CS670: Cryptographic Techniques for Privacy Preservation

Module 2: Secure Memory Access

Adithya Vadapalli

Drawings of Alice, Bob, and other crypto characters by Simon Oya: https://simonoya.com/drawings/

Memory Access

Alice

Bob

Alice has a lot of data.

However, she doesn’t have much space.

Bob has a lot of space, though! (Unfortunately, she does not trust him)

What can she do?

Memory Access on Secret Shares

Alice

$\overrightarrow{D}_0 \quad i^*_0 \quad M_0$

Hold shares of database: $\overrightarrow{D}$

Hold shares of index: i^*

Hold shares of a value: M

Bob

$\overrightarrow{D}_1 \quad i^*_1 \quad M_1$

Their goal: To obtain the shares of $D[\, i^* \,]$

Similarly, do a write operation: To obtain the shares of D'

Such that: $D'[i^*] = D[i^*] + M$

$D'[i] = D[i] \qquad i \neq i^*$

Secure Multiparty Computation

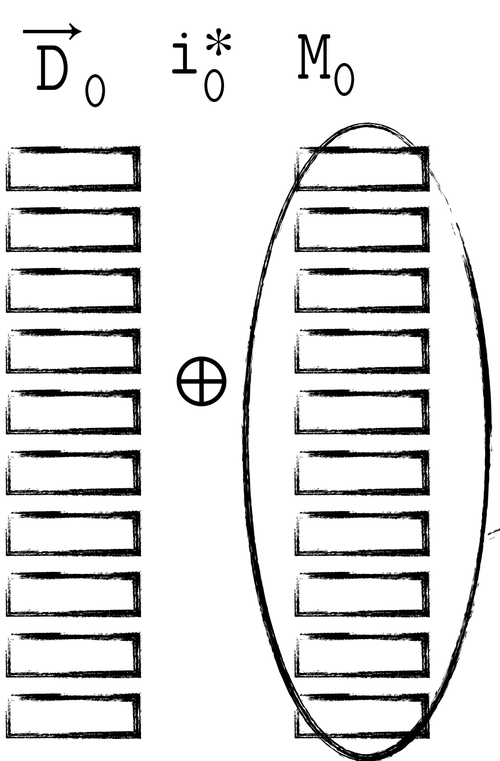
(but on a very specific task!)

Oblivious Write

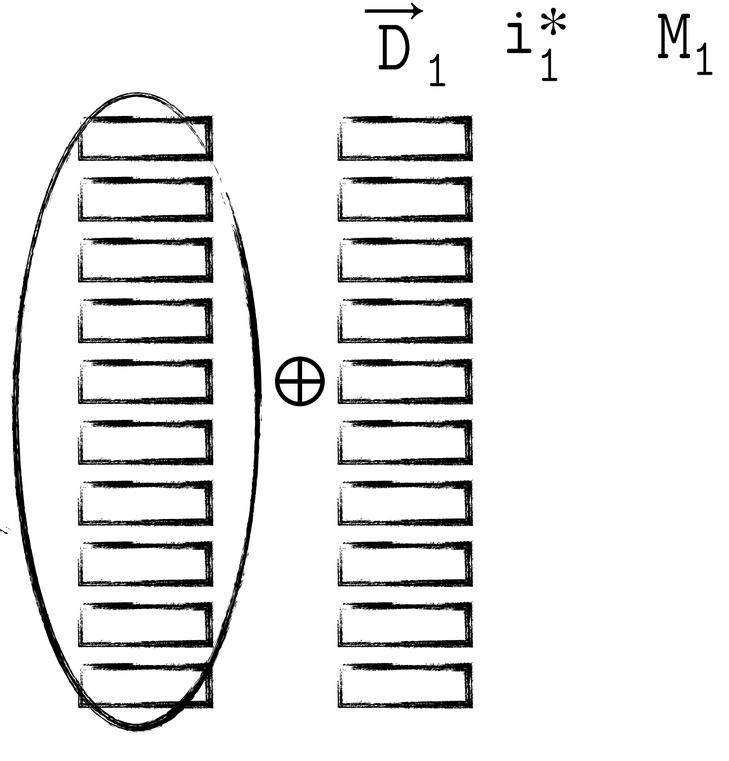
Alice

Bob

Any idea on how to do an oblivious write?



Shares of a standard-basis vector



Oblivious Write

Alice

$\overrightarrow{\mathtt{D}}_0 \quad \mathtt{i}^*_0 \quad \mathