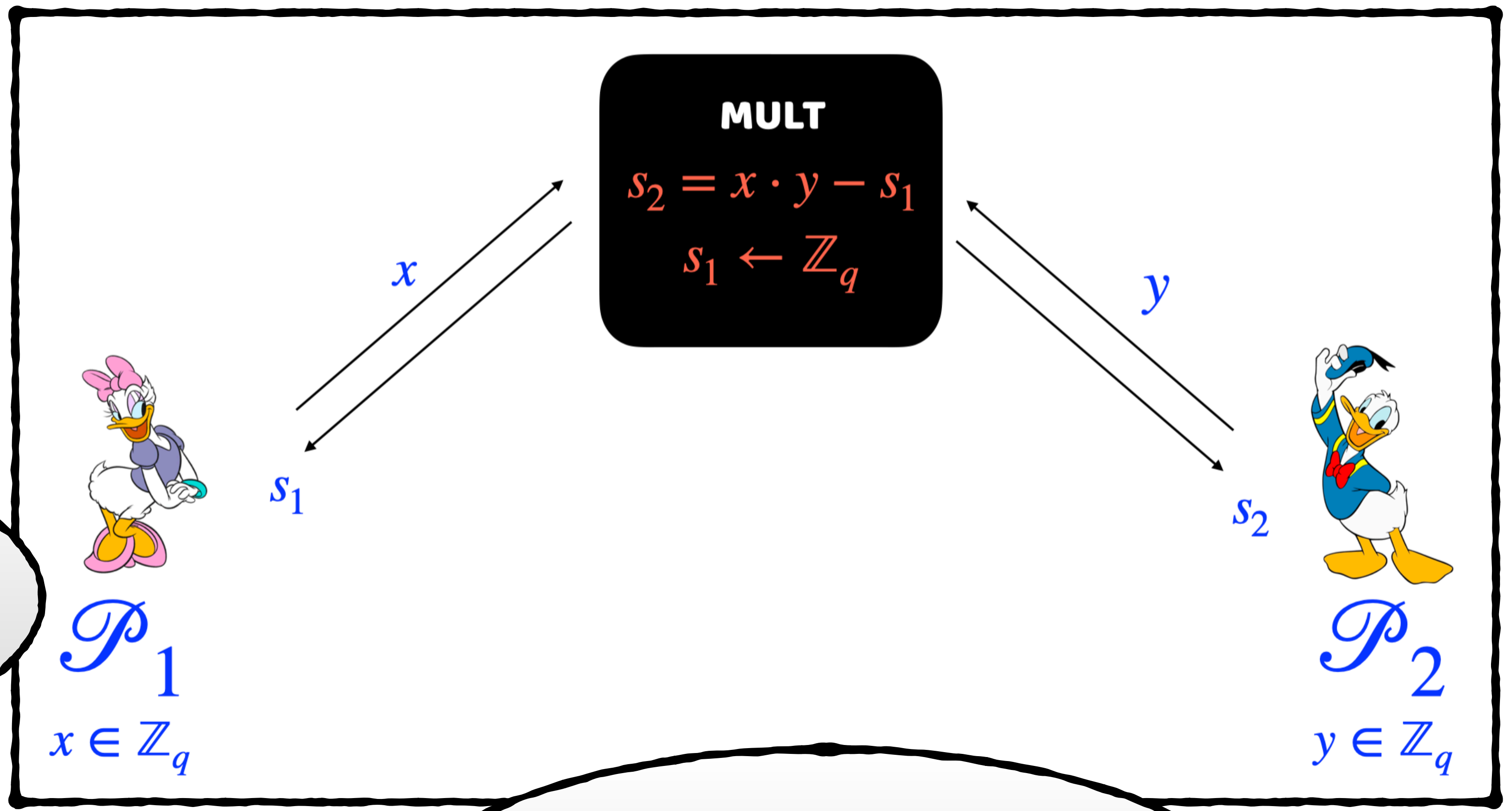
- Building block in *arithmetic* MPC
- E.g., for generating tuples in preprocessing model
  - *Beaver triplets*: 2-out-of-2 shares of $(a, b, c = a \cdot b)$
  - *Authenticated triplets* (SPDZ)

Motivation

Why Multiplication Protocols?

Analogous to OT in *Boolean* MPC

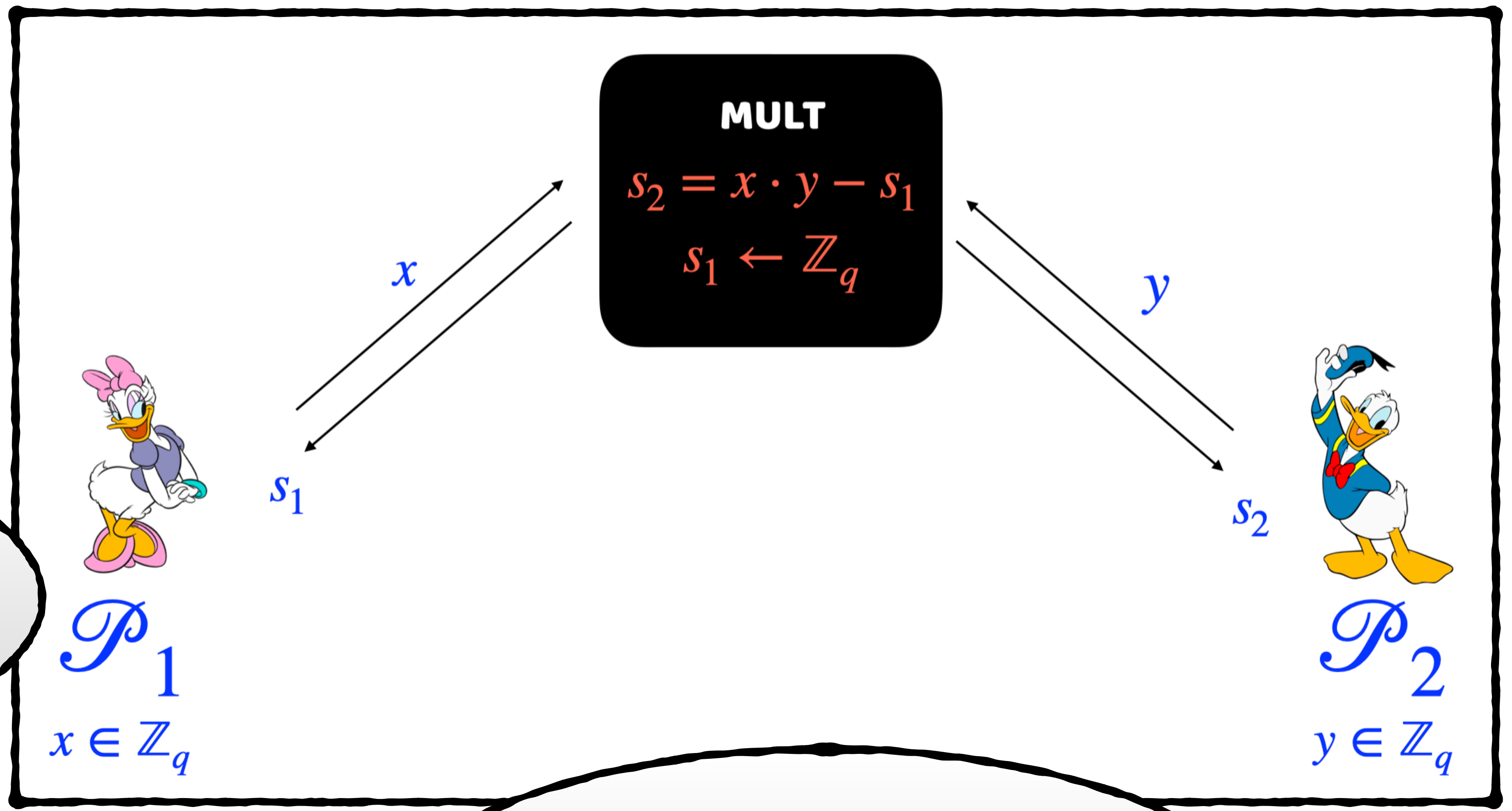
- Building block in *arithmetic* MPC
- E.g., for generating tuples in preprocessing model
  - *Beaver triplets*: 2-out-of-2 shares of $(a, b, c = a \cdot b)$
  - *Authenticated triplets* (SPDZ)



These triplets facilitate *fast* info-theoretic MPC

Motivation

Why Multiplication Protocols?



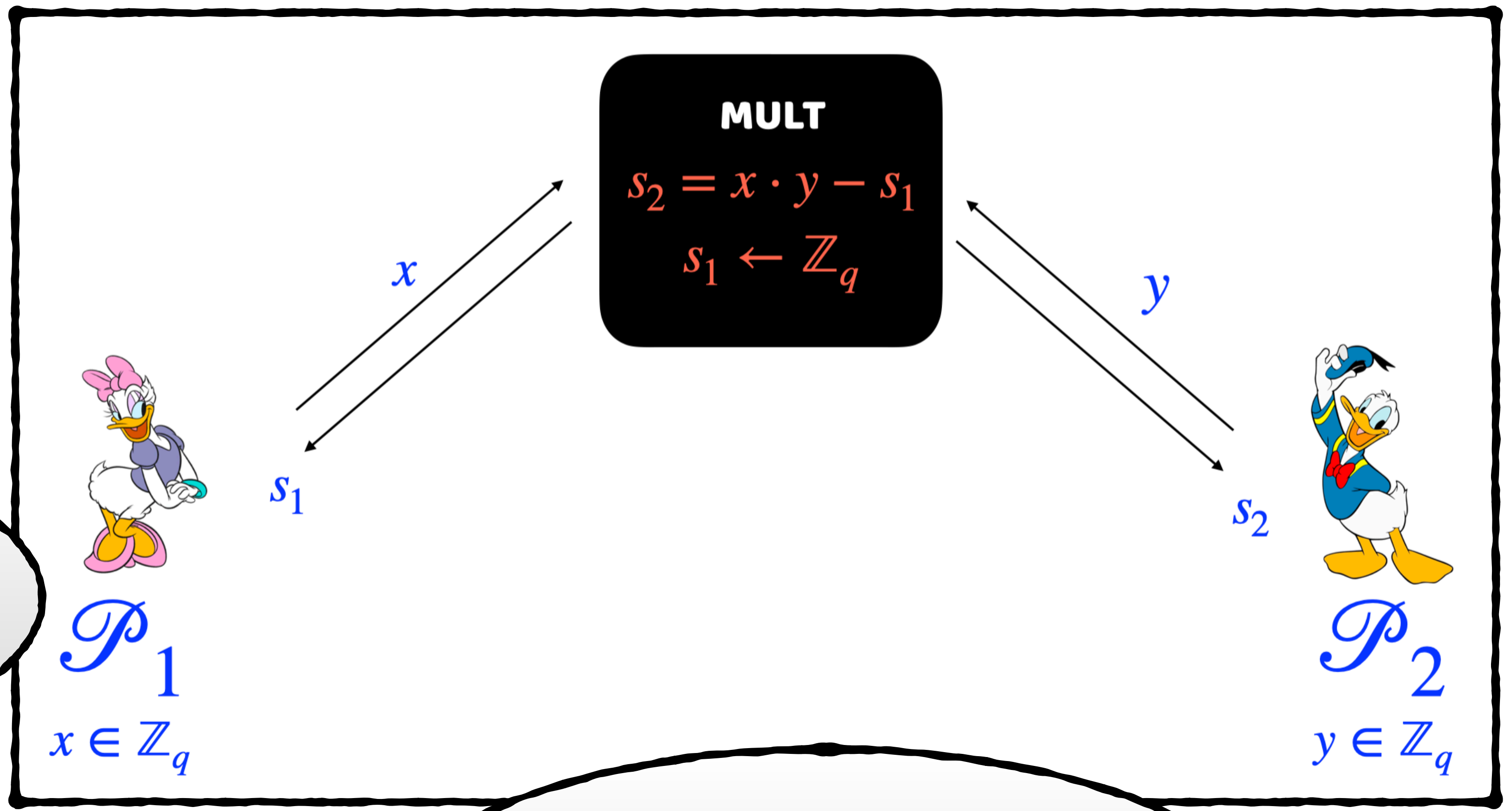
Analogous to OT in *Boolean* MPC

These triplets facilitate *fast* info-theoretic MPC

- Building block in *arithmetic* MPC
- E.g., for generating tuples in preprocessing model
  - *Beaver triplets*: 2-out-of-2 shares of $(a, b, c = a \cdot b)$
  - *Authenticated triplets* (SPDZ)
- Realizing threshold-*ECDSA*
  - Applications to *distributed EC* cryptography (e.g. SNARKS)

Motivation

Why Multiplication Protocols?



Analogous to OT in *Boolean* MPC

- Building block in *arithmetic* MPC
- E.g., for generating tuples in preprocessing model
  - *Beaver triplets*: 2-out-of-2 shares of $(a, b, c = a \cdot b)$
  - *Authenticated triplets* (SPDZ)

These triplets facilitate *fast* info-theoretic MPC

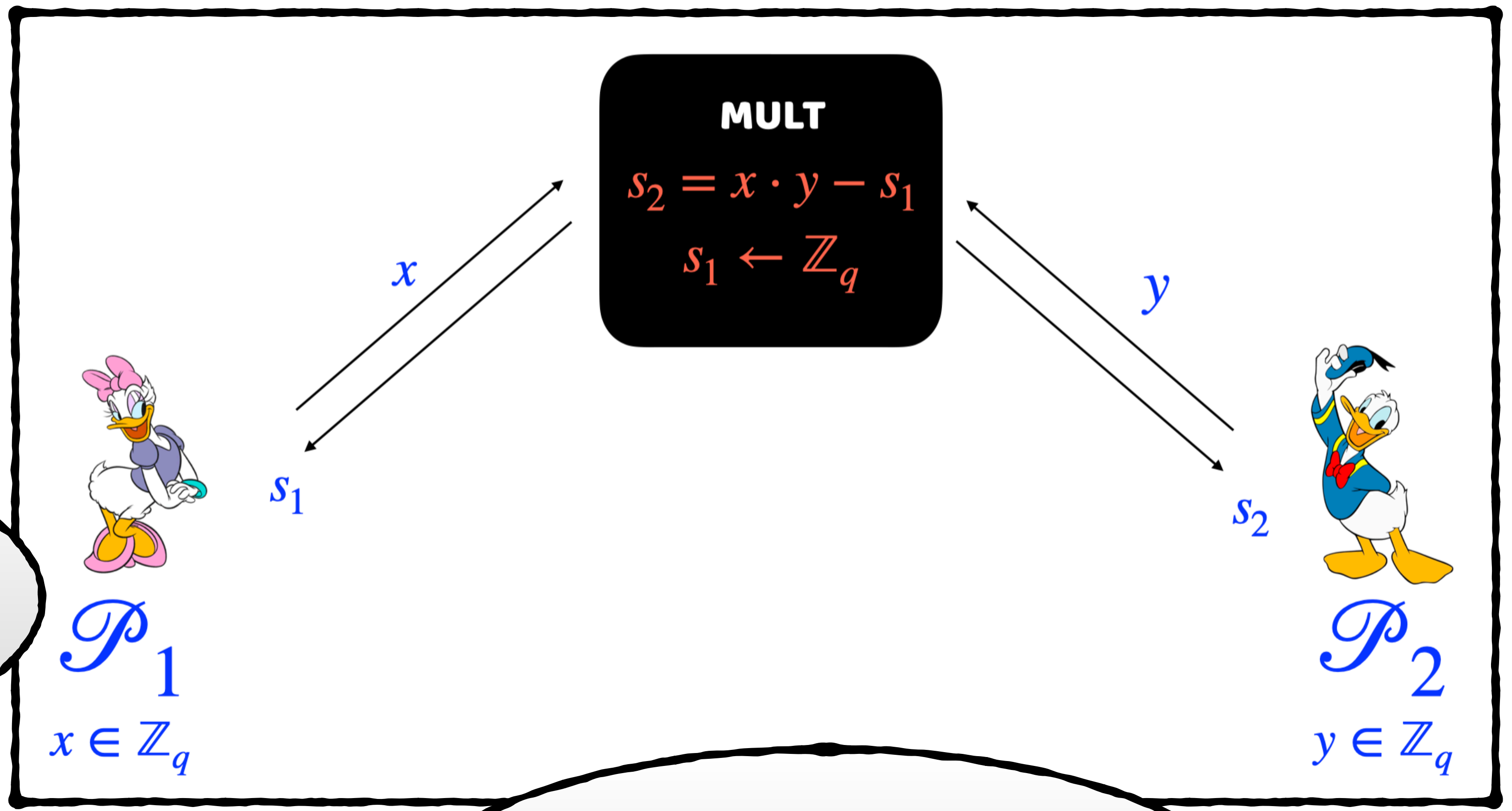
- Realizing threshold-*ECDSA*
  - Applications to *distributed EC* cryptography (e.g. SNARKS)

Use MPC to “thresholdize”

Motivation

Why Multiplication Protocols?

Analogous to OT in *Boolean* MPC



These triplets facilitate *fast* info-theoretic MPC

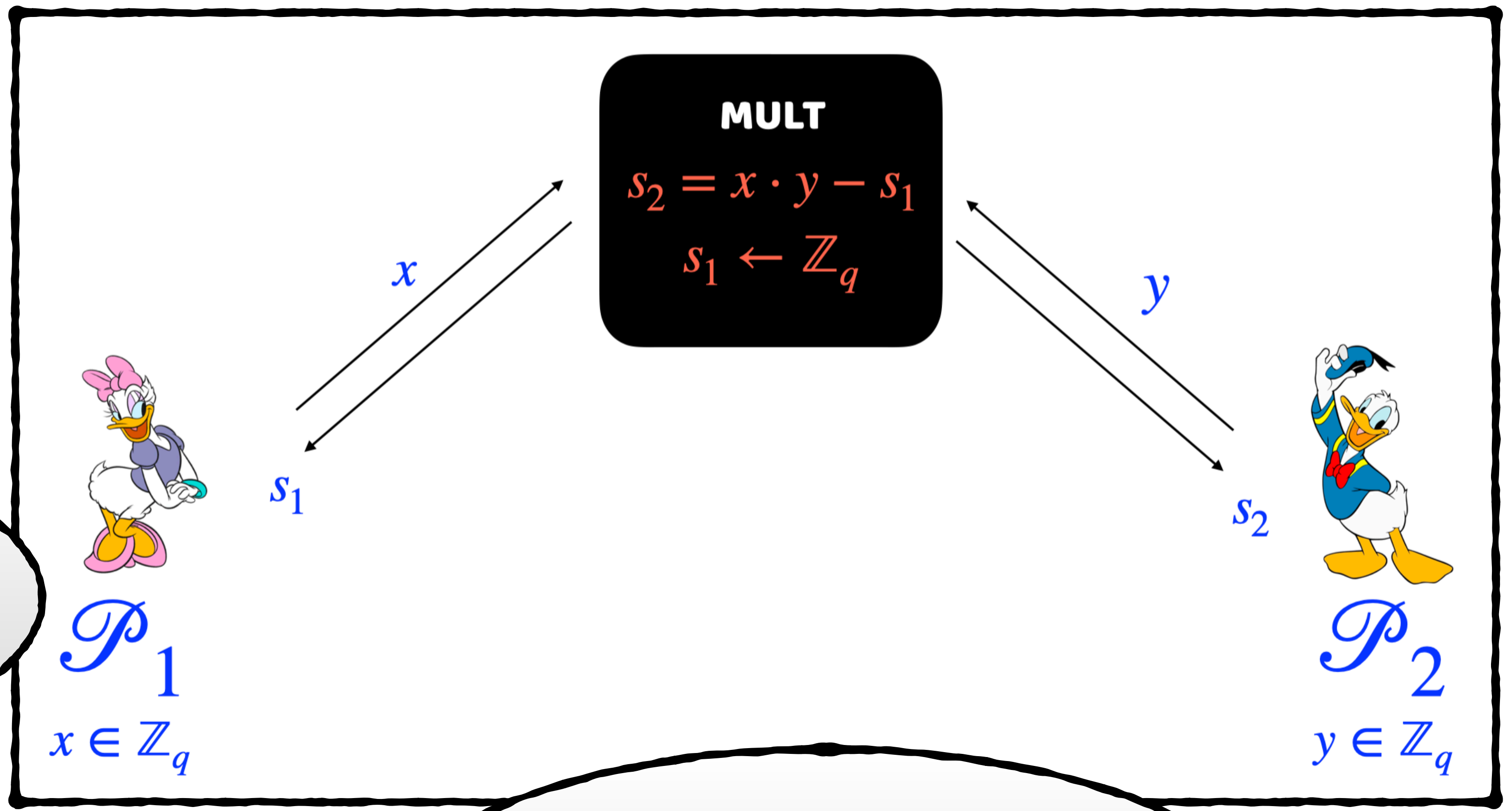
- Building block in *arithmetic* MPC
- E.g., for generating tuples in preprocessing model
  - *Beaver triplets*: 2-out-of-2 shares of $(a, b, c = a \cdot b)$
  - *Authenticated triplets* (SPDZ)
- Realizing threshold-*ECDSA* $\implies$ Shallow arithmetic circuit over $\mathbb{Z}_q$
  - Applications to *distributed EC* cryptography (e.g. SNARKS)

Use MPC to "thresholdize"

Motivation

Why Multiplication Protocols?

Analogous to OT in *Boolean* MPC



- Building block in *arithmetic* MPC
- E.g., for generating tuples in preprocessing model
  - *Beaver triplets*: 2-out-of-2 shares of $(a, b, c = a \cdot b)$
  - *Authenticated triplets* (SPDZ)

These triplets facilitate *fast* info-theoretic MPC

- Realizing threshold-*ECDSA* $\Longrightarrow$ Shallow arithmetic circuit over $\mathbb{Z}_q$
  - Applications to *distributed EC* cryptography (e.g. SNARKS)

Use MPC to "thresholdize"

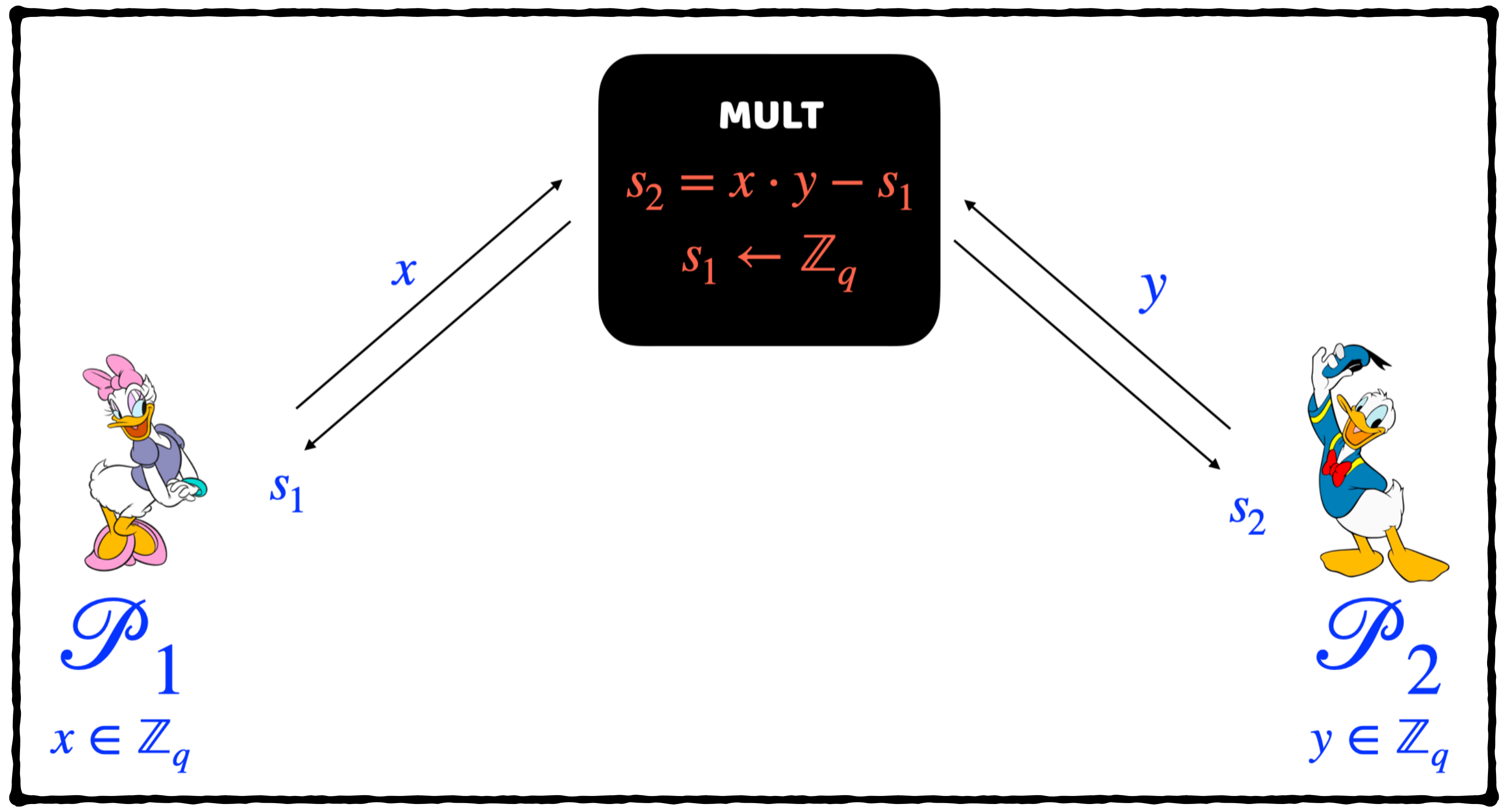
- Mult is "almost" the same functionality as *OLE* (Oblivious Linear Evaluation)

  $\text{OLE}(x, (y, \sigma)) \mapsto (x \cdot y + \sigma, \perp)$

Background

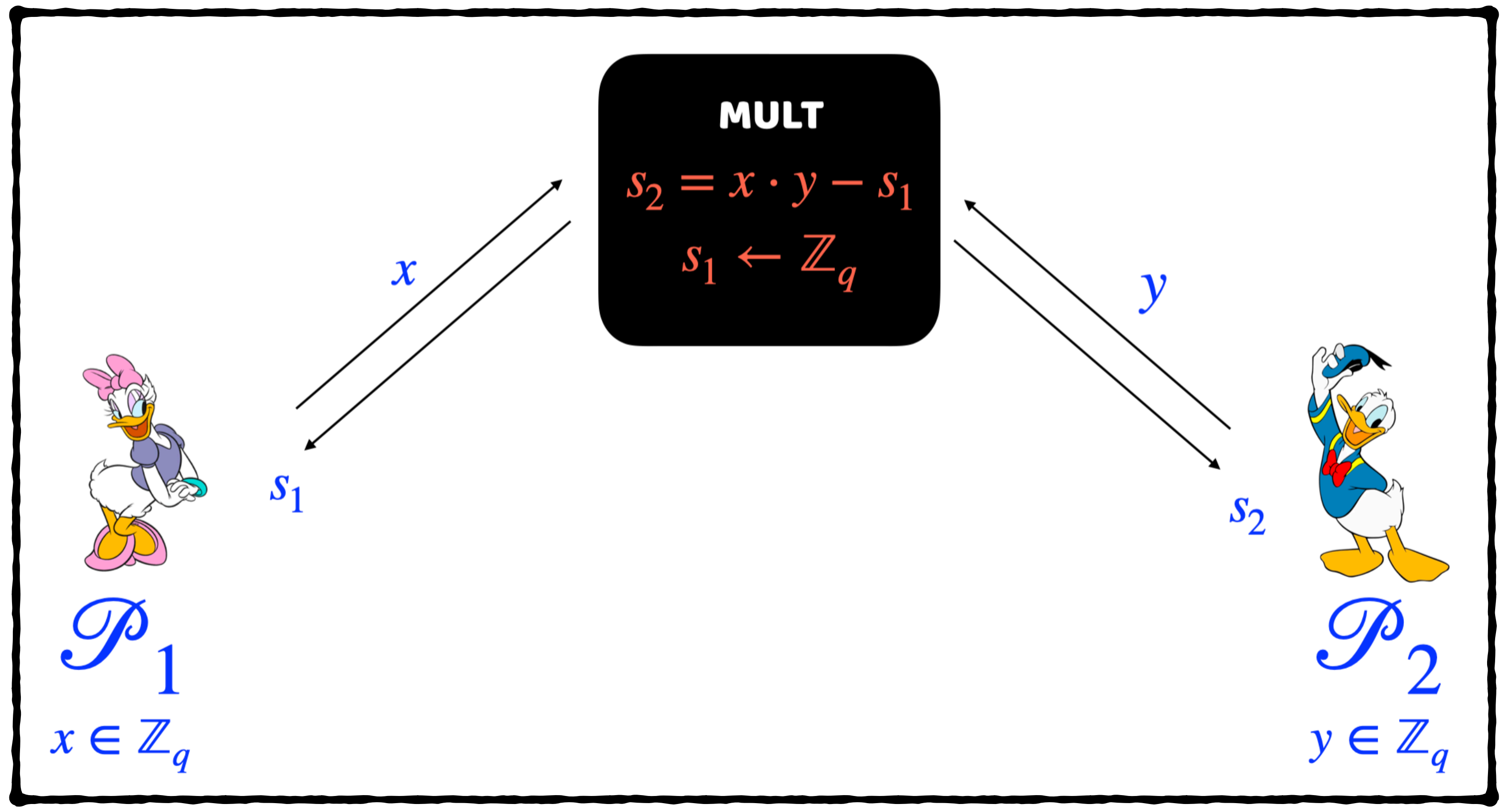
Background

Known Constructions/Paradigms



Background

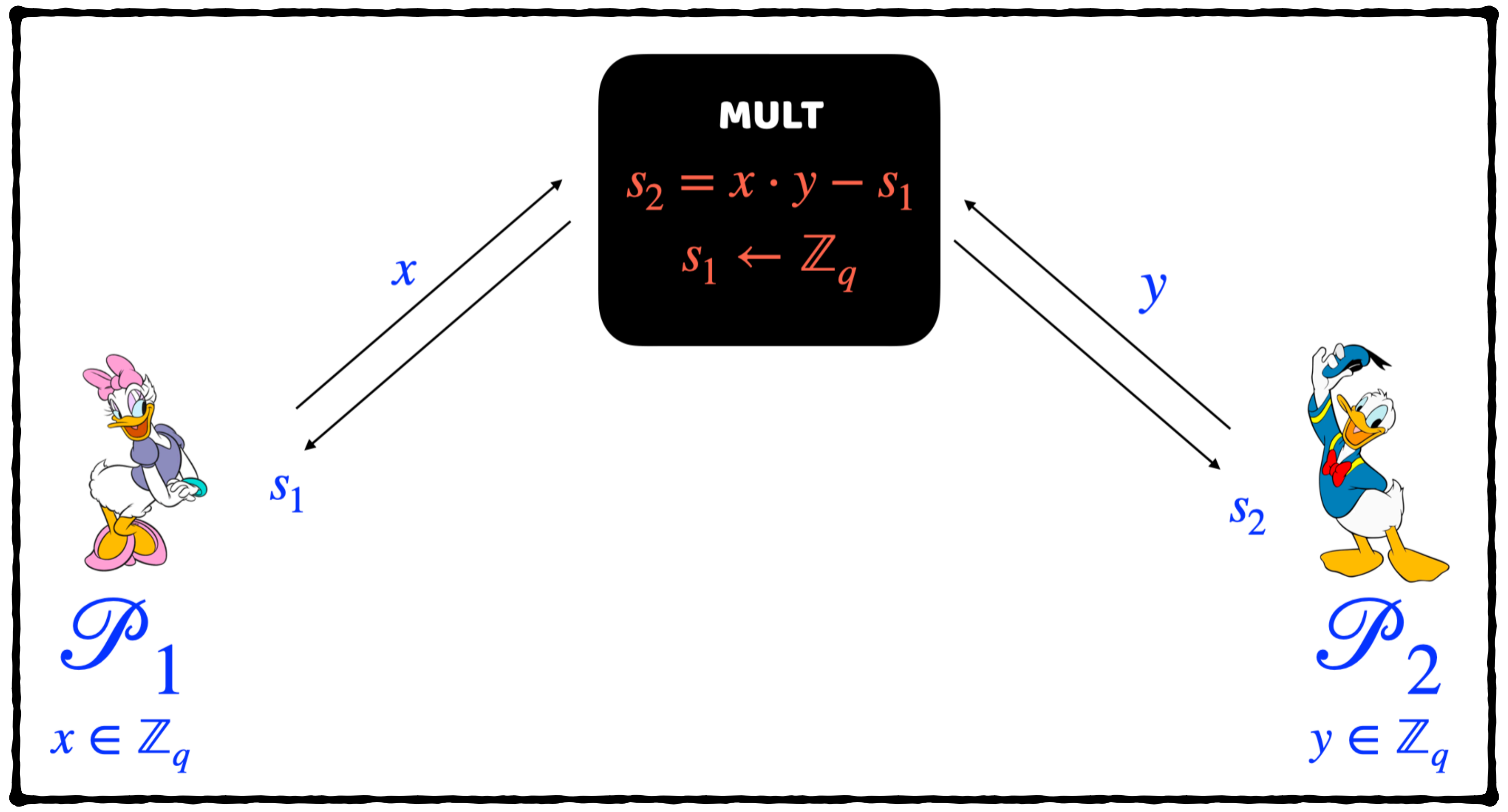
Known Constructions/Paradigms



- Homomorphic Encryption, e.g., *Paillier*
  - 👍 Light Communication
  - 👎 Heavy Computation
  - 👎 Stronger Assumptions

Background

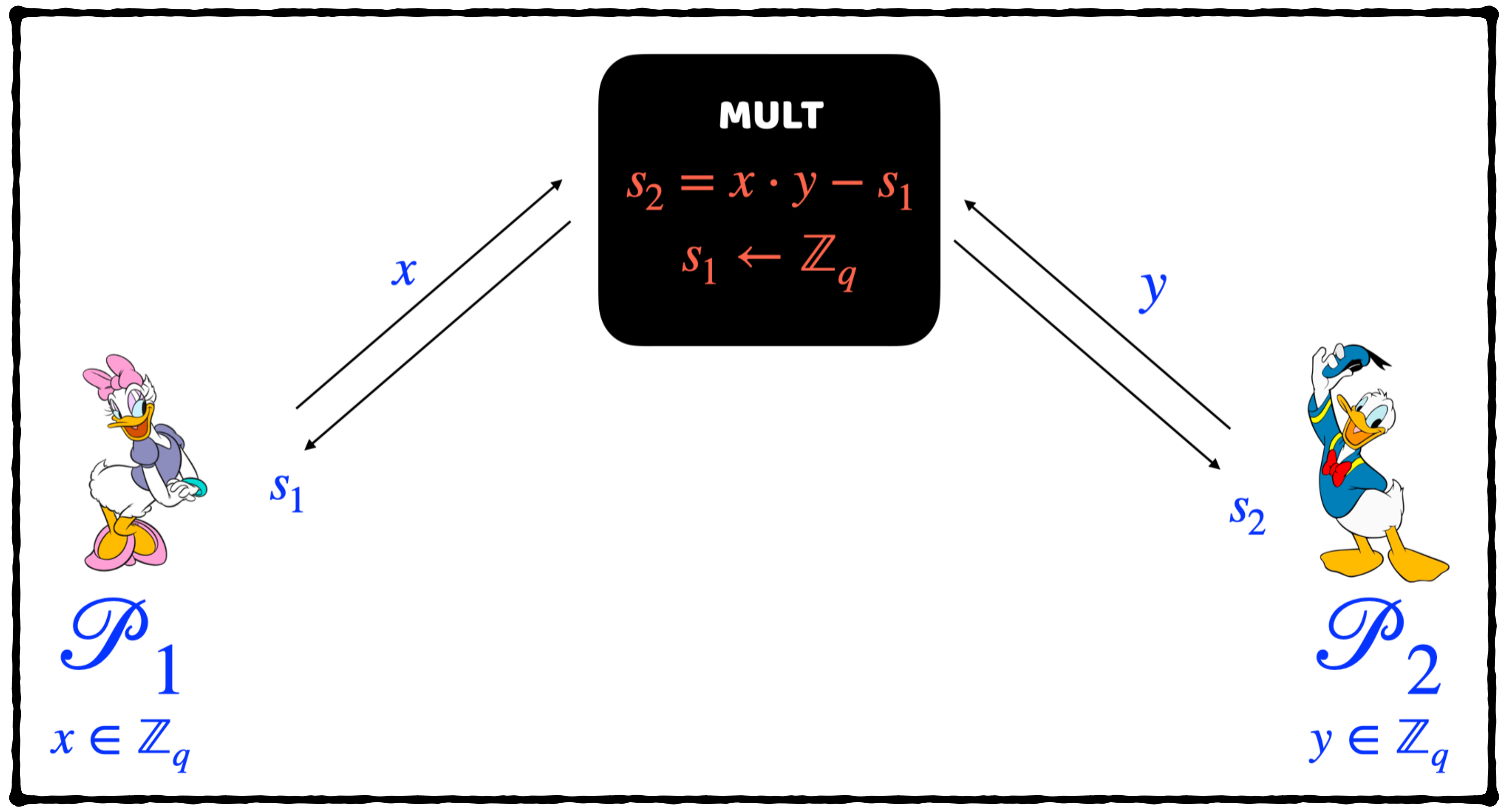
Known Constructions/Paradigms



- Homomorphic Encryption, e.g., *Paillier*
  - 👍 Light Communication
  - 👎 Heavy Computation
  - 👎 Stronger Assumptions
- OT-Based
  - 👎 Heavy Communication
  - 👍 Light Computation
  - 👍 Minimal Assumptions

Background

Known Constructions/Paradigms

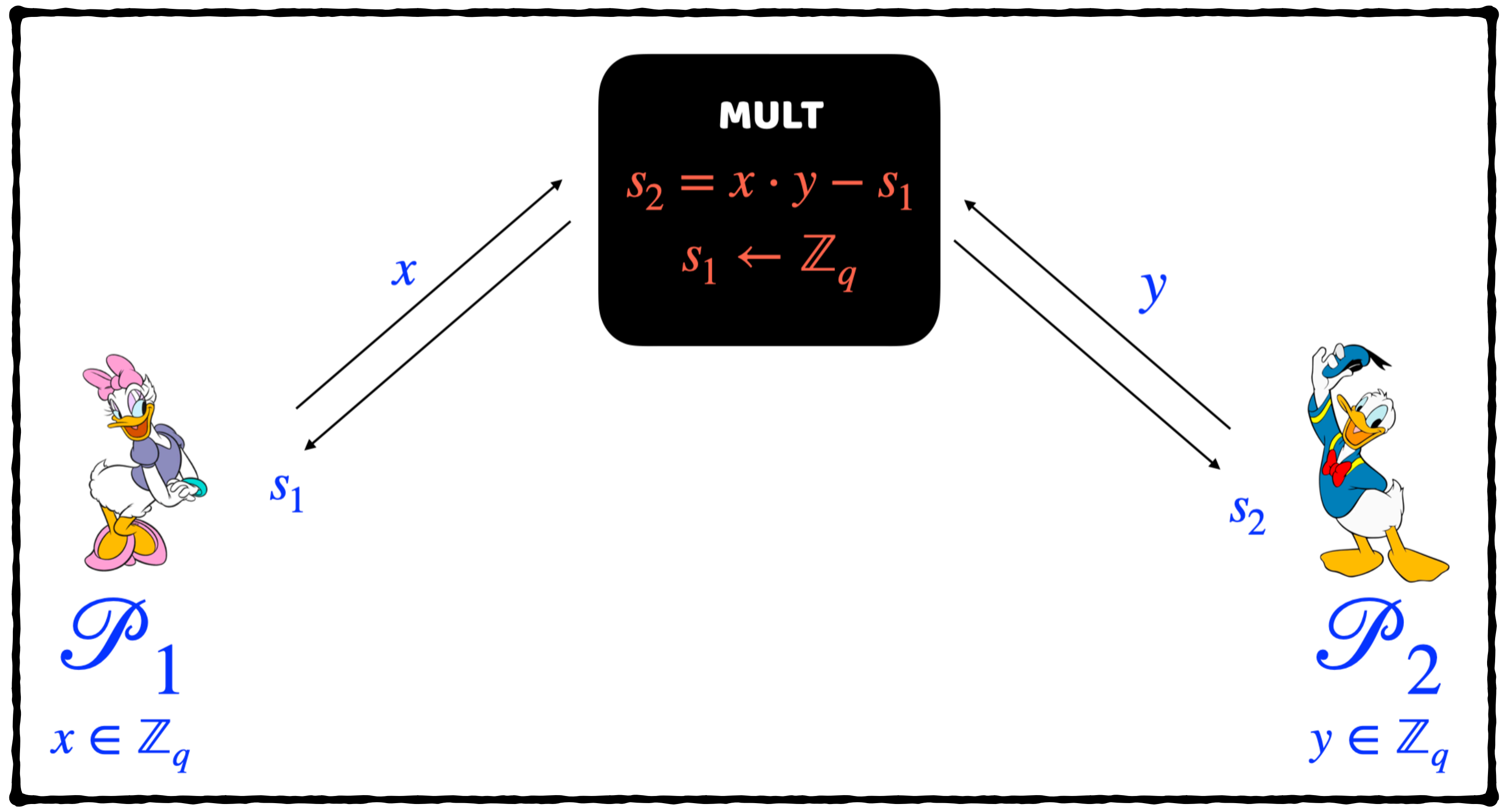


- Homomorphic Encryption, e.g., *Paillier*
  - 👍 Light Communication
  - 👎 Heavy Computation
  - 👎 Stronger Assumptions
- OT-Based
  - 👎 Heavy Communication
  - 👍 Light Computation
  - 👍 Minimal Assumptions

Tens to *hundreds* of KiB
for multiplying 32B values!

Background

Known Constructions/Paradigms



- Homomorphic Encryption, e.g., *Paillier*
  - Light Communication
  - Heavy Computation
  - Stronger Assumptions
- OT-Based
  - Heavy Communication
  - Light Computation
  - Minimal Assumptions

Tens to *hundreds* of KiB
for multiplying 32B values!

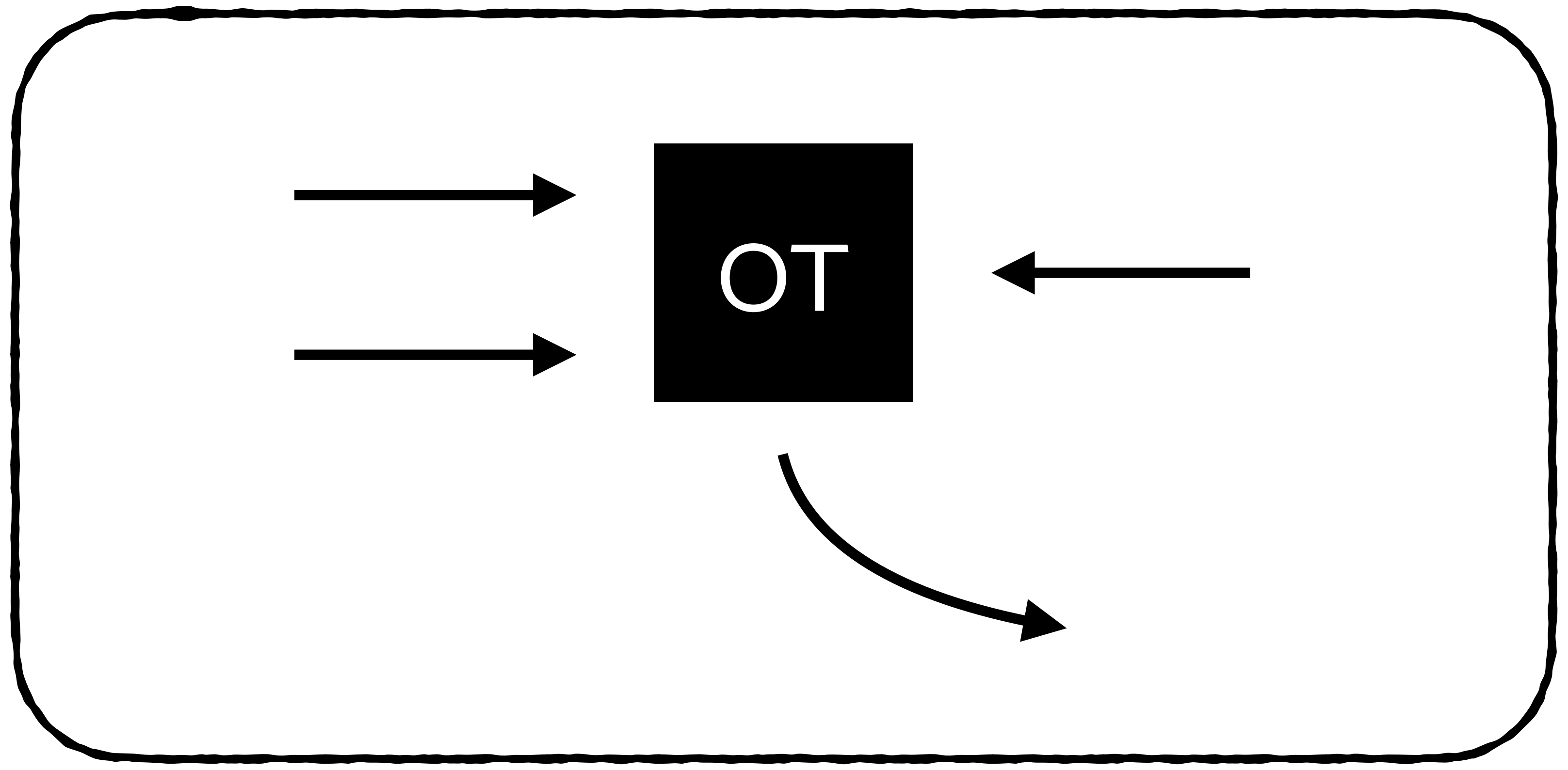
This work: New OT-Based Multiplication protocol!

Background

OT-Hybrid Model

Background

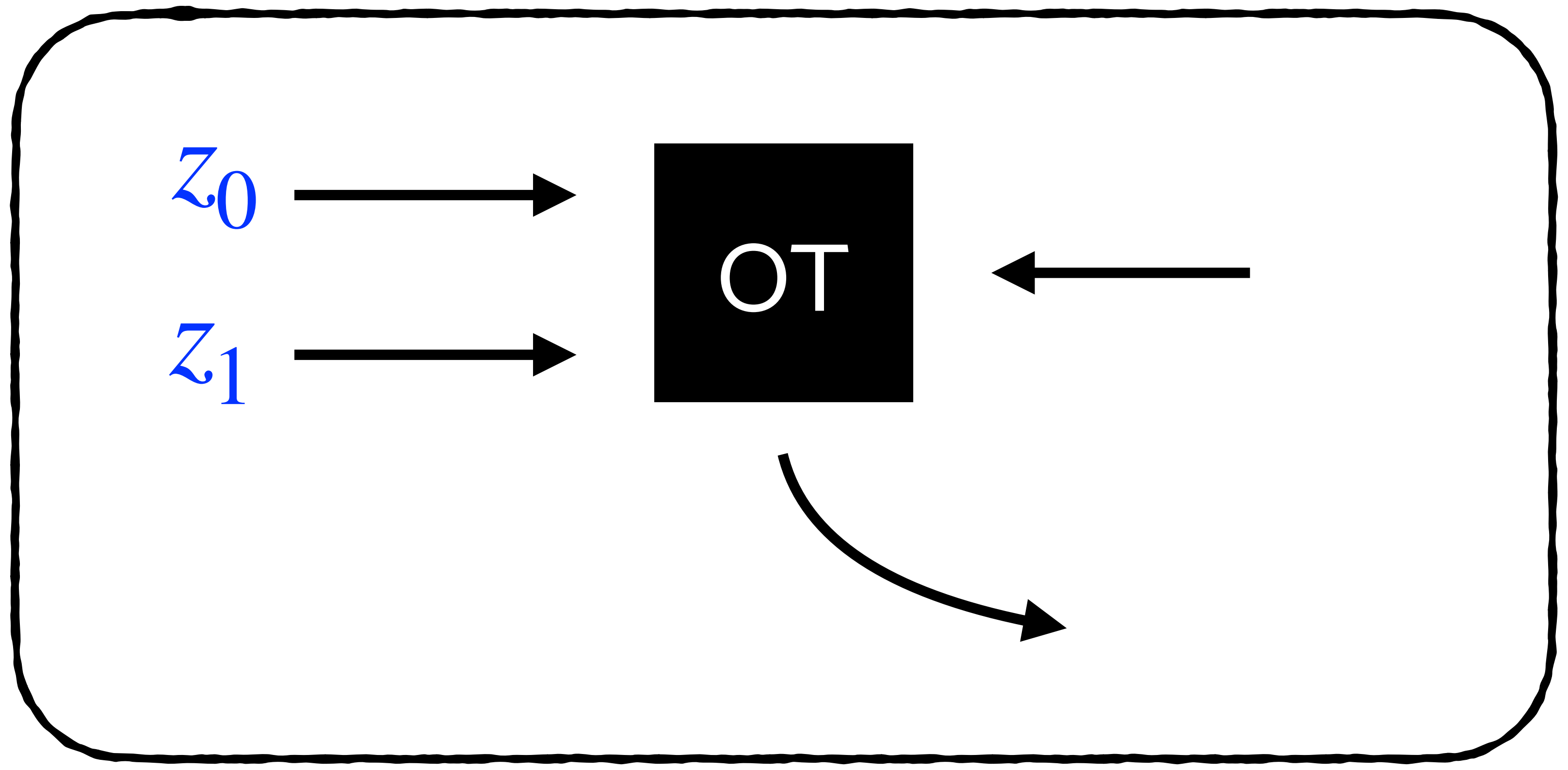
OT-Hybrid Model



- Augment the plain model with an oracle to $\mathsf{OT}((z_0, z_1), \alpha) \mapsto (\perp, z_\alpha)$

Background

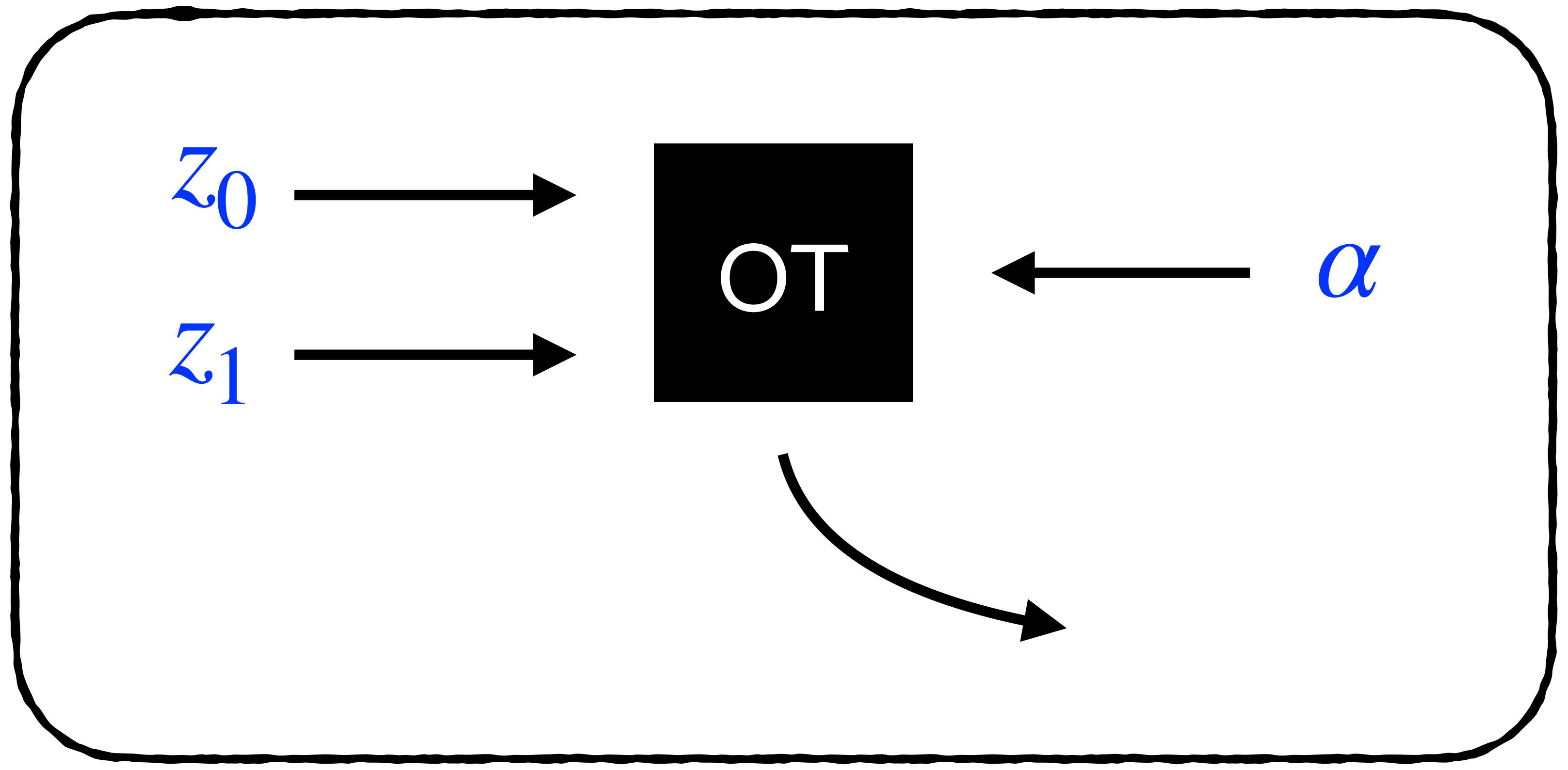
OT-Hybrid Model



- Augment the plain model with an oracle to OT$((z_0, z_1), \alpha) \mapsto (\ \perp\ , z_\alpha\ )$

Background

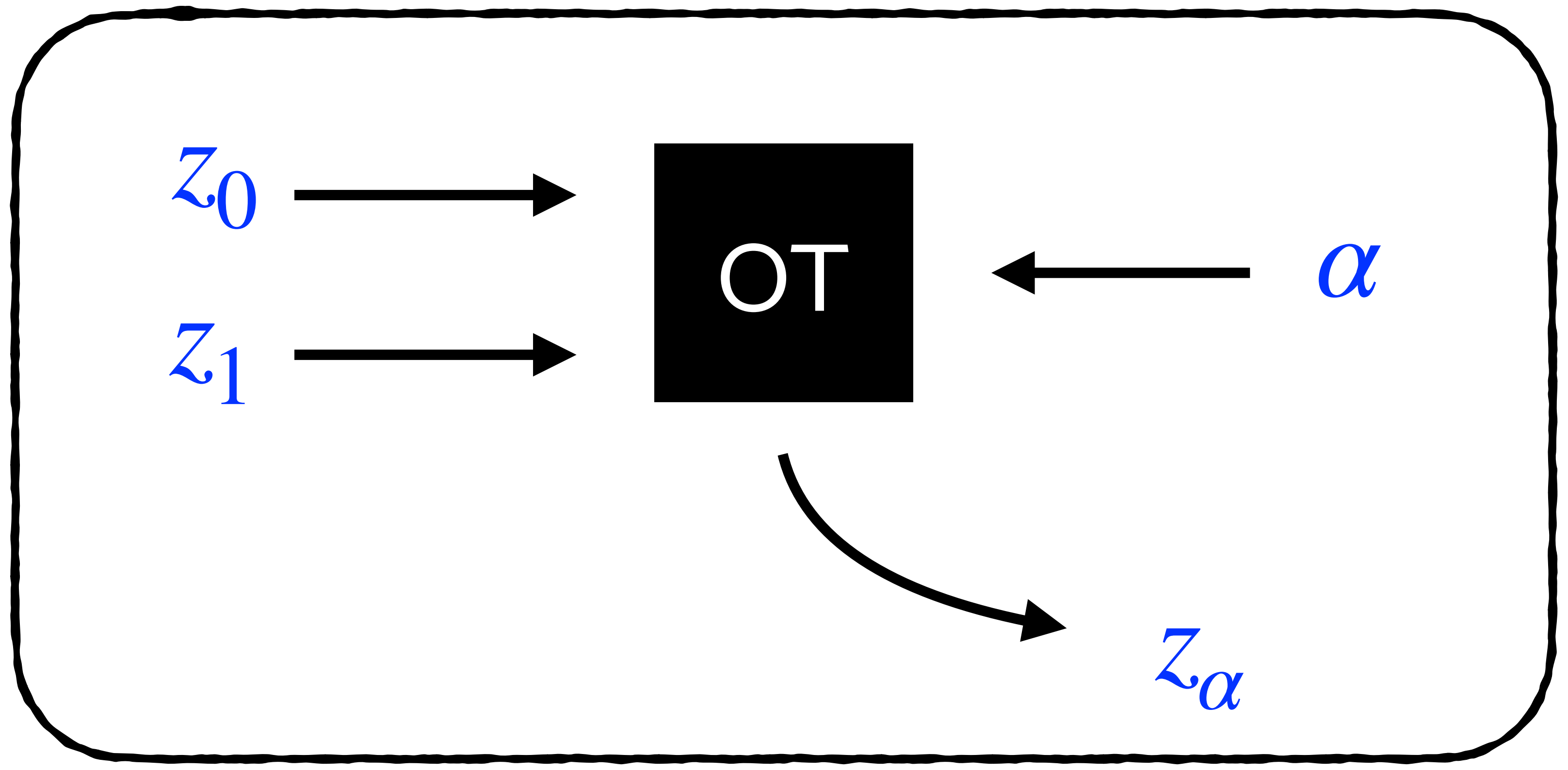
OT-Hybrid Model



- Augment the plain model with an oracle to $\mathrm{OT}((z_0, z_1), \alpha) \mapsto (\perp, z_\alpha)$

Background

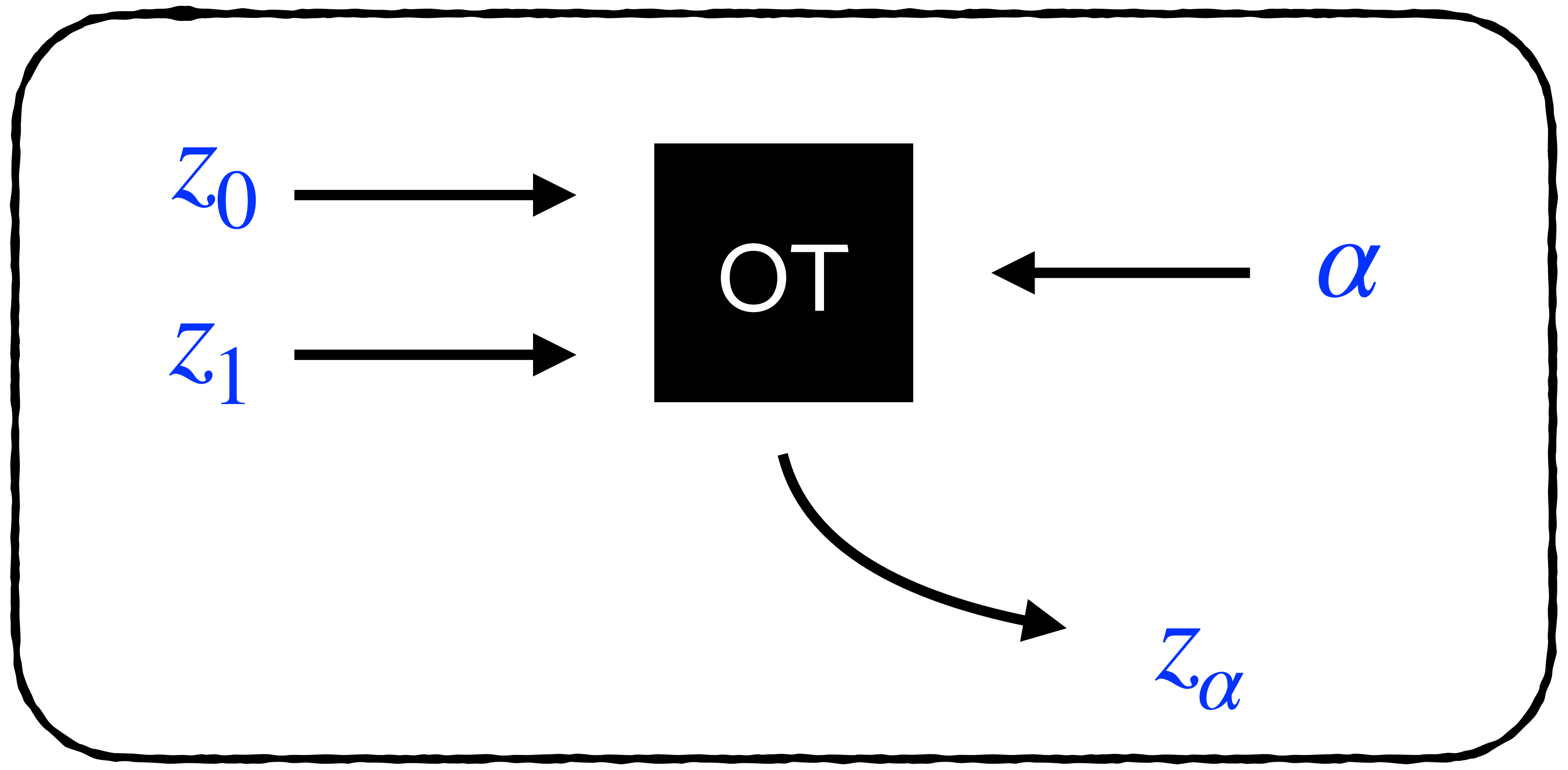
OT-Hybrid Model



- Augment the plain model with an oracle to $\mathsf{OT}((z_0, z_1), \alpha) \mapsto (\perp, z_\alpha)$

Background

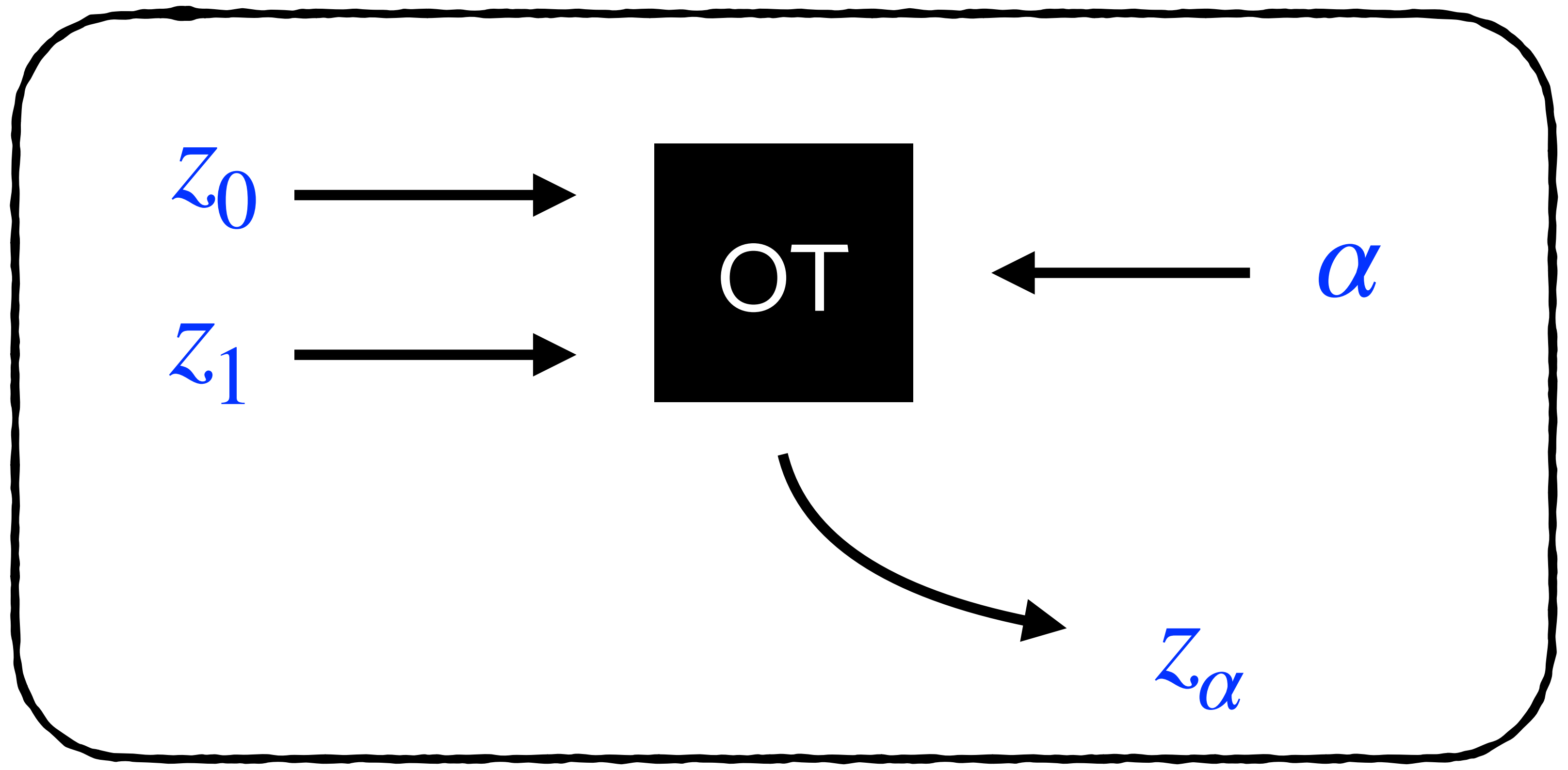
OT-Hybrid Model



- Augment the plain model with an oracle to $\mathsf{OT}((z_0, z_1), \alpha) \mapsto (\perp, z_\alpha)$
- Hybrid Protocol $\implies$ Real Protocol (via composition [Can01])

Background

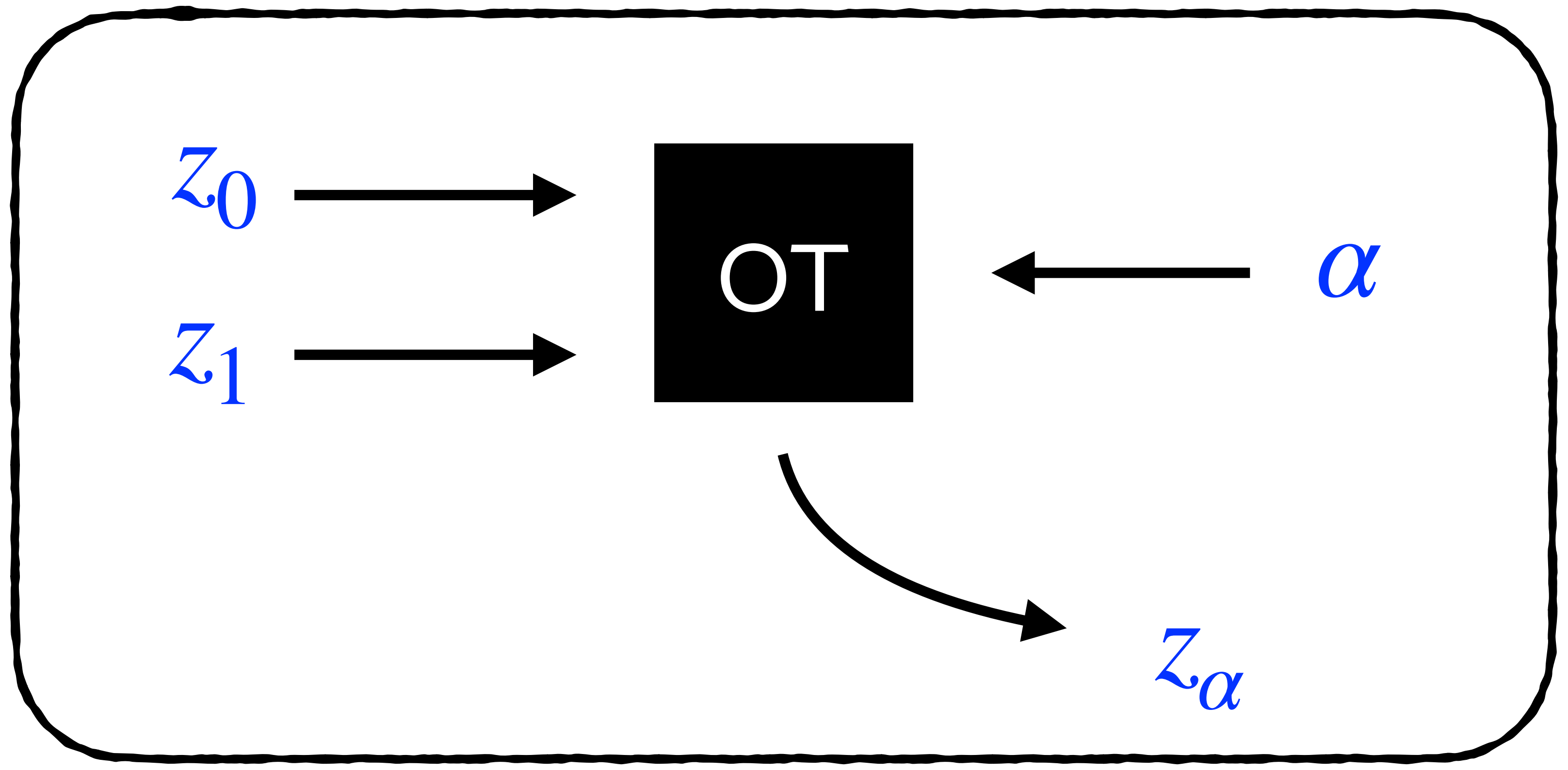
OT-Hybrid Model



- Augment the plain model with an oracle to $\mathsf{OT}((z_0, z_1), \alpha) \mapsto (\ \perp\ , z_\alpha\ )$
- Hybrid Protocol $\implies$ Real Protocol (via composition [Can01])
- Costs Breakdown:

Background

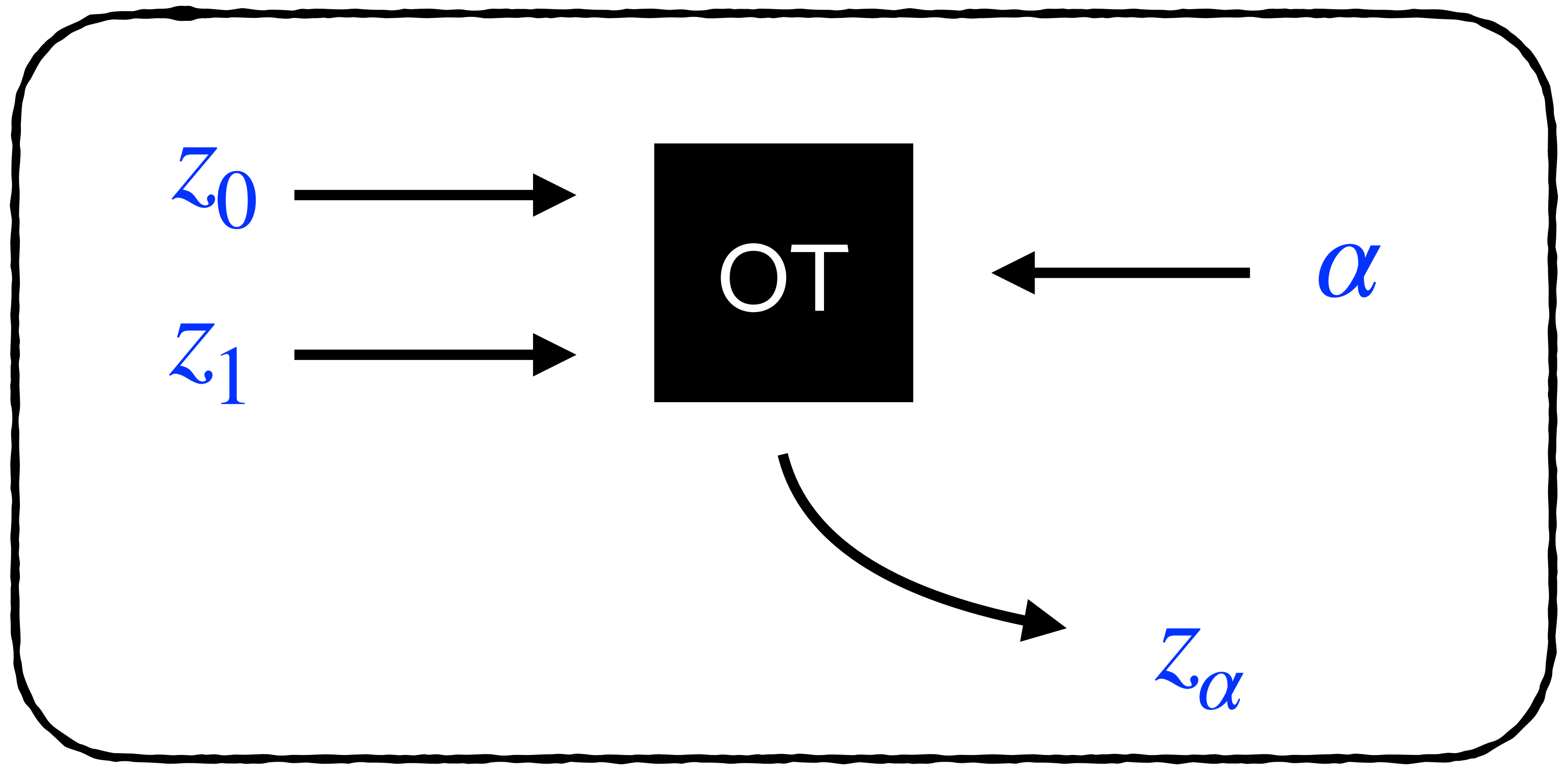
OT-Hybrid Model



- Augment the plain model with an oracle to $\mathsf{OT}((z_0, z_1), \alpha) \mapsto (\perp, z_\alpha)$
- Hybrid Protocol $\Longrightarrow$ Real Protocol (via composition [Can01])
- Costs Breakdown:
  - Computation/Communication & Round Complexity

Background

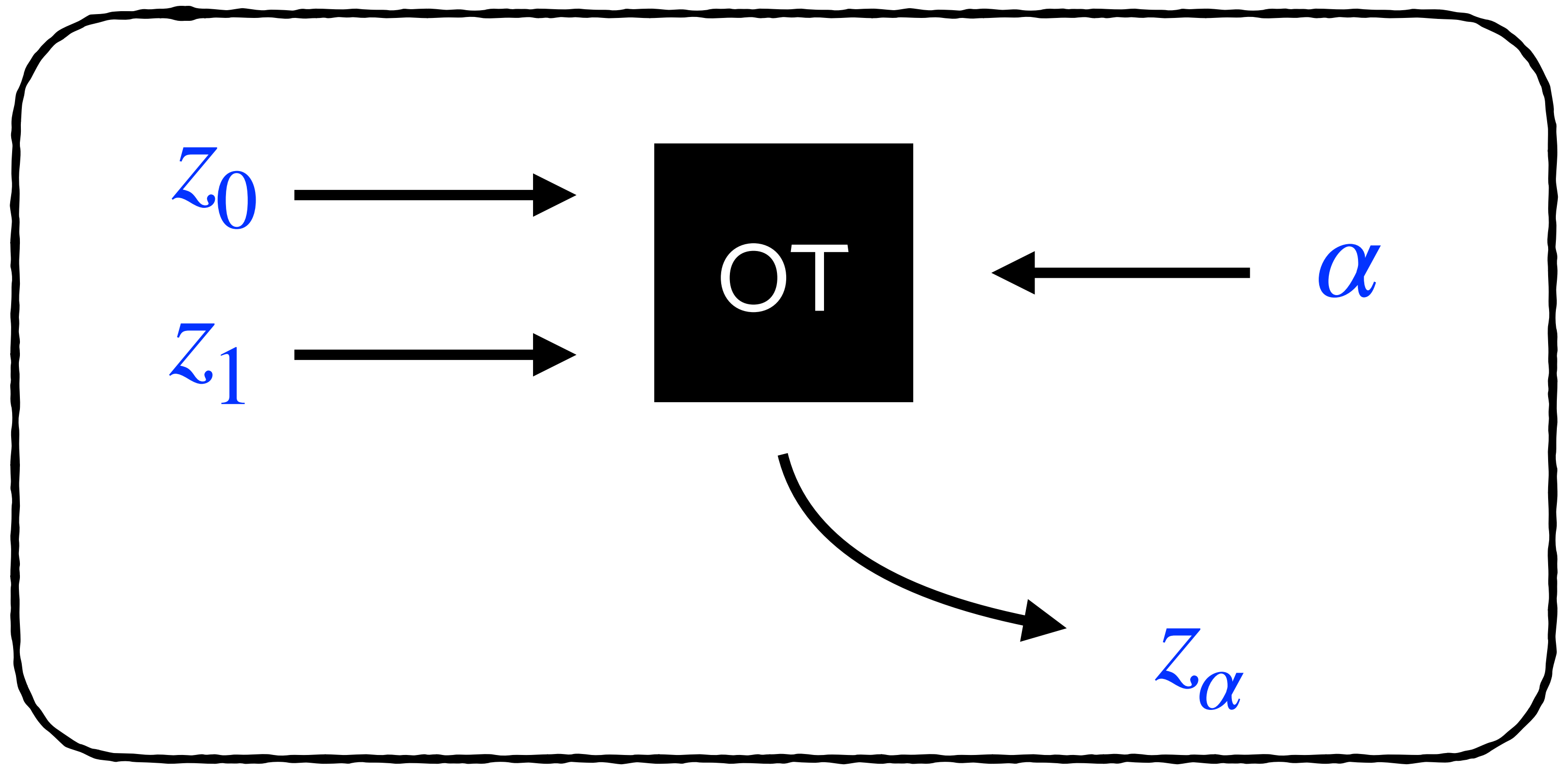
OT-Hybrid Model



- Augment the plain model with an oracle to $\mathsf{OT}((z_0, z_1), \alpha) \mapsto (\perp, z_\alpha)$
- Hybrid Protocol $\implies$ Real Protocol (via composition [Can01])
- Costs Breakdown:
  - Computation/Communication & Round Complexity
  - Calls to OT

Background

OT-Hybrid Model

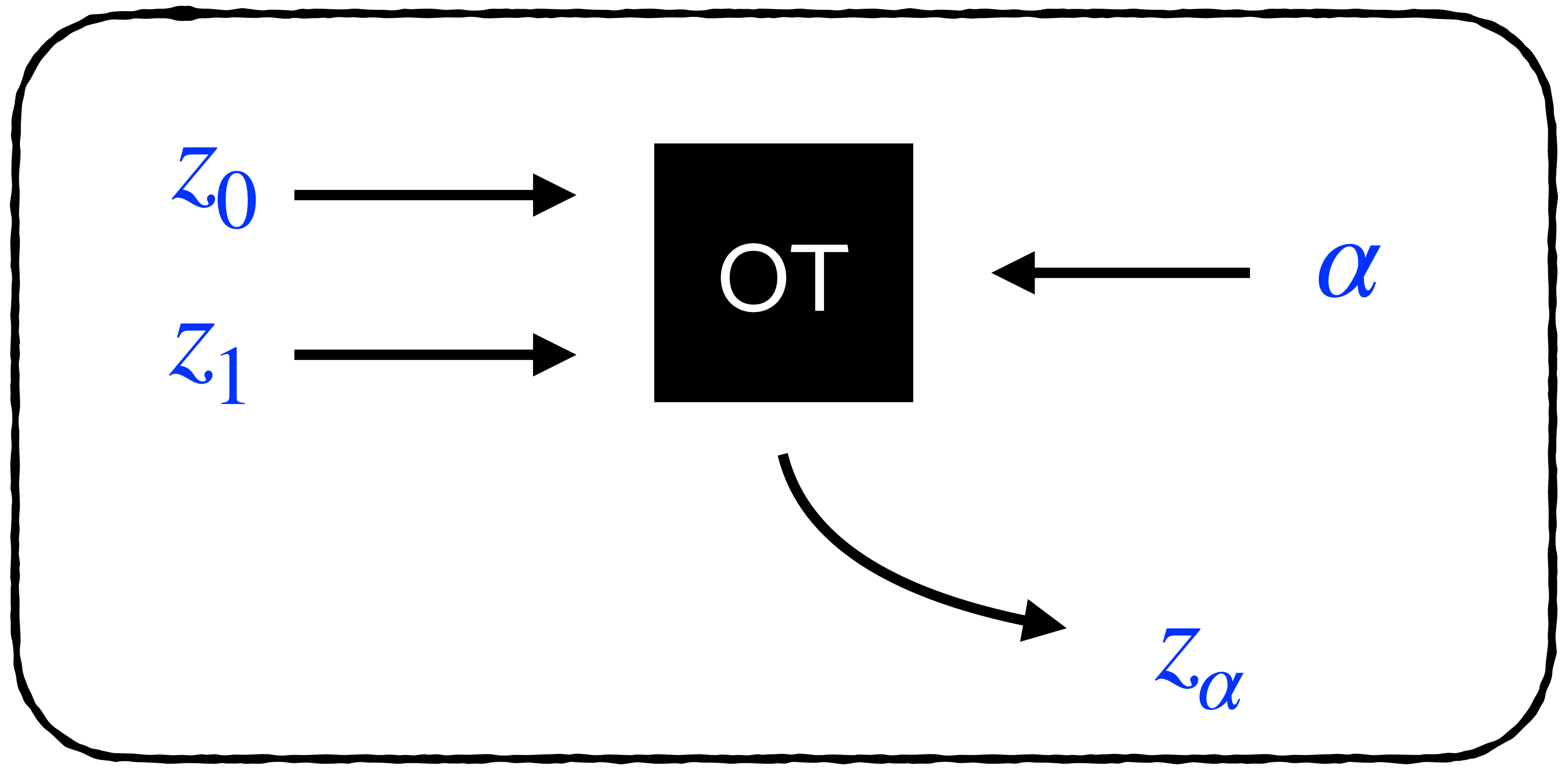


- Augment the plain model with an oracle to $\mathsf{OT}((z_0, z_1), \alpha) \mapsto (\perp, z_\alpha)$
- Hybrid Protocol $\implies$ Real Protocol (via composition [Can01])
- Costs Breakdown:
  - Computation/Communication & Round Complexity
  - Calls to OT
  - Sender input-length

Size of the z's

Background

OT-Hybrid Model



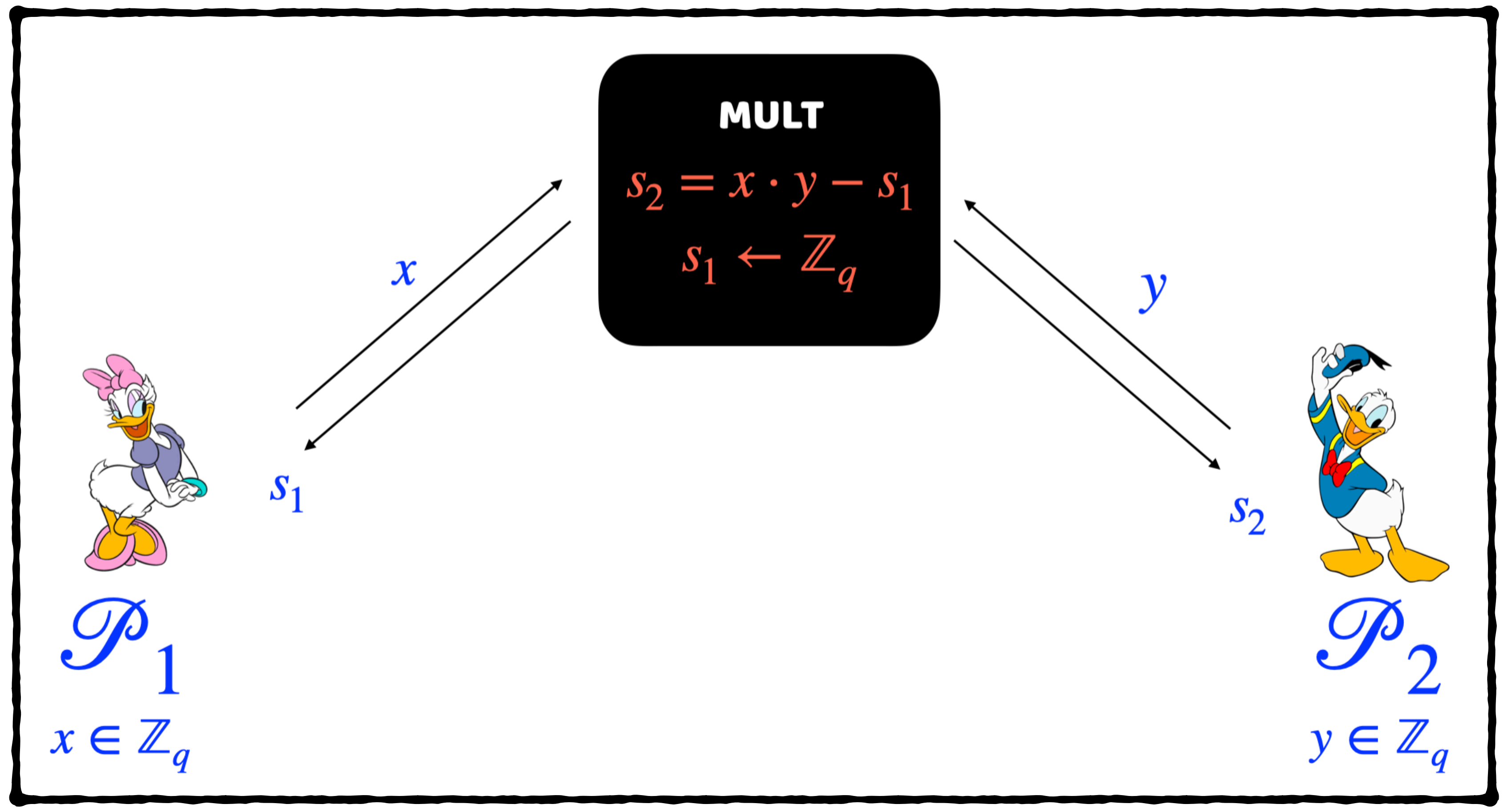
- Augment the plain model with an oracle to $\mathsf{OT}((z_0, z_1), \alpha) \mapsto (\perp, z_\alpha)$
- Hybrid Protocol $\implies$ Real Protocol (via composition [Can01])
- Costs Breakdown:
  - Computation/Communication & Round Complexity
  - Calls to OT
  - Sender input-length

Size of the z's

$\implies$ communication costs in standard model

Background

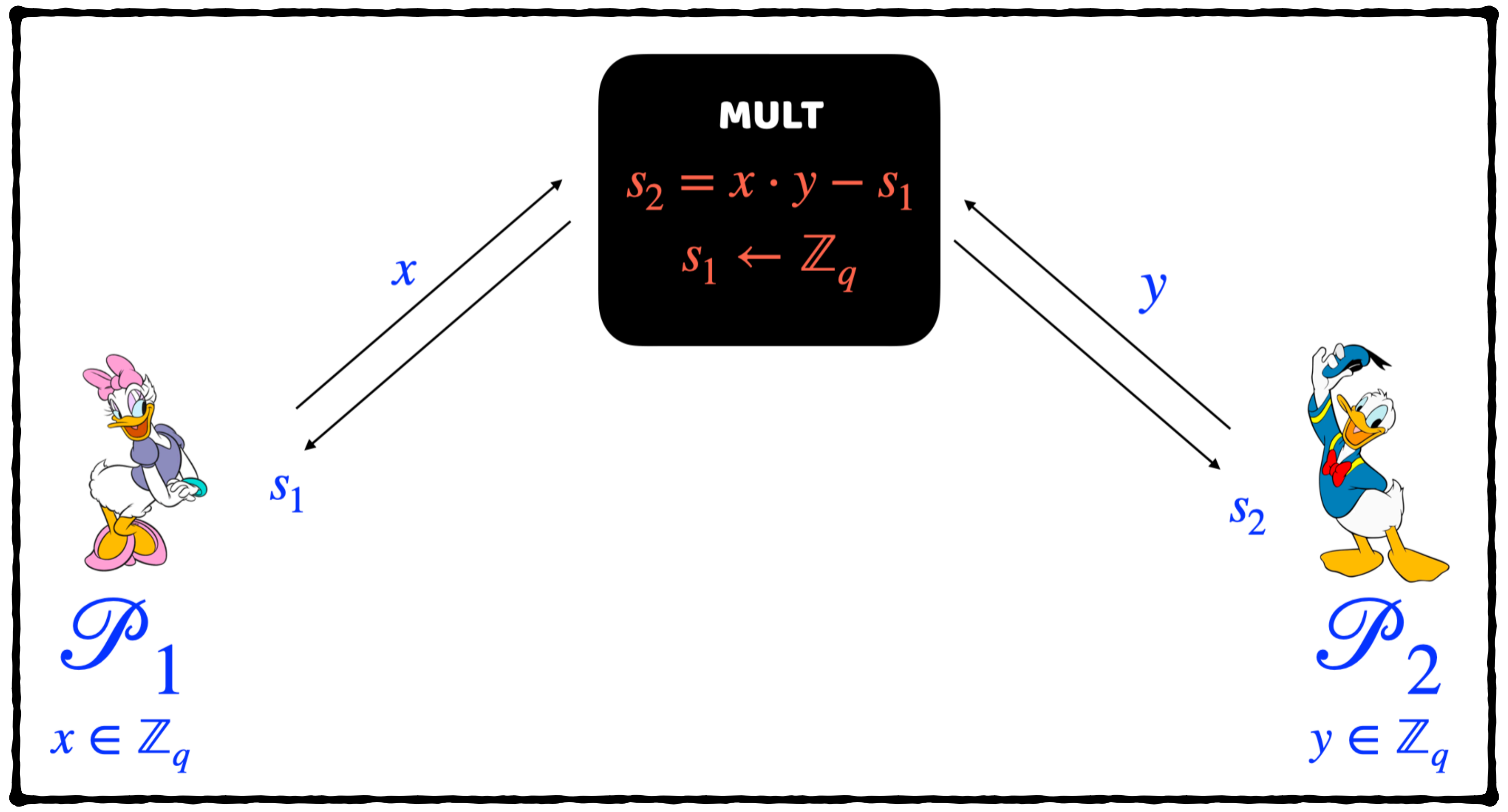
OT-Based Constructions



Background

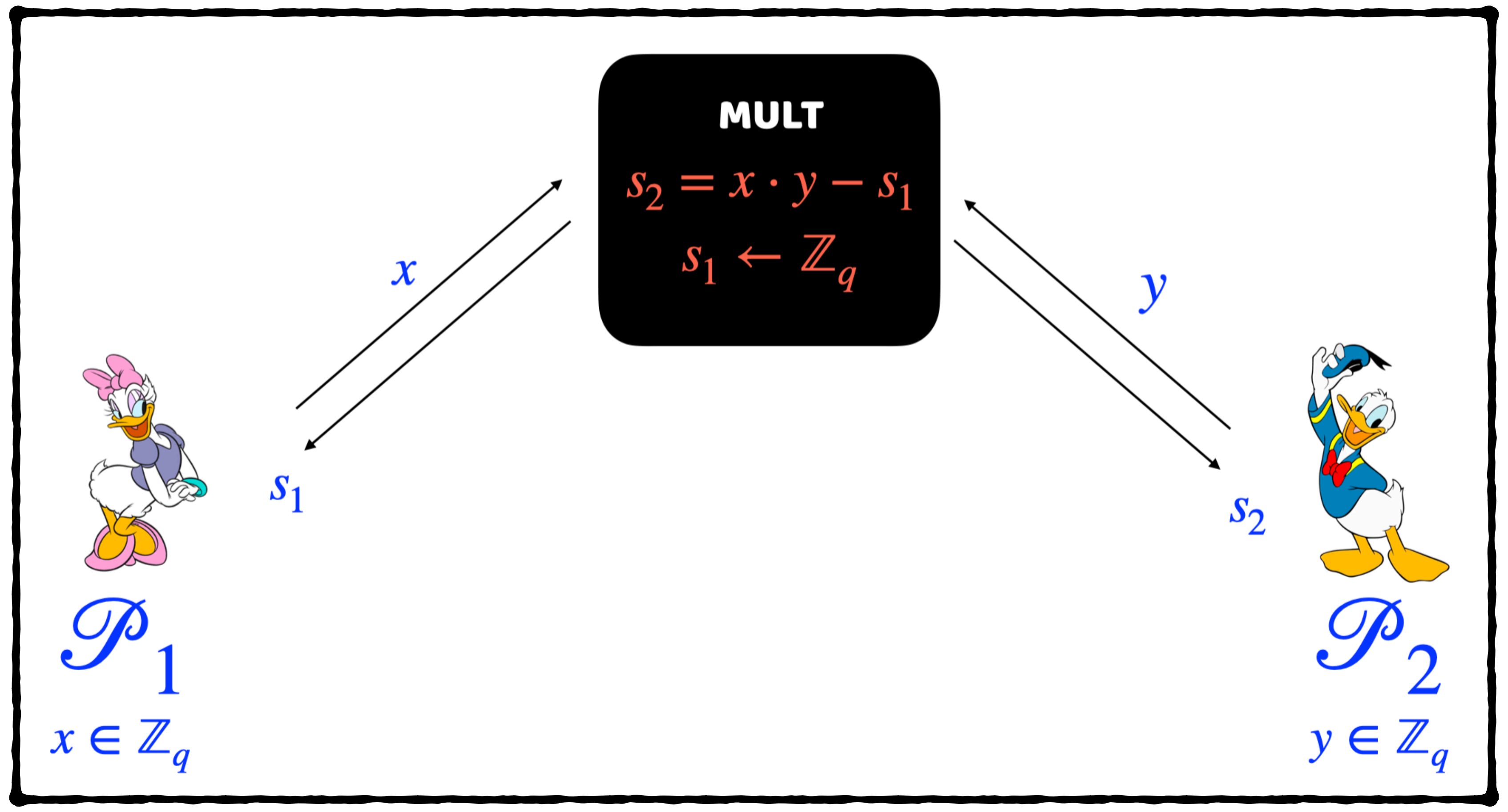
OT-Based Constructions

- Gilboa99



Background

OT-Based Constructions

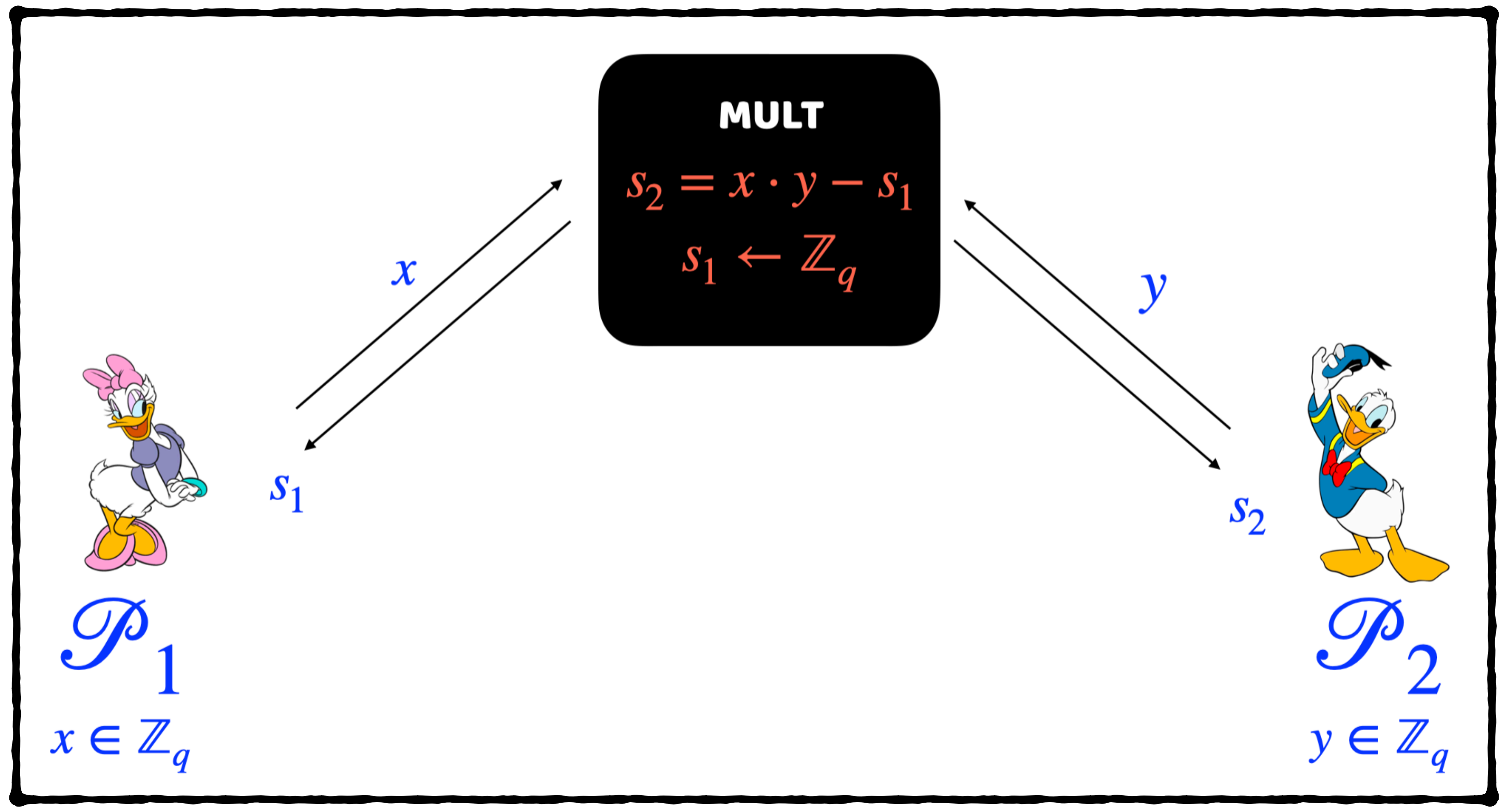


- Gilboa99

- Ishai, Prabhakaran and Sahai (IPS09)

Background

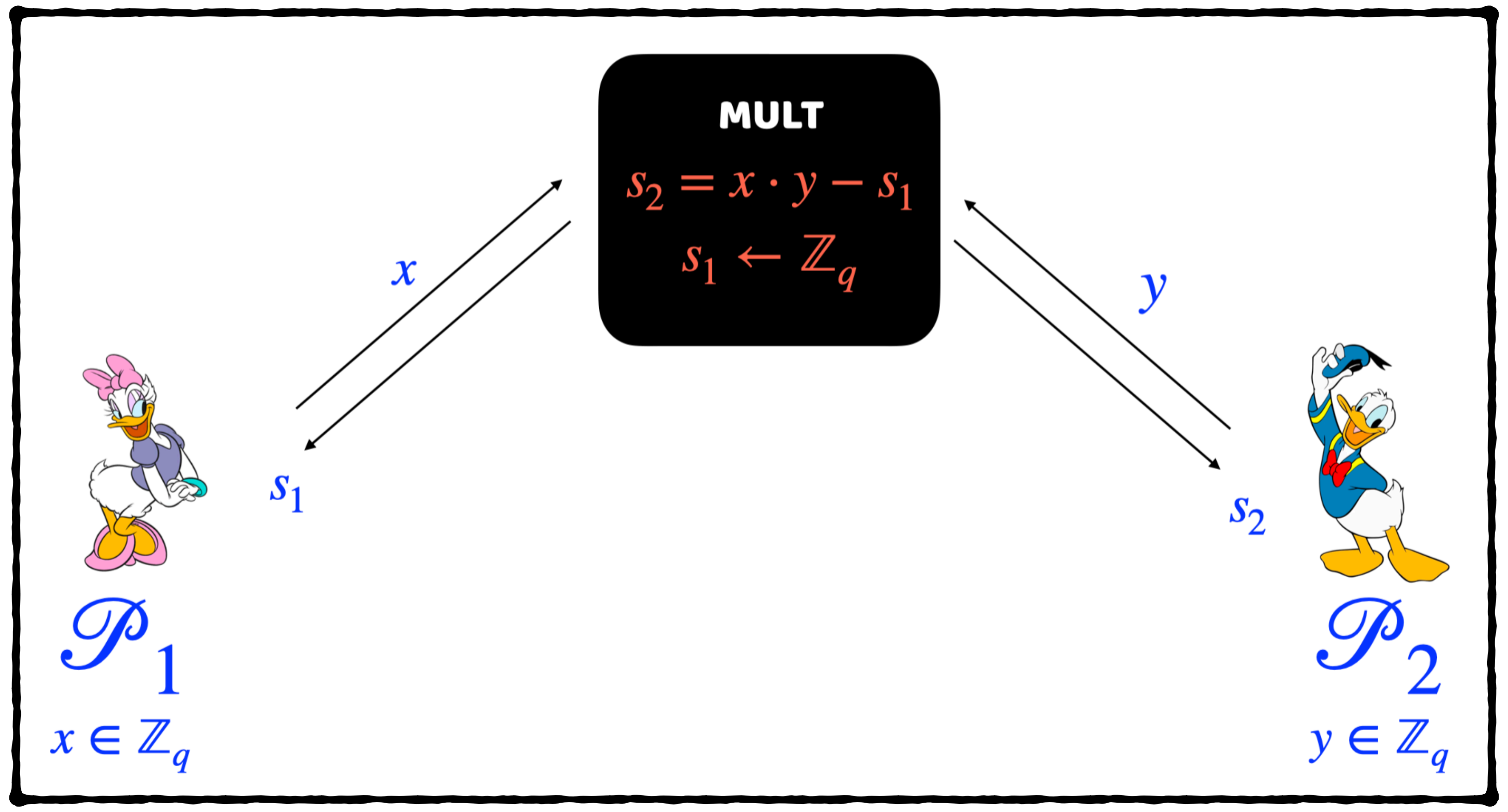
OT-Based Constructions



- Gilboa99
  - 👍 $\log(q)$ OT-calls & Communication
- Ishai, Prabhakaran and Sahai (IPS09)
  - 👎 Less efficient than Gilboa

Background

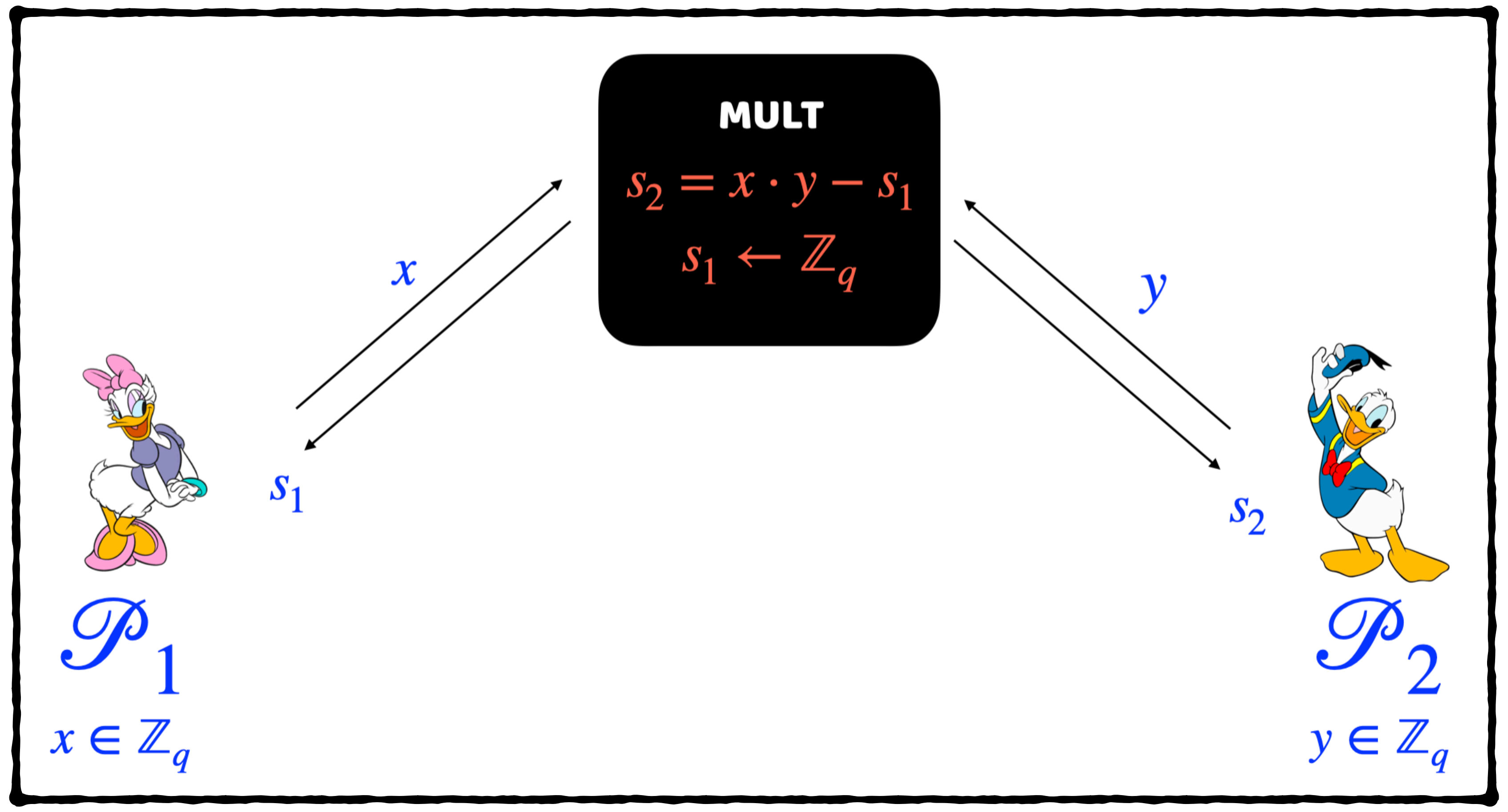
OT-Based Constructions



- Gilboa99
  - 👍 $\log(q)$ OT-calls & Communication
  - 👎 Semi-Honest Security
- Ishai, Prabhakaran and Sahai (IPS09)
  - 👎 Less efficient than Gilboa
  - 👎 Semi-Honest security

Background

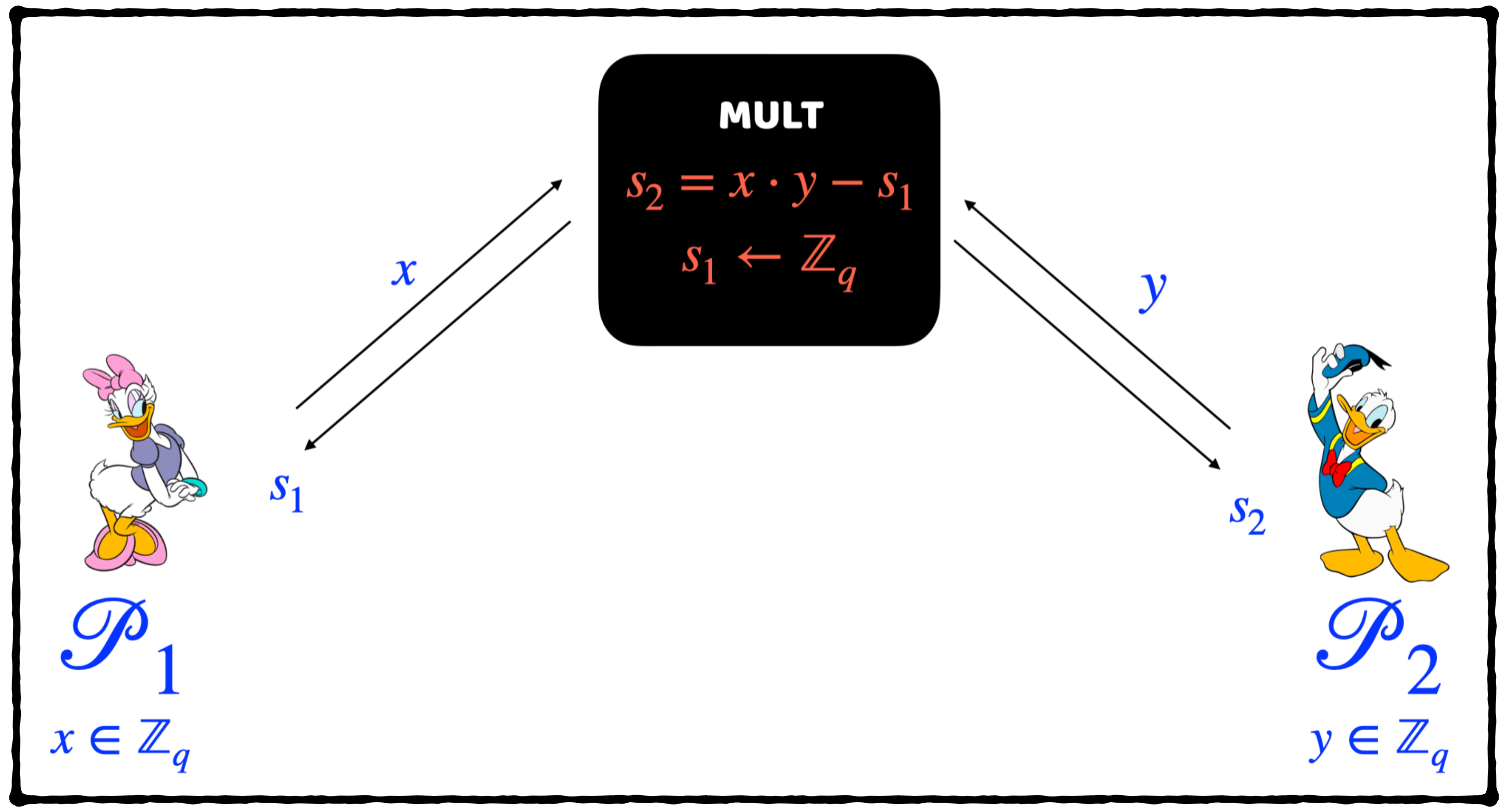
OT-Based Constructions



- Gilboa99
  - 👍 $\log(q)$ OT-calls & Communication
  - 👎 Semi-Honest Security
    - ∃ compilers (MASCOT, DKLs19) for Malicious Security w/ overhead
- Ishai, Prabhakaran and Sahai (IPS09)
  - 👎 Less efficient than Gilboa
  - 👎 Semi-Honest security

Background

OT-Based Constructions



- Gilboa99
  - 👍 $\log(q)$ OT-calls & Communication
  - 👎 Semi-Honest Security
  - ∃ compilers (MASCOT, DKLs19) for Malicious Security w/ overhead
- Ishai, Prabhakaran and Sahai (IPS09)
  - 👎 Less efficient than Gilboa
  - 👎 Semi-Honest security
  - ∃ malicious variant under non-standard assumption

Our Results

Our Results

New OT-Based Multiplication Protocol

Our Results

New OT-Based Multiplication Protocol

- We present a new OT-based multiplication protocol Π

Our Results

New OT-Based Multiplication Protocol

- We present a new OT-based multiplication protocol Π

  1. *Almost* maliciously secure “out of the box”

Our Results

New OT-Based Multiplication Protocol

- We present a new OT-based multiplication protocol Π

  1. *Almost* maliciously secure “out of the box”

  2. Admits a batching variant

$\approx$ Vector-OLE

$$\mathsf{BatchMult}(x, (y_1, \ldots, y_\beta)) \mapsto ((\ldots, s_i, \ldots), (\ldots, x \cdot y_i - s_i, \ldots))$$

Our Results

New OT-Based Multiplication Protocol

- We present a new OT-based multiplication protocol Π

1. *Almost* maliciously secure “out of the box”

2. Admits a batching variant

$\approx$ Vector-OLE

$$\mathsf{BatchMult}(x, (y_1, \ldots, y_\beta)) \mapsto ((\ldots, s_i, \ldots), (\ldots, x \cdot y_i - s_i, \ldots))$$

3. Can be compiled cheaply into a fully secure protocol

Our Results

New OT-Based Multiplication Protocol

- We present a new OT-based multiplication protocol Π

1. *Almost* maliciously secure “out of the box”

2. Admits a batching variant

$\approx$ Vector-OLE

$$\mathsf{BatchMult}(x, (y_1, \ldots, y_\beta)) \mapsto ((\ldots, s_i, \ldots), (\ldots, x \cdot y_i - s_i, \ldots))$$

3. Can be compiled cheaply into a fully secure protocol

4. x2 improvement in communication compared to SoA

Our Results (Cont'd)

New OT-Based Multiplication Protocol

Our Results (Cont'd)

New OT-Based Multiplication Protocol

Theorem (Informal)

Our Results (Cont'd)

New OT-Based Multiplication Protocol

Π is secure *or*
output is utterly unpredictable

Theorem (Informal)

Our Results (Cont'd)

New OT-Based Multiplication Protocol

Π is secure *or*
output is utterly unpredictable

Theorem (Informal)

Stat param κ, honest input z, exactly one of the following holds true.

Our Results (Cont'd)

New OT-Based Multiplication Protocol

Π is secure *or*
output is utterly unpredictable

Theorem (Informal)

Stat param κ, honest input z, exactly one of the following holds true.

- Π realizes the ideal (perfect) multiplication functionality with $2^{-\kappa/4}$ statistical closeness.

Our Results (Cont'd)

New OT-Based Multiplication Protocol

Π is secure *or*
output is utterly unpredictable

Theorem (Informal)

Stat param κ, honest input z, exactly one of the following holds true.

- Π realizes the ideal (perfect) multiplication functionality with $2^{-\kappa/4}$ statistical closeness.
- $H_\infty(\mathsf{out}_\Pi^{\mathscr{A}}(z) \mid \mathsf{view}_\Pi^{\mathscr{A}}(z), z) \geq \kappa/4$.

Our Results (Cont'd)

New OT-Based Multiplication Protocol

Π is secure *or*
output is utterly unpredictable

Theorem (Informal)

Stat param κ, honest input z, exactly one of the following holds true.

- Π realizes the ideal (perfect) multiplication functionality with $2^{-\kappa/4}$ statistical closeness.
- $H_\infty(\mathsf{out}_\Pi^{\mathscr{A}}(z) \mid \mathsf{view}_\Pi^{\mathscr{A}}(z), z) \geq \kappa/4$.

MinEntropy of honest output given
$\mathscr{A}$'s ***view and honest input***

Our Results (Cont'd)

New OT-Based Multiplication Protocol

Π is secure *or*
output is utterly unpredictable

Theorem (Informal)

Stat param κ, honest input z, exactly one of the following holds true.

- Π realizes the ideal (perfect) multiplication functionality with $2^{-\kappa/4}$ statistical closeness.
- $H_\infty(\mathsf{out}_\Pi^{\mathscr{A}}(z) \mid \mathsf{view}_\Pi^{\mathscr{A}}(z), z) \geq \kappa/4$.

MinEntropy of honest output given
$\mathscr{A}$'s ***view and honest input***

- Output is highly unpredictable under attack, *even given the input.*

Our Results (Cont'd)

New OT-Based Multiplication Protocol

Π is secure *or*
output is utterly unpredictable

Theorem (Informal)

Stat param κ, honest input z, exactly one of the following holds true.

- Π realizes the ideal (perfect) multiplication functionality with $2^{-\kappa/4}$ statistical closeness.
- $H_\infty(\mathsf{out}_\Pi^{\mathscr{A}}(z) \mid \mathsf{view}_\Pi^{\mathscr{A}}(z), z) \geq \kappa/4$.

MinEntropy of honest output given
$\mathscr{A}$'s ***view and honest input***

Batching: each output is
unpredictable, even given ***all*** the inputs

- Output is highly unpredictable under attack, *even given the input.*

Our Results (Cont'd)

New OT-Based Multiplication Protocol

Π is secure *or*
output is utterly unpredictable

Theorem (Informal)

Stat param κ, honest input z, exactly one of the following holds true.

- Π realizes the ideal (perfect) multiplication functionality with $2^{-\kappa/4}$ statistical closeness.
- $H_\infty(\mathsf{out}_\Pi^{\mathscr{A}}(z) \mid \mathsf{view}_\Pi^{\mathscr{A}}(z), z) \geq \kappa/4$.

MinEntropy of honest output given
$\mathscr{A}$'s ***view and honest input***

Batching: each output is
unpredictable, even given ***all*** the inputs

- Output is highly unpredictable under attack, *even given the input.*
- Formal definition of "unpredictability" via functionality WeakMult

Applications

New OT-Based Multiplication Protocol

Applications

New OT-Based Multiplication Protocol

- Achieving Full Security

Applications

New OT-Based Multiplication Protocol

- Achieving Full Security
  - Simple *a posteriori* check on the shares suffices.

Applications

New OT-Based Multiplication Protocol

- Achieving Full Security
  - Simple *a posteriori* check on the shares suffices.
  - In a q-order group $\mathbb{G} = \langle g \rangle$, party $\mathscr{P}_1$ simply checks that $B^x \cdot g^{-s_1} = R_2$

Applications

New OT-Based Multiplication Protocol

- Achieving Full Security
  - Simple *a posteriori* check on the shares suffices.
  - In a q-order group $\mathbb{G} = \langle g \rangle$, party $\mathscr{P}_1$ simply checks that $B^x \cdot g^{-s_1} = R_2$

Assuming
$(B, R_2) = (g^y, g^{s_2})$
and B is public

Applications

New OT-Based Multiplication Protocol

- Achieving Full Security
  - Simple *a posteriori* check on the shares suffices.
  - In a q-order group $\mathbb{G} = \langle g \rangle$, party $\mathscr{P}_1$ simply checks that $B^x \cdot g^{-s_1} = R_2$
  - Above generalizes to arbitrary settings. (ask me later)

Assuming
$(B, R_2) = (g^y, g^{s_2})$
and B is public

Applications

New OT-Based Multiplication Protocol

- Achieving Full Security
  - Simple *a posteriori* check on the shares suffices.
  - In a q-order group $\mathbb{G} = \langle g \rangle$, party $\mathscr{P}_1$ simply checks that $B^x \cdot g^{-s_1} = R_2$
  - Above generalizes to arbitrary settings. (ask me later)
- Achieving WeakMult is good enough for certain applications.

Assuming $(B, R_2) = (g^y, g^{s_2})$ and B is public

Applications

New OT-Based Multiplication Protocol

- Achieving Full Security
  - Simple *a posteriori* check on the shares suffices.
  - In a q-order group $\mathbb{G} = \langle g \rangle$, party $\mathscr{P}_1$ simply checks that $B^x \cdot g^{-s_1} = R_2$
  - Above generalizes to arbitrary settings. (ask me later)
- Achieving WeakMult is good enough for certain applications.

Assuming $(B, R_2) = (g^y, g^{s_2})$ and B is public

e.g. t-ECDSA [LN18] (new version soon)

Applications

New OT-Based Multiplication Protocol

- Achieving Full Security
  - Simple *a posteriori* check on the shares suffices.
  - In a q-order group $\mathbb{G} = \langle g \rangle$, party $\mathscr{P}_1$ simply checks that $B^x \cdot g^{-s_1} = R_2$
  - Above generalizes to arbitrary settings. (ask me later)
- Achieving WeakMult is good enough for certain applications.
- Batching variant is useful for generating triplets in preprocessing model.

Assuming $(B, R_2) = (g^y, g^{s_2})$ and B is public

e.g. t-ECDSA [LN18] (new version soon)

Comparison (in ROM)

Gilboa, IPS, MASCOT, DKLs

	Gilboa99	IPS09	MASCOT/ DKLs19	Our Work
Security	Semi-Honest	Semi-Honest	**Malicious**	**Malicious**
# of OT-Calls				
Communication per OT-Call				
Batching Overhead				

Comparison (in ROM)

Gilboa, IPS, MASCOT, DKLs

	Gilboa99	IPS09	MASCOT/ DKLs19	Our Work
Security	Semi-Honest	Semi-Honest	**Malicious**	**Malicious**
# of OT-Calls	$\log(q)$	n	n	n
Communication per OT-Call				
Batching Overhead				

$n = \log(q) + \kappa$

Comparison (in ROM)

Gilboa, IPS, MASCOT, DKLs

	Gilboa99	IPS09	MASCOT/ DKLs19	Our Work
Security	Semi-Honest	Semi-Honest	**Malicious**	**Malicious**
# of OT-Calls	$\log(q)$	n	n	n
Communication per OT-Call	$\log(q)$	$\log(q)$	$2 \cdot \log(q)$	$\log(q)$
Batching Overhead				

$n = \log(q) + \kappa$

Comparison (in ROM)

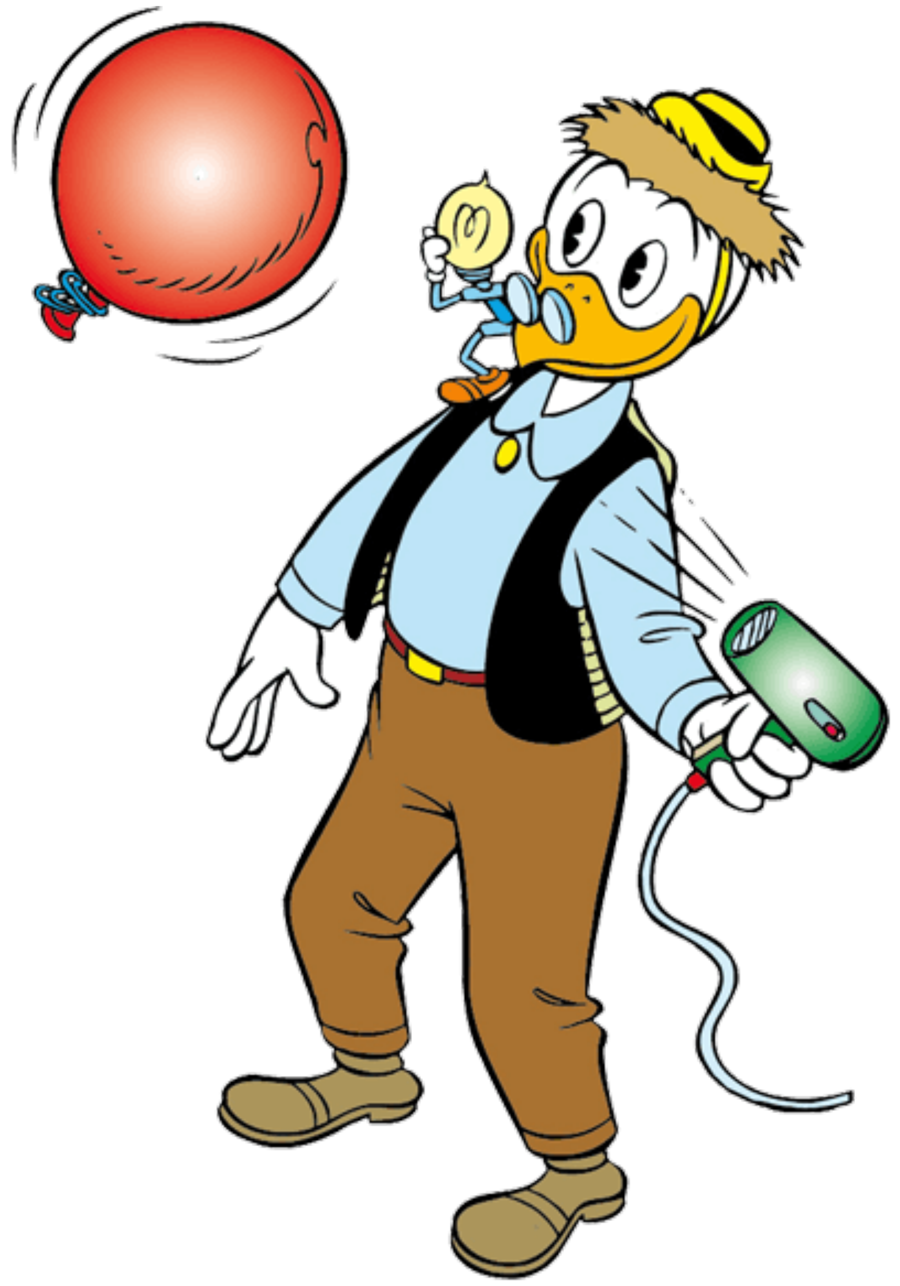
Gilboa, IPS, MASCOT, DKLs

	Gilboa99	IPS09	MASCOT/ DKLs19	Our Work
Security	Semi-Honest	Semi-Honest	**Malicious**	**Malicious**
# of OT-Calls	$\log(q)$	n	n	n
Communication per OT-Call	$\log(q)$	$\log(q)$	$2 \cdot \log(q)$	$\log(q)$
Batching Overhead	$\beta \cdot \log(q)$	$\beta \cdot n$	$\beta \cdot n$	$\beta \cdot \log(q)$

$n = \log(q) + \kappa$

Batch Size $= \beta$

Technical Overview



Technical Overview

Notation

Technical Overview

Notation

1. Boldface $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{z}, \ldots$ denote vectors with $\boldsymbol{u} = (u_1, \ldots, u_n), \ldots$

Technical Overview

Notation

1. Boldface $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{z}, \ldots$ denote vectors with $\boldsymbol{u} = (u_1, \ldots, u_n), \ldots$

2. Write $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \sum_{i=1}^{n} u_i \cdot v_i$ for the inner product of $\boldsymbol{u}$ and $\boldsymbol{v}$.

Technical Overview

Notation

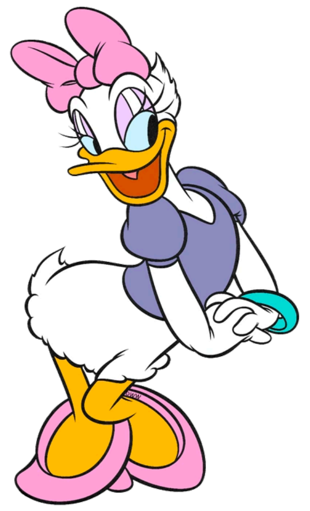
1. Boldface $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{z}, \ldots$ denote vectors with $\boldsymbol{u} = (u_1, \ldots, u_n), \ldots$

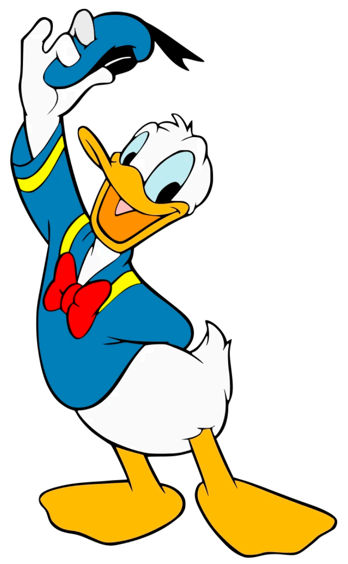
2. Write $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \sum_{i=1}^{n} u_i \cdot v_i$ for the inner product of $\boldsymbol{u}$ and $\boldsymbol{v}$.

3. Write $\boldsymbol{u} * \boldsymbol{v} = (u_1 \cdot v_1, \ldots, u_n \cdot v_n)$ for pointwise (Hadamard) product.

Our Protocol

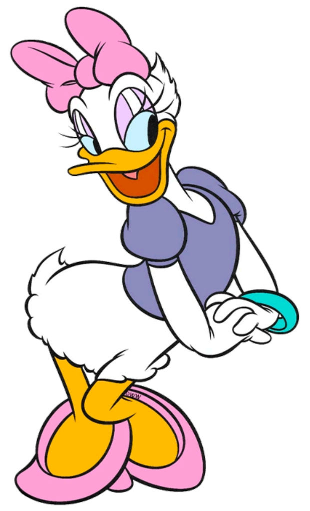
$\mathscr{P}_1$ *and* $\mathscr{P}_2$ *hold inputs* x *and* $y \in \mathbb{Z}_q$ *respectively*

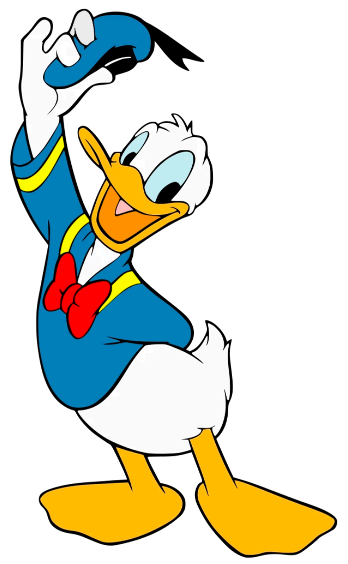




Our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

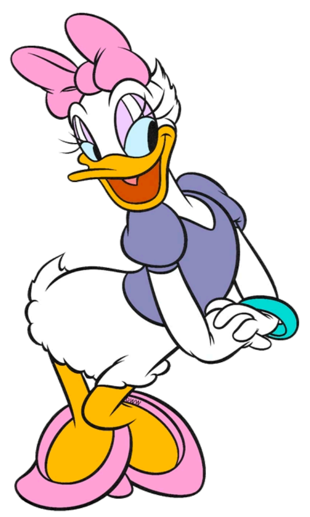


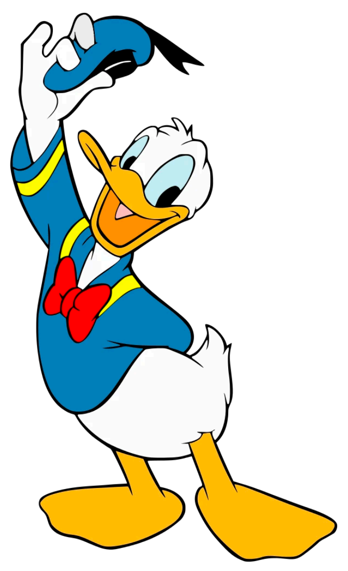


Our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$n = \log(q) + \kappa$
calls to OT (in parallel)

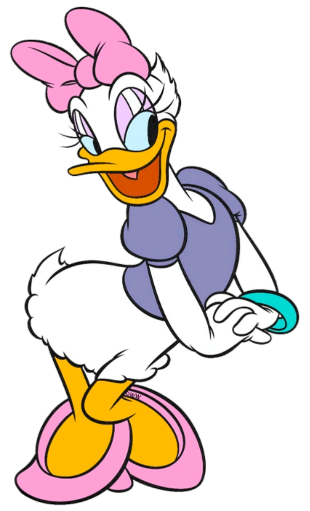


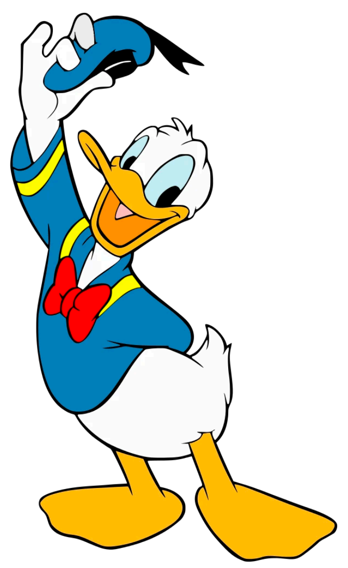


Our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$n = \log(q) + \kappa$
calls to OT (in parallel)



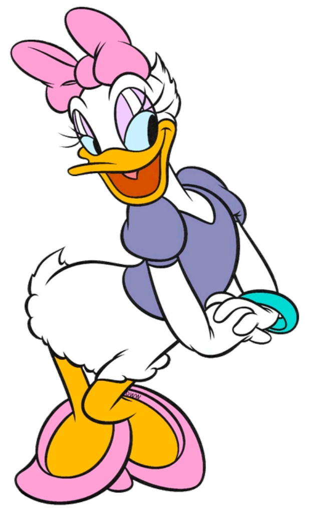


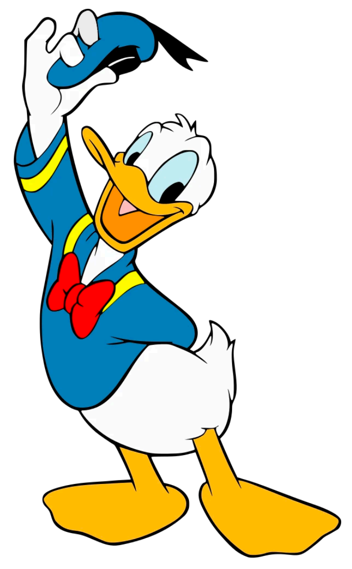
1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.

Our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$n = \log(q) + \kappa$
calls to OT (in parallel)



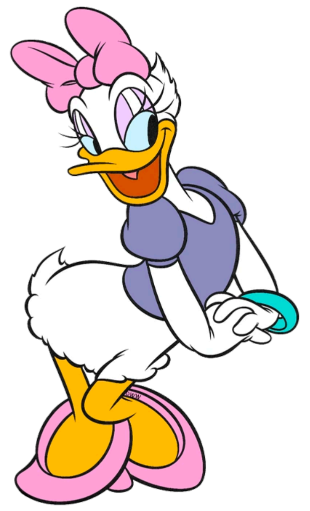


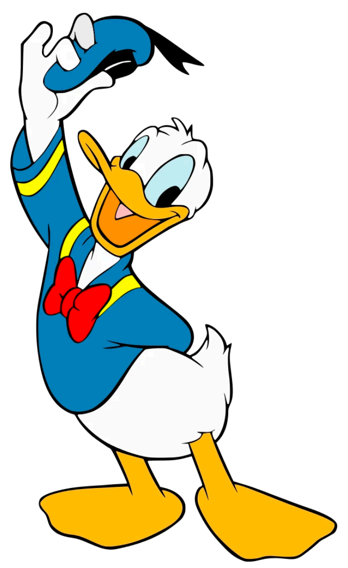
1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.

Our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$n = \log(q) + \kappa$
calls to OT (in parallel)





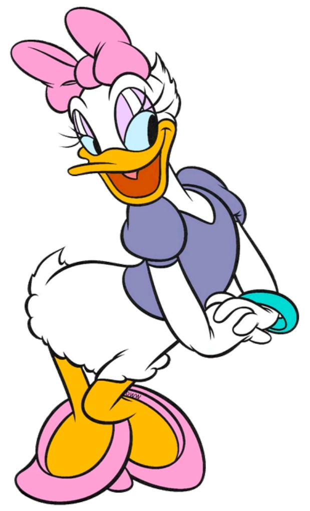
1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

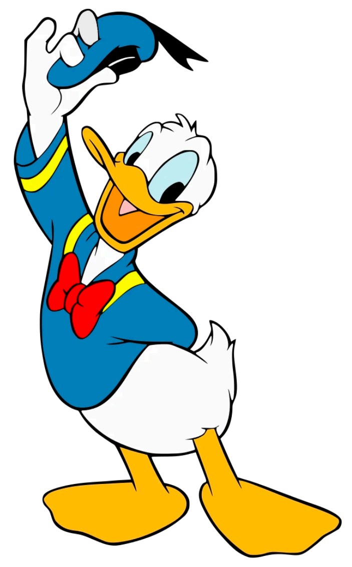
1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.

Our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$n = \log(q) + \kappa$
calls to OT (in parallel)





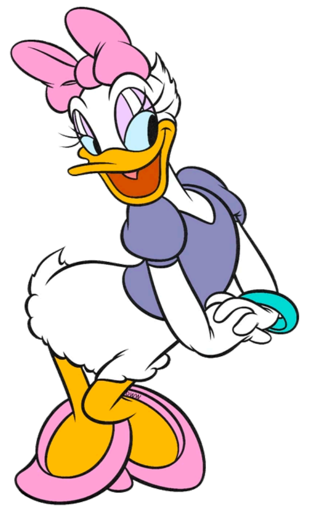
1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

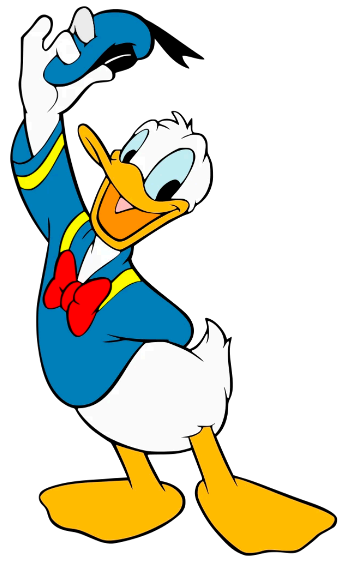
1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.

Our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$n = \log(q) + \kappa$
calls to OT (in parallel)





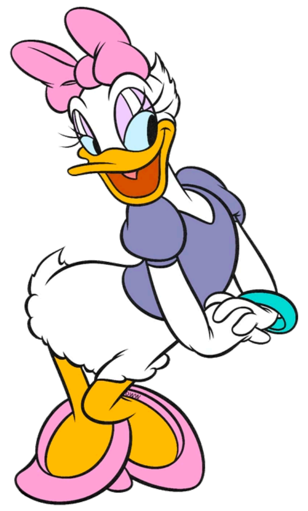
1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

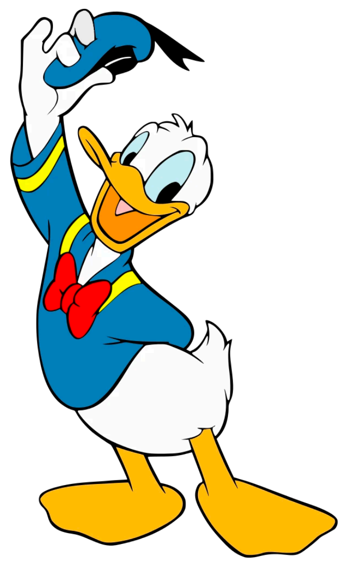
1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.

Our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$n = \log(q) + \kappa$
calls to OT (in parallel)





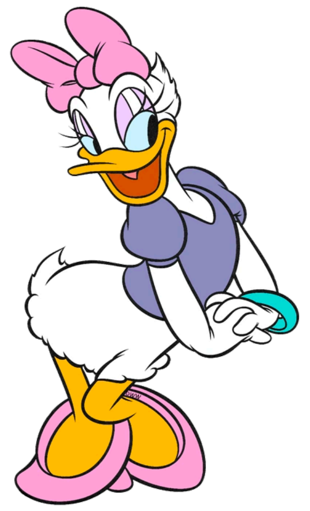
1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.
2. Sample $\boldsymbol{v} \leftarrow \mathbb{Z}_q^n$ s.t. $\langle \boldsymbol{t}, \boldsymbol{v} \rangle = y$.

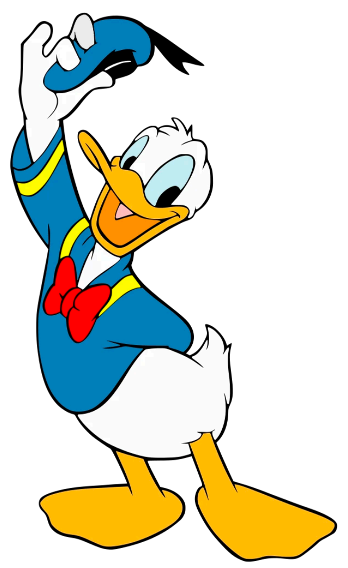
Our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$n = \log(q) + \kappa$
calls to OT (in parallel)



$\boldsymbol{v} = (v_1, \ldots, v_n)$



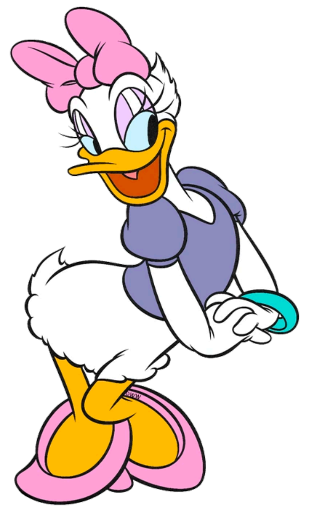
1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

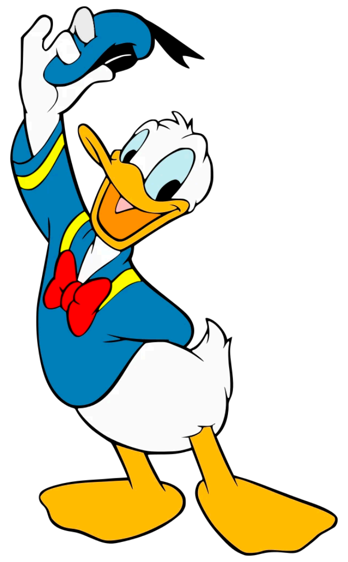
1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.
2. Sample $\boldsymbol{v} \leftarrow \mathbb{Z}_q^n$ s.t. $\langle \boldsymbol{t}, \boldsymbol{v} \rangle = y$.

Our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$n = \log(q) + \kappa$
calls to OT (in parallel)





$\boldsymbol{v} = (v_1, \dots, v_n)$

1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

   Output $\langle -\boldsymbol{\delta}, \boldsymbol{v} \rangle$.

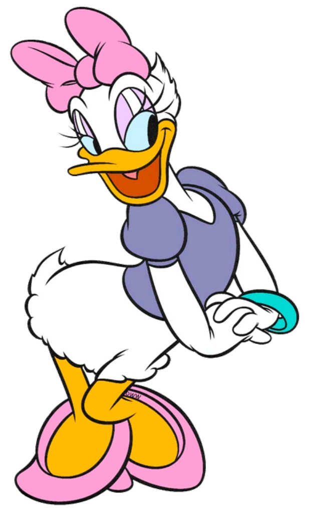
1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.
2. Sample $\boldsymbol{v} \leftarrow \mathbb{Z}_q^n$ s.t. $\langle \boldsymbol{t}, \boldsymbol{v} \rangle = y$.

   Output $\langle z, \boldsymbol{v} \rangle$.

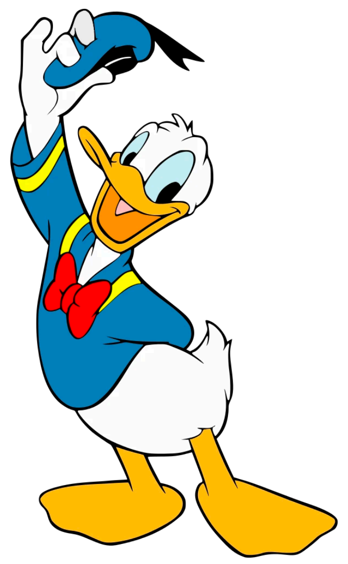
Our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$n = \log(q) + \kappa$
calls to OT (in parallel)



$\boldsymbol{v} = (v_1, \ldots, v_n)$ $\longleftarrow$



1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

Output $\langle -\boldsymbol{\delta}, \boldsymbol{v} \rangle$.

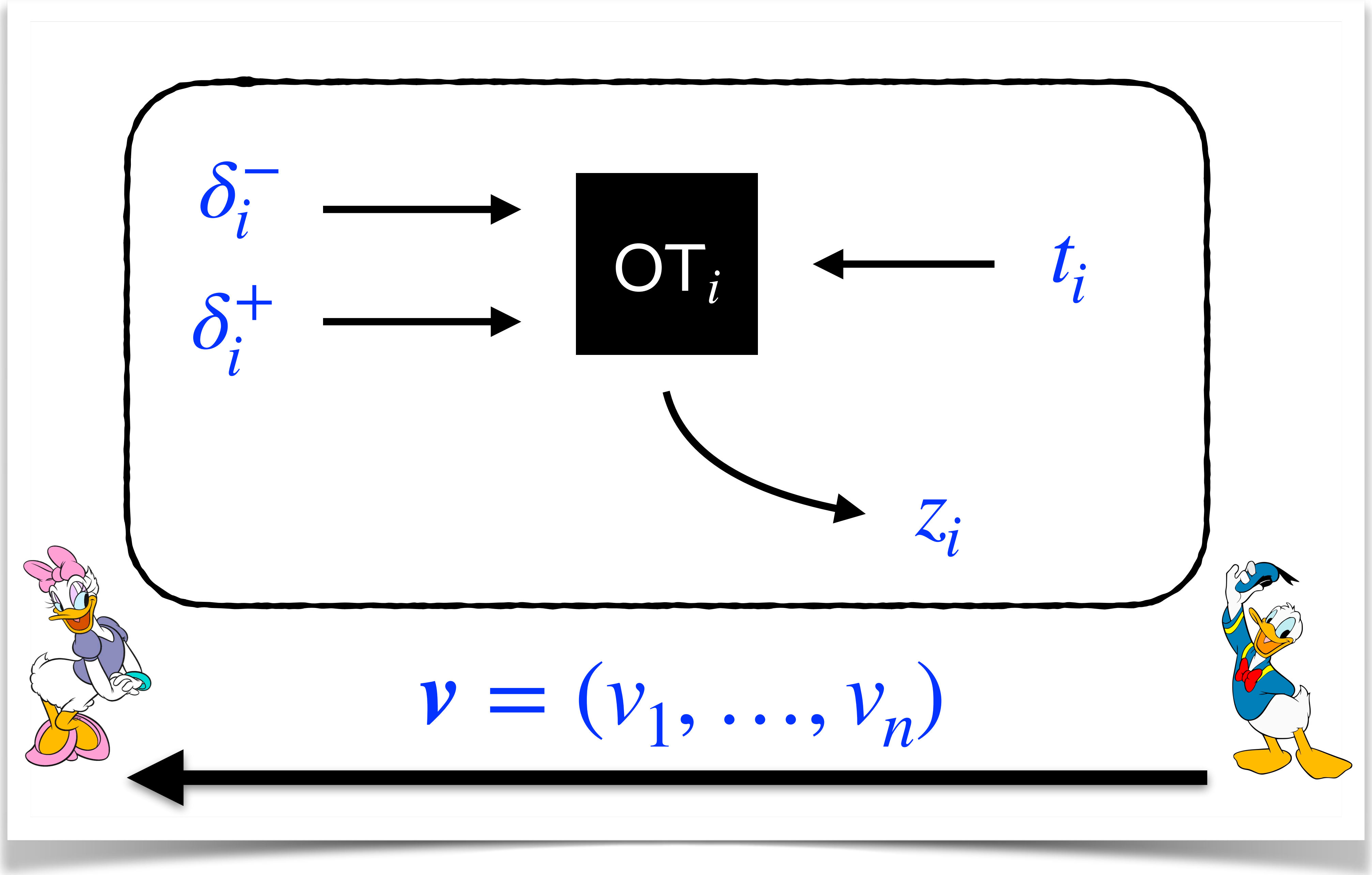
1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.
2. Sample $\boldsymbol{v} \leftarrow \mathbb{Z}_q^n$ s.t. $\langle \boldsymbol{t}, \boldsymbol{v} \rangle = y$.

Output $\langle \boldsymbol{z}, \boldsymbol{v} \rangle$.

$$\boxed{\langle -\boldsymbol{\delta} + \boldsymbol{z}, \boldsymbol{v} \rangle = \langle x \cdot \boldsymbol{t}, \boldsymbol{v} \rangle = x \cdot \langle \boldsymbol{t}, \boldsymbol{v} \rangle = x \cdot y}$$

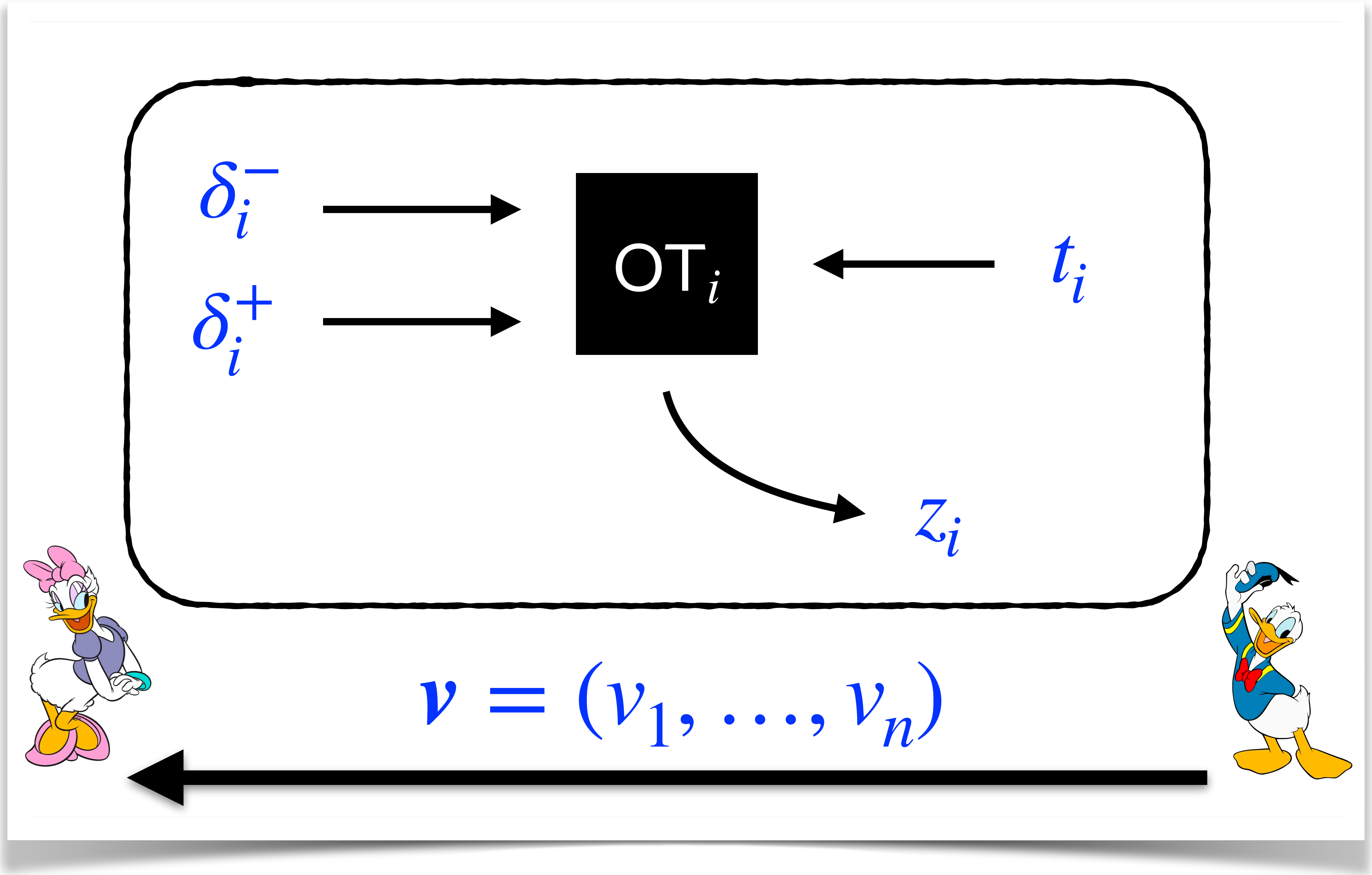
Security

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively



Security

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

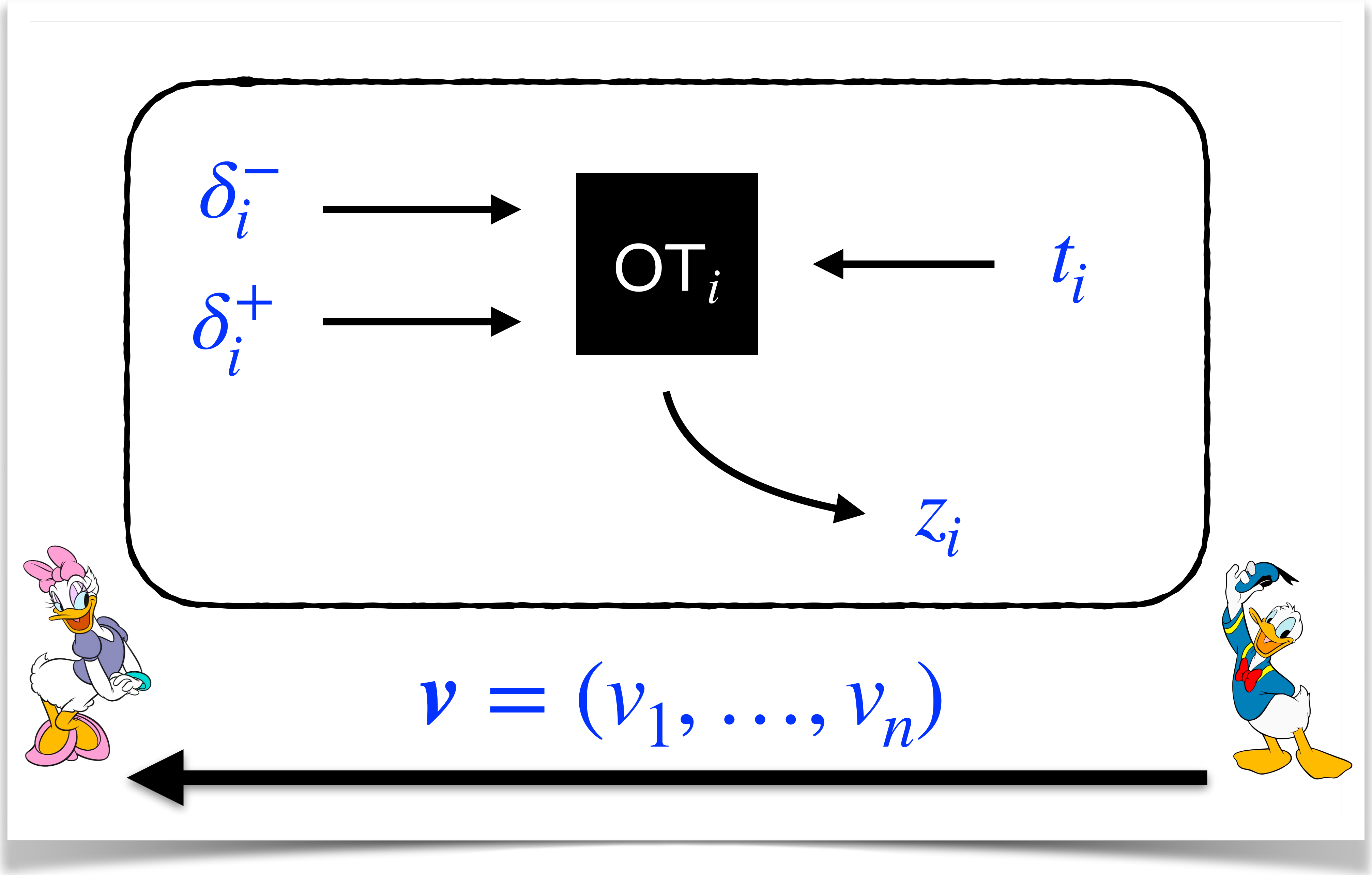


- Protocol is fully secure against $\mathscr{P}_2^{\mathscr{A}}$

Security

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

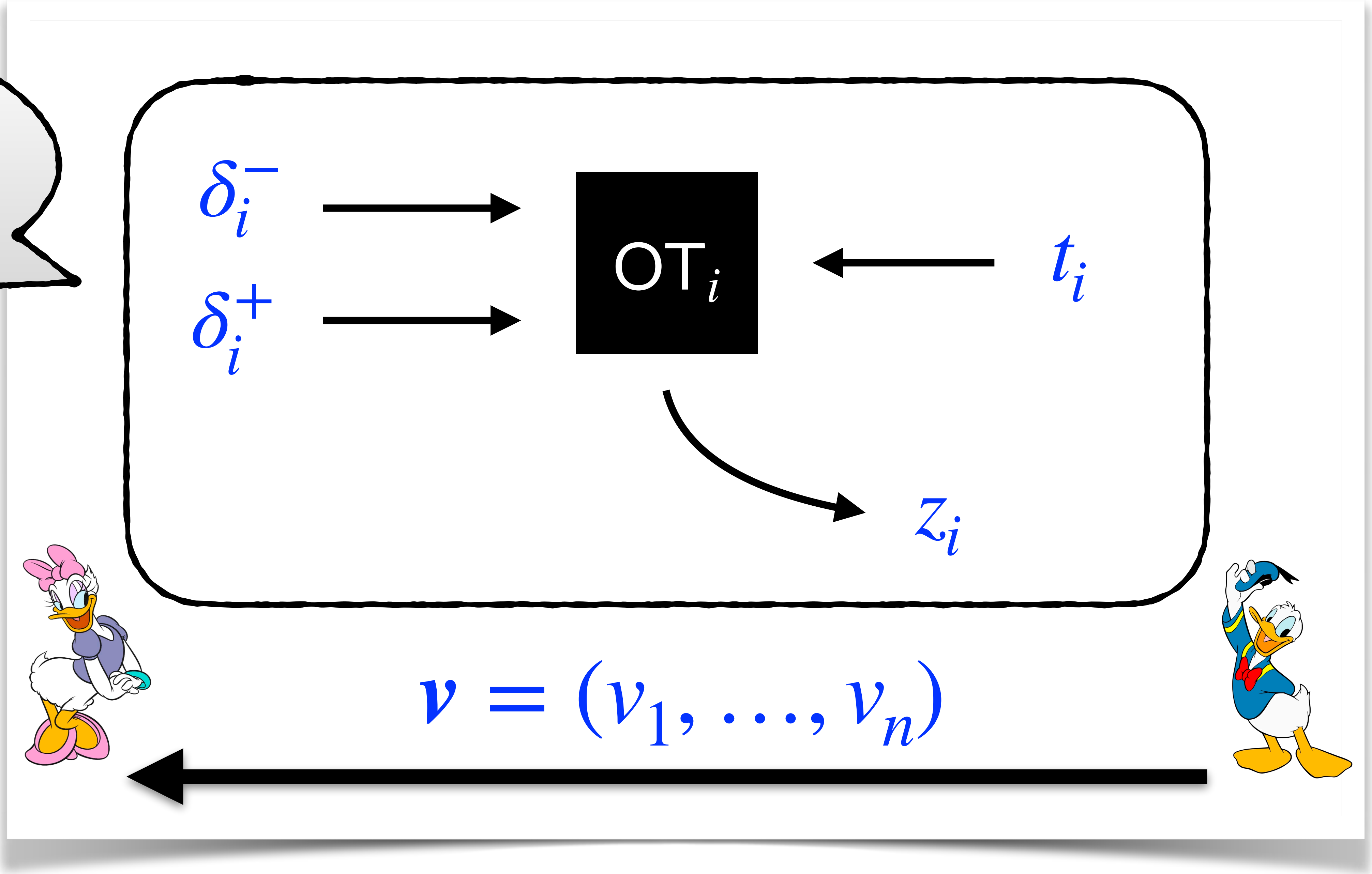
- Protocol is fully secure against $\mathscr{P}_2^{\mathscr{A}}$
- $\mathscr{P}_1^{\mathscr{A}}$ can use inconsistent inputs in OT



Security

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

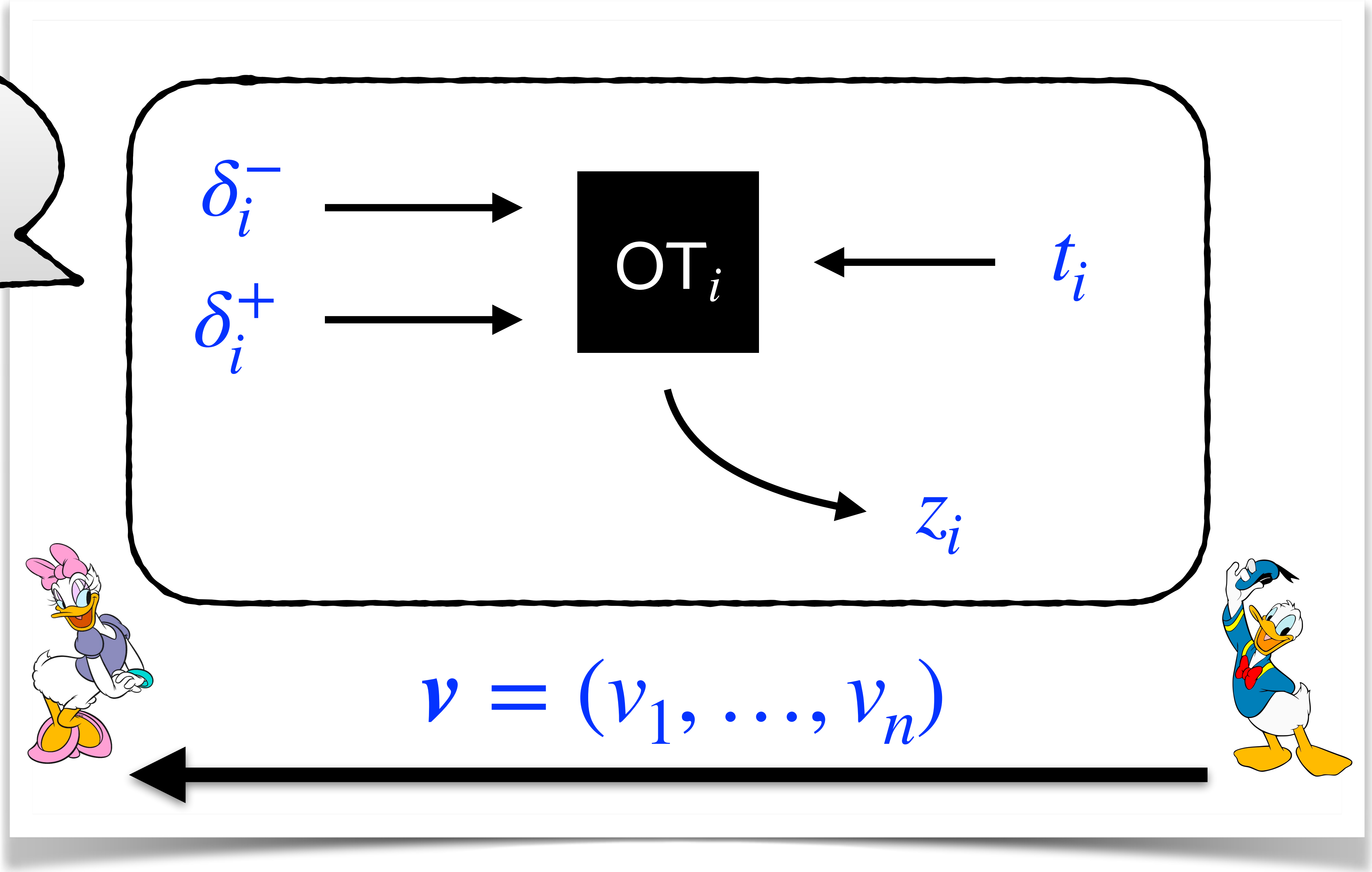
$\delta_j^+ - \delta_j^- \neq \delta_k^+ - \delta_k^-$
for $j \neq k$



- Protocol is fully secure against $\mathscr{P}_2^{\mathscr{A}}$
- $\mathscr{P}_1^{\mathscr{A}}$ can use inconsistent inputs in OT

Security

$\delta_j^+ - \delta_j^- \neq \delta_k^+ - \delta_k^-$
for $j \neq k$



$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

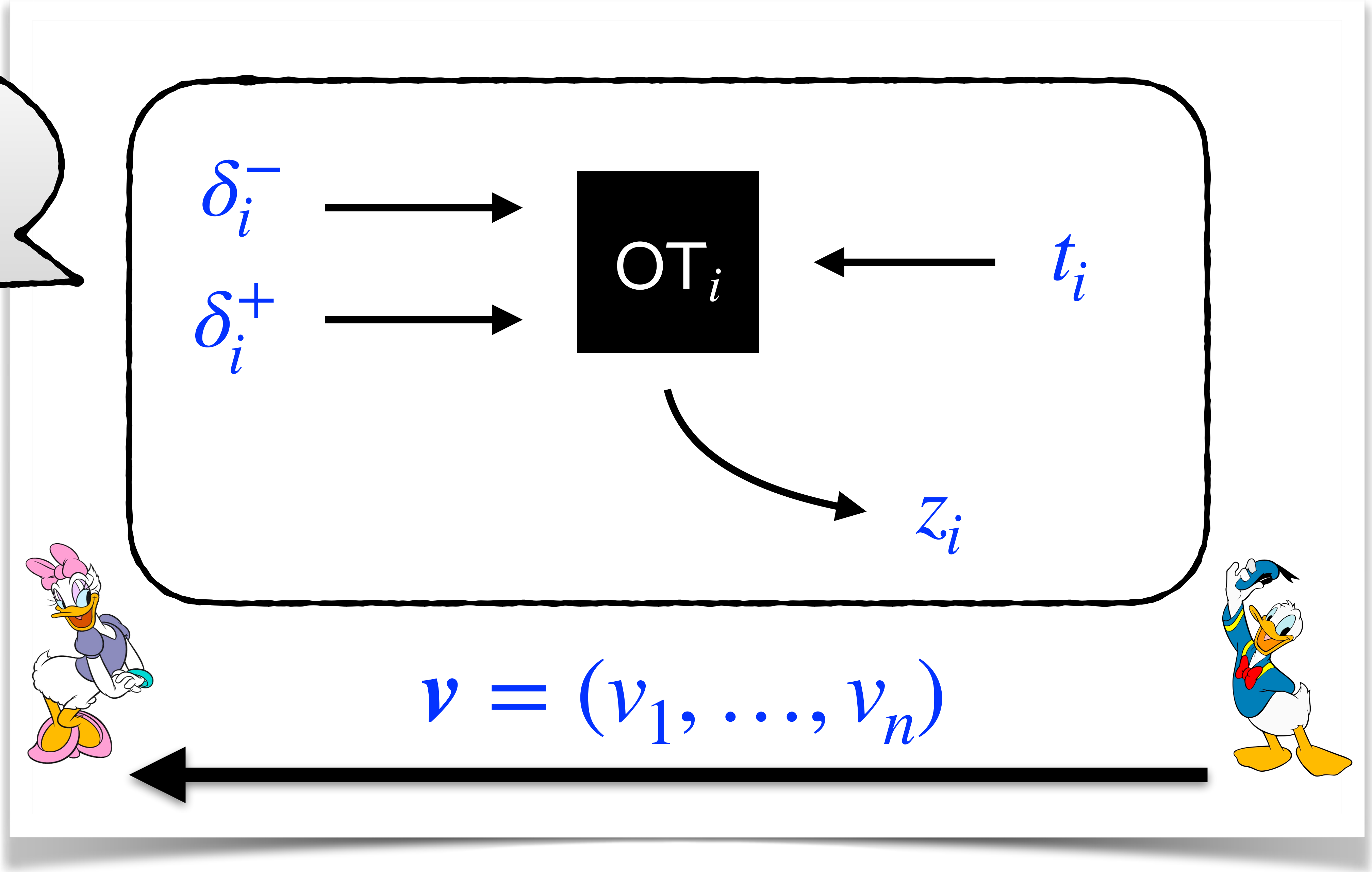
- Protocol is fully secure against $\mathscr{P}_2^{\mathscr{A}}$
- $\mathscr{P}_1^{\mathscr{A}}$ can use inconsistent inputs in OT
  - Output is *offset* by $\langle \boldsymbol{v} * \boldsymbol{t}, \boldsymbol{d} \rangle$ for $\boldsymbol{d}$ smallest vec in

$$\{\gamma \cdot \mathbf{1} - (\boldsymbol{\delta}^+ - \boldsymbol{\delta}^-)/2 \ \text{ s.t. } \gamma \in \mathbb{Z}_q\}$$

Security

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$\delta_j^+ - \delta_j^- \neq \delta_k^+ - \delta_k^-$
for $j \neq k$



- Protocol is fully secure against $\mathscr{P}_2^{\mathscr{A}}$
- $\mathscr{P}_1^{\mathscr{A}}$ can use inconsistent inputs in OT
  - Output is *offset* by $\langle \boldsymbol{v} * \boldsymbol{t}, \boldsymbol{d} \rangle$ for $\boldsymbol{d}$ smallest vec in

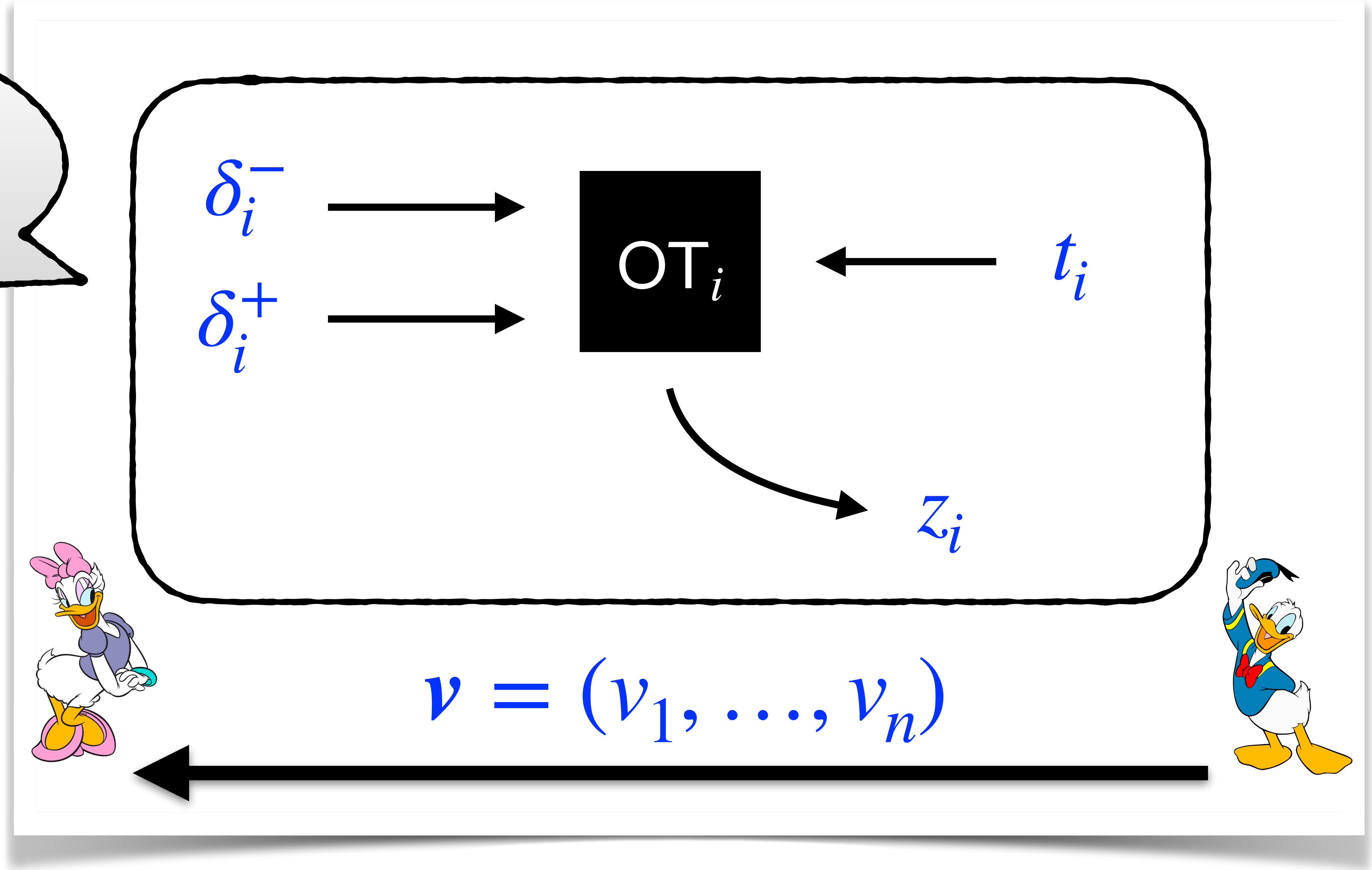
$$\{\gamma \cdot \mathbf{1} - (\boldsymbol{\delta}^+ - \boldsymbol{\delta}^-)/2 \ \text{ s.t. } \gamma \in \mathbb{Z}_q\}$$

Weight($\boldsymbol{d}$)
captures how much $\mathscr{A}$ deviates.

Security

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively

$\delta_j^+ - \delta_j^- \neq \delta_k^+ - \delta_k^-$
for $j \neq k$



- Protocol is fully secure against $\mathscr{P}_2^{\mathscr{A}}$
- $\mathscr{P}_1^{\mathscr{A}}$ can use inconsistent inputs in OT
  - Output is *offset* by $\langle \boldsymbol{v} * \boldsymbol{t}, \boldsymbol{d} \rangle$ for $\boldsymbol{d}$ smallest vec in

    $$\{\gamma \cdot \mathbf{1} - (\boldsymbol{\delta}^+ - \boldsymbol{\delta}^-)/2 \text{ s.t. } \gamma \in \mathbb{Z}_q\}$$

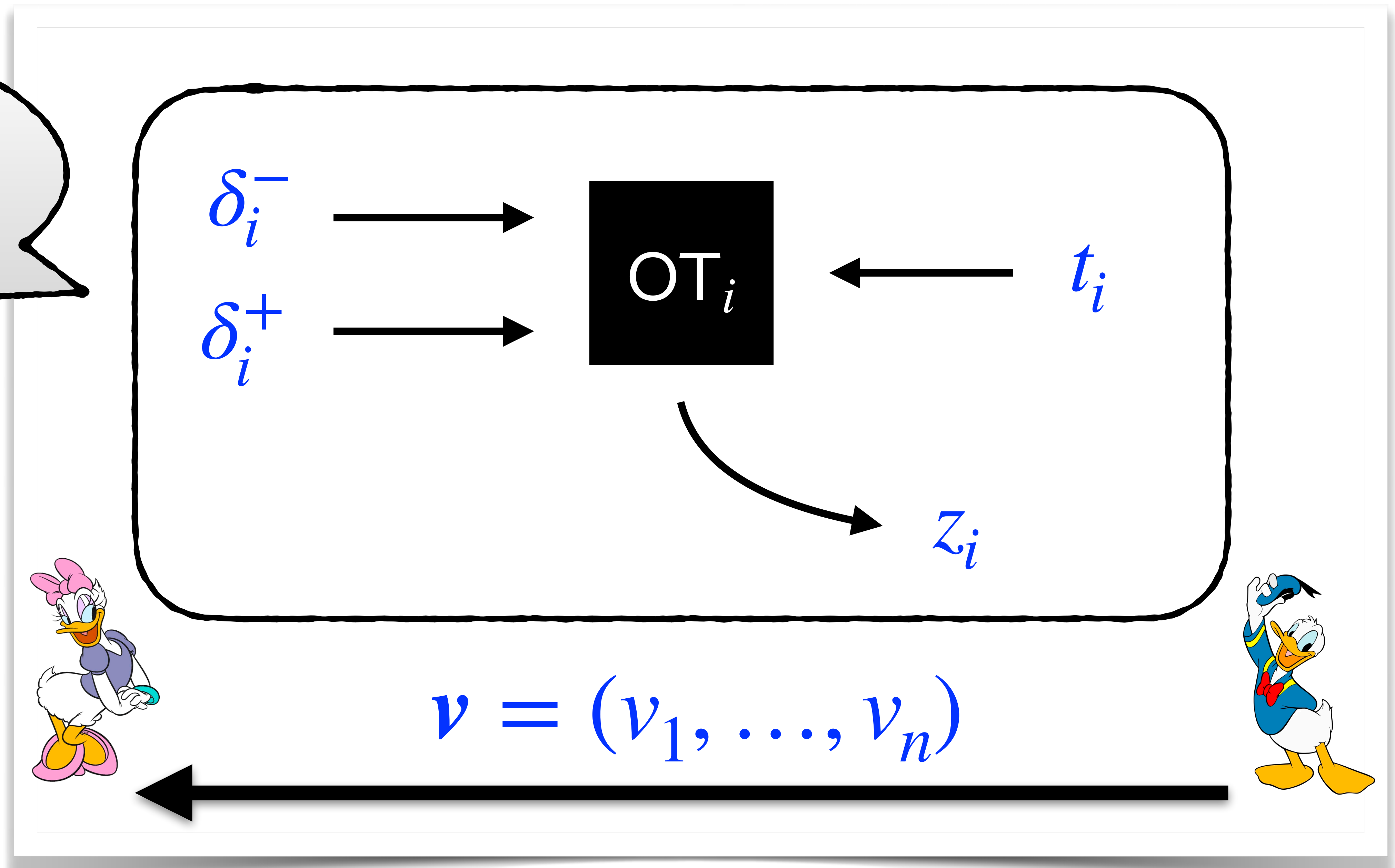
  - If $\boldsymbol{d}$ is close to zero then $\langle \boldsymbol{v} * \boldsymbol{t}, \boldsymbol{d} \rangle$ and $y = \langle \boldsymbol{t}, \boldsymbol{v} \rangle$ are $2^{-\kappa/4}$-close to ind.

$\mathsf{Weight}(\boldsymbol{d})$
captures how much $\mathscr{A}$ deviates.

Security

$\delta_j^+ - \delta_j^- \neq \delta_k^+ - \delta_k^-$
for $j \neq k$

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y \in \mathbb{Z}_q$ respectively



- Protocol is fully secure against $\mathscr{P}_2^{\mathscr{A}}$
- $\mathscr{P}_1^{\mathscr{A}}$ can use inconsistent inputs in OT
  - Output is *offset* by $\langle \boldsymbol{v} * \boldsymbol{t}, \boldsymbol{d} \rangle$ for $\boldsymbol{d}$ smallest vec in

    $$\{\gamma \cdot \mathbf{1} - (\boldsymbol{\delta}^+ - \boldsymbol{\delta}^-)/2 \ \text{ s.t. } \gamma \in \mathbb{Z}_q\}$$

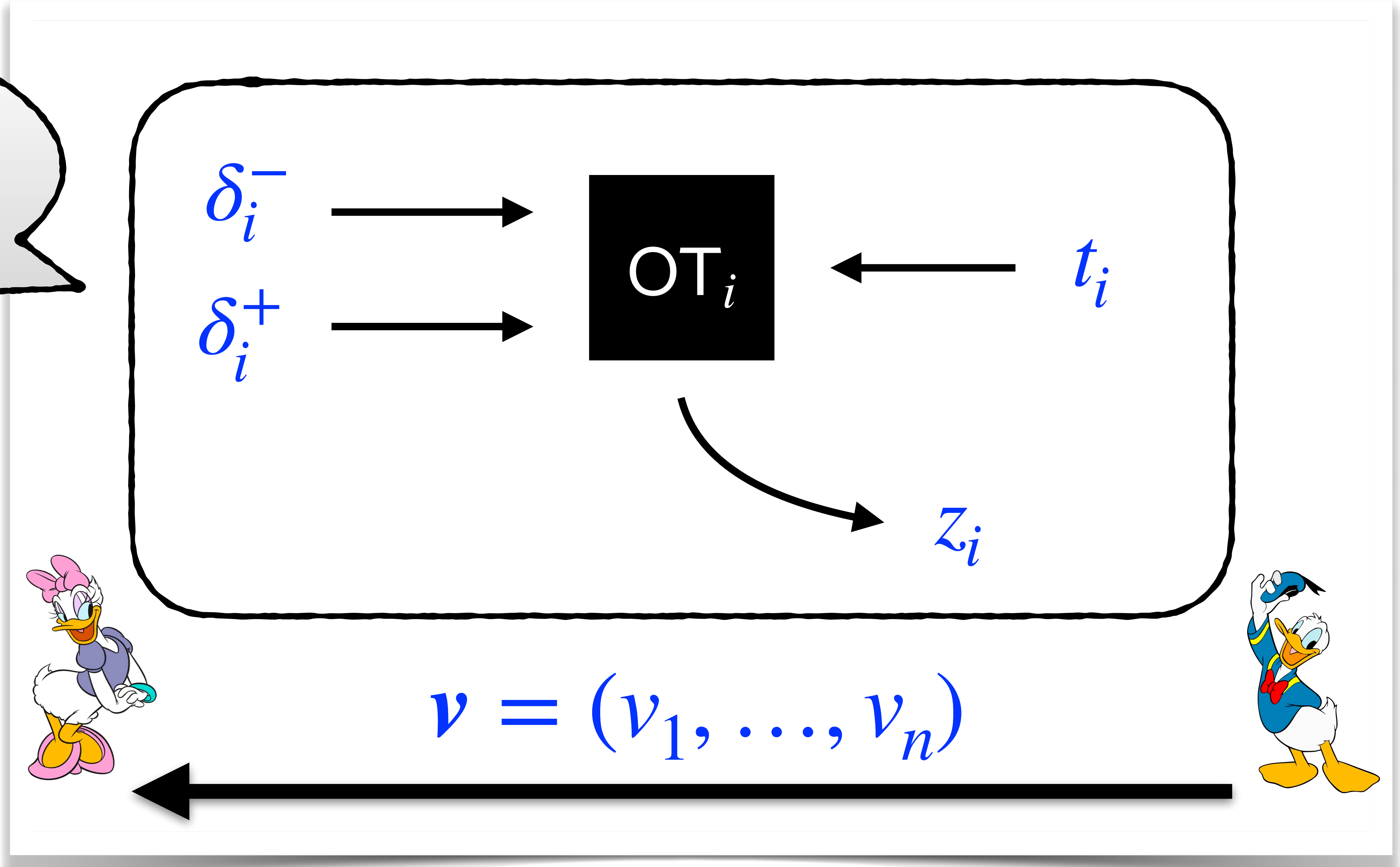
  - If $\boldsymbol{d}$ is close to zero then $\langle \boldsymbol{v} * \boldsymbol{t}, \boldsymbol{d} \rangle$ and $y = \langle \boldsymbol{t}, \boldsymbol{v} \rangle$ are $2^{-\kappa/4}$-close to ind.
  - If not, then $\langle \boldsymbol{v} * \boldsymbol{t}, \boldsymbol{d} \rangle$ has min-entropy $\kappa/4$, given $\boldsymbol{v}$ and y.

$\mathsf{Weight}(\boldsymbol{d})$
captures how much $\mathscr{A}$ deviates.

Security

$\mathscr{P}_1$ *and* $\mathscr{P}_2$ *hold inputs* x *and* $y \in \mathbb{Z}_q$ *respectively*

$\delta_j^+ - \delta_j^- \neq \delta_k^+ - \delta_k^-$
for $j \neq k$



- Protocol is fully secure against $\mathscr{P}_2^{\mathscr{A}}$
- $\mathscr{P}_1^{\mathscr{A}}$ can use inconsistent inputs in OT
  - Output is *offset* by $\langle \boldsymbol{v} * \boldsymbol{t}, \boldsymbol{d} \rangle$ for $\boldsymbol{d}$ smallest vec in

    $$\{\gamma \cdot \mathbf{1} - (\boldsymbol{\delta}^+ - \boldsymbol{\delta}^-)/2 \text{ s.t. } \gamma \in \mathbb{Z}_q\}$$

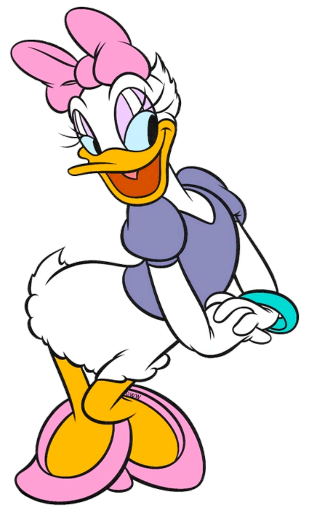
  - If $\boldsymbol{d}$ is close to zero then $\langle \boldsymbol{v} * \boldsymbol{t}, \boldsymbol{d} \rangle$ and $y = \langle \boldsymbol{t}, \boldsymbol{v} \rangle$ are $2^{-\kappa/4}$-close to ind.
  - If not, then $\langle \boldsymbol{v} * \boldsymbol{t}, \boldsymbol{d} \rangle$ has min-entropy $\kappa/4$, given $\boldsymbol{v}$ and y.

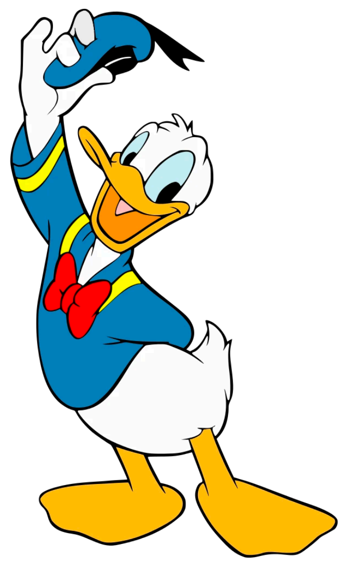
Weight($\boldsymbol{d}$)
captures how much $\mathscr{A}$ deviates.

3rd moment concentration inequality

Batching Variant of our Protocol

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y_1, \ldots, y_\beta \in \mathbb{Z}_q$ respectively





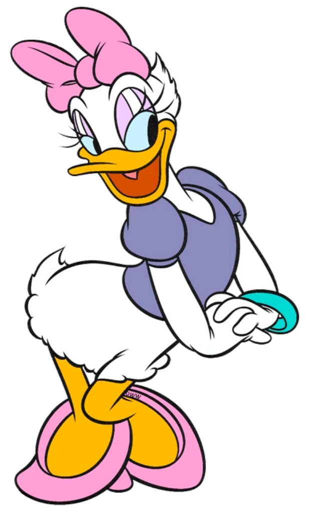
1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

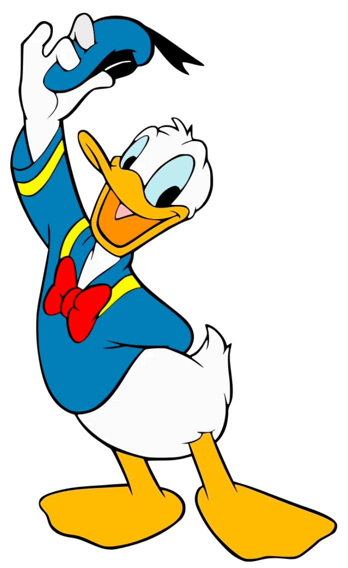
1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.

Batching Variant of our Protoc

$n = \beta \cdot \log(q) + \kappa$
calls to OT (in parallel)

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y_1, \ldots, y_\beta \in \mathbb{Z}_q$ respectively





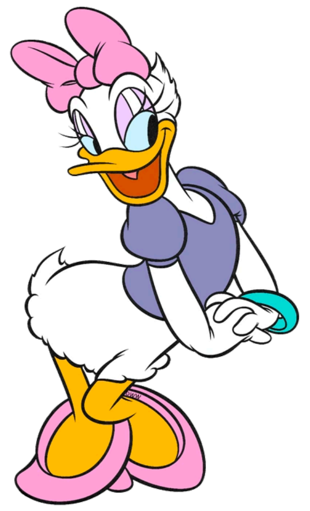
1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

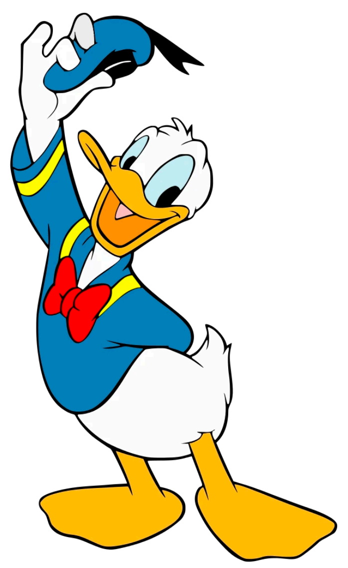
1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.

Batching Variant of our Protoc

$n = \beta \cdot \log(q) + \kappa$
calls to OT (in parallel)

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y_1, \ldots, y_\beta \in \mathbb{Z}_q$ respectively





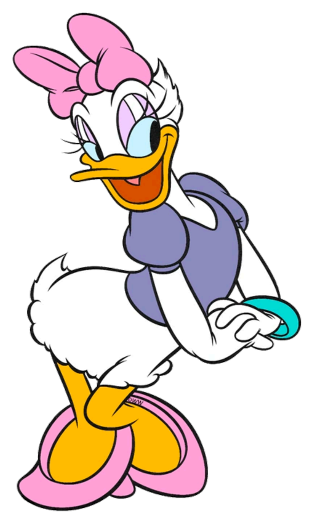
1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.
2. $\forall j$, sample $\boldsymbol{v}_j \leftarrow \mathbb{Z}_q^n$ s.t. $\langle \boldsymbol{t}, \boldsymbol{v}_j \rangle = y_j$.

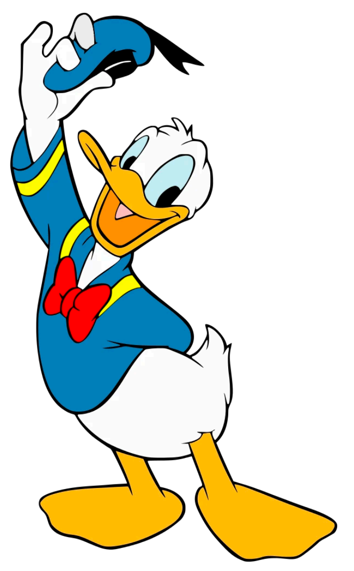
Batching Variant of our Protoc

$\mathscr{P}_1$ and $\mathscr{P}_2$ hold inputs x and $y_1, \ldots, y_\beta \in \mathbb{Z}_q$ respectively

$n = \beta \cdot \log(q) + \kappa$
calls to OT (in parallel)



$v_1, v_2, \ldots, v_\beta$



1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.

2. $\forall j$, sample $\boldsymbol{v}_j \leftarrow \mathbb{Z}_q^n$ s.t. $\langle \boldsymbol{t}, \boldsymbol{v}_j \rangle = y_j$.

Batching Variant of our Protoc

$\mathscr{P}_1$ *and* $\mathscr{P}_2$ *hold inputs* x *and* $y_1, \ldots, y_\beta \in \mathbb{Z}_q$ *respectively*

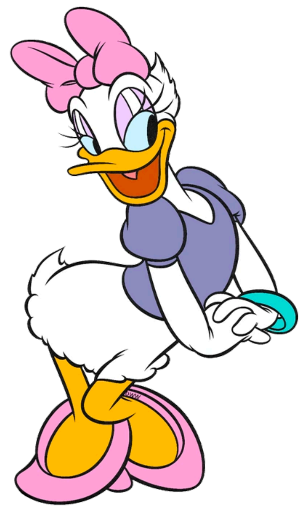
$n = \beta \cdot \log(q) + \kappa$
calls to OT (in parallel)

… OT_{i-1} …

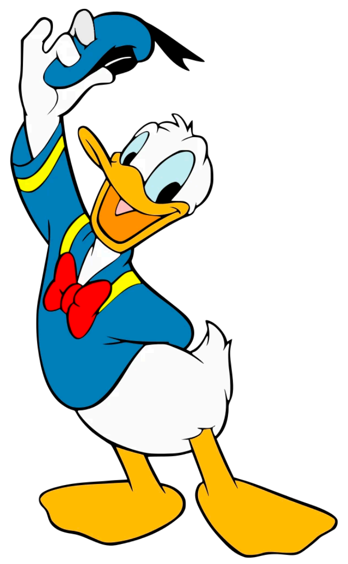
$\delta_i^- \longrightarrow \mathrm{OT}_i \longleftarrow t_i$

$\delta_i^+ \longrightarrow \mathrm{OT}_i \longrightarrow z_i$

OT_{i+1} …



$\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_\beta$ ←



1. Sample $\boldsymbol{\delta} \leftarrow \mathbb{Z}_q^n$ and set $\begin{cases} \delta_i^+ = \delta_i + x \\ \delta_i^- = \delta_i - x \end{cases}$

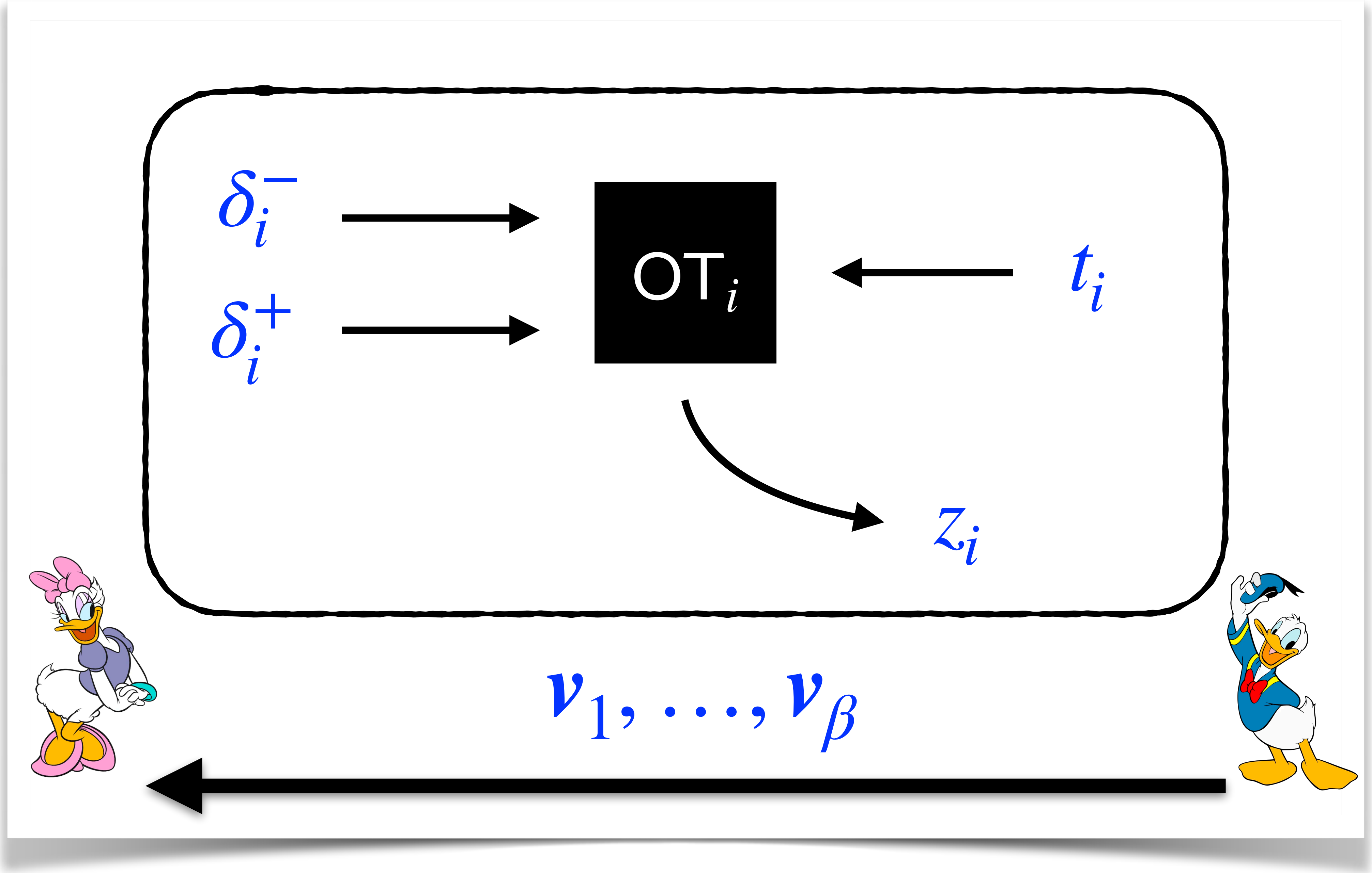
Output $\forall j, \langle -\boldsymbol{\delta}, \boldsymbol{v}_j \rangle$.

1. Sample $\boldsymbol{t} \leftarrow \{-1,1\}^n$.
2. $\forall j$, sample $\boldsymbol{v}_j \leftarrow \mathbb{Z}_q^n$ s.t. $\langle \boldsymbol{t}, \boldsymbol{v}_j \rangle = y_j$.

Output $\forall j, \langle \boldsymbol{z}, \boldsymbol{v}_j \rangle$.

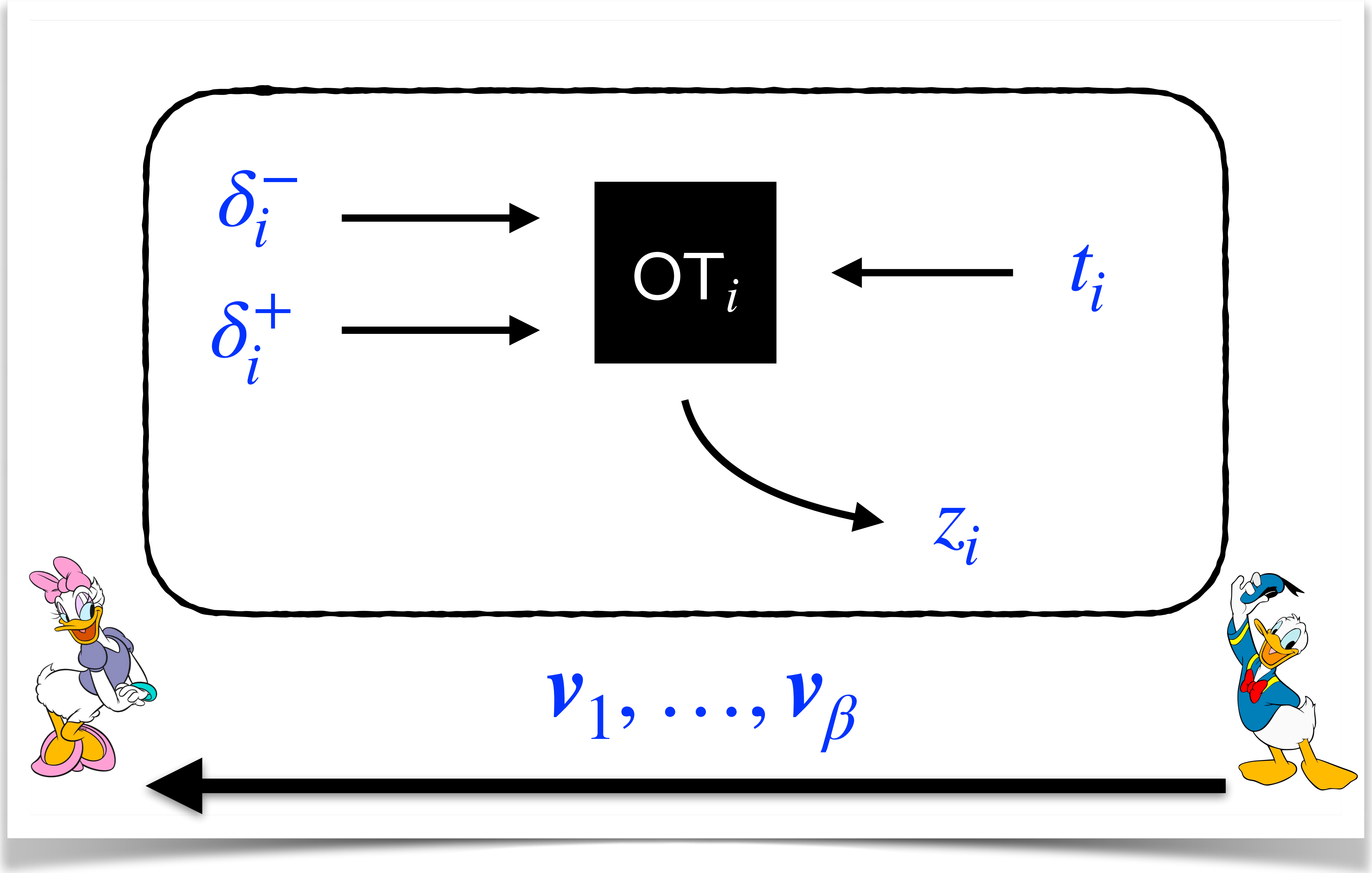
Summary

New OT-based two-party mult. protocol



Summary

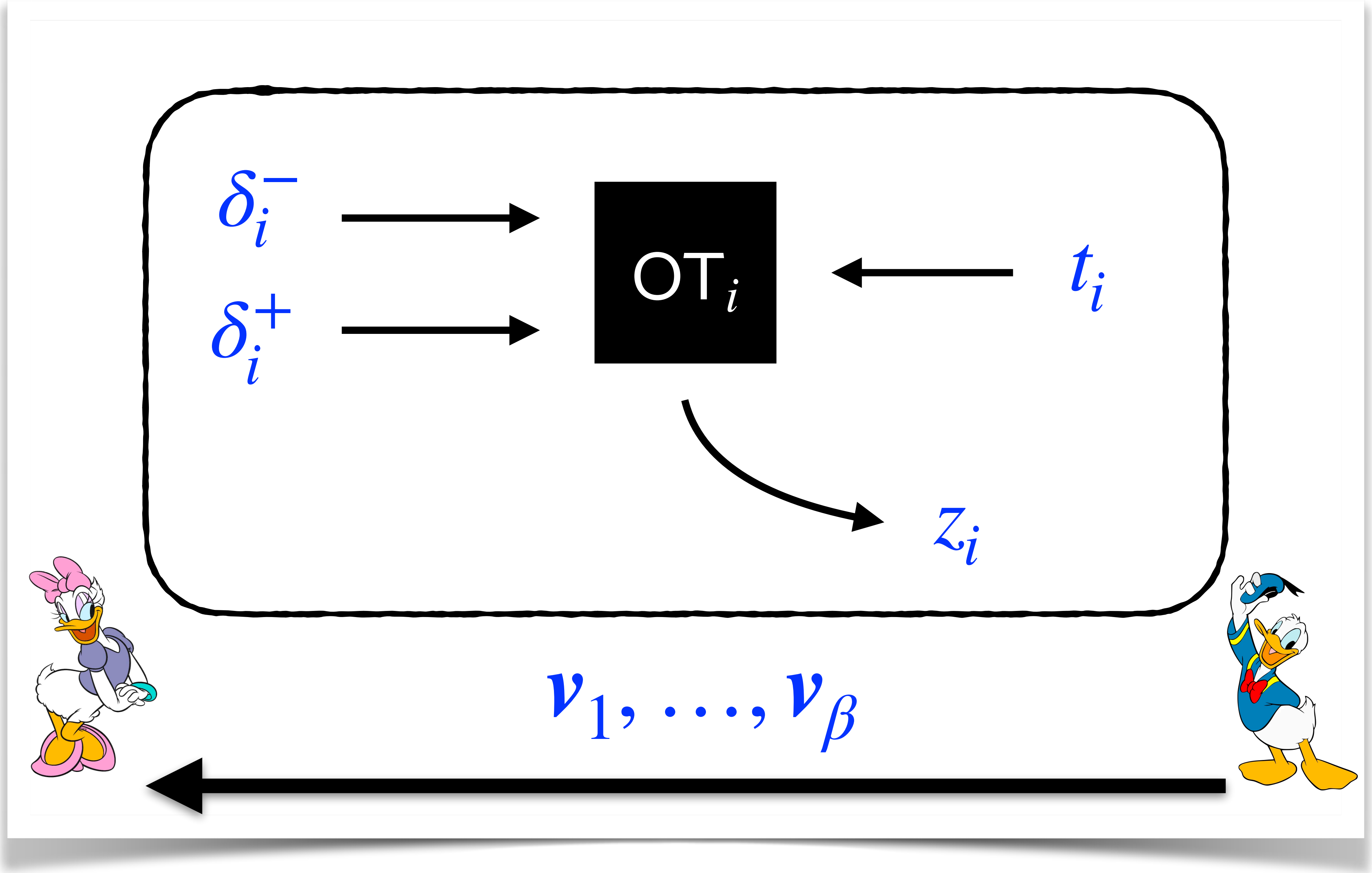
New OT-based two-party mult. protocol



1. Our Protocol
    - "Sufficiently" secure for some applications
    - "Almost" as efficient as SoA semi-honest protocols
    - x2 more efficient than SoA in its fully malicious variant

Summary

New OT-based two-party mult. protocol



1. Our Protocol
   - "Sufficiently" secure for some applications
   - "Almost" as efficient as SoA semi-honest protocols
   - x2 more efficient than SoA in its fully malicious variant
2. Open Questions
   - Push efficiency further (go beyond Gilboa's $\log(q) \times \log(q)$ barrier)
   - Does IPS09 realize WeakMult?
   - Lower Bounds? *Is OT-mult. inherently wasteful communication-wise?*

Highly Efficient OT-Based Multiplication Protocols

Nikolaos Makriyannis (Fireblocks)

Joint work w/ **Iftach Haitner**, **Samuel Ranellucci** & **Eliad Tsfadia**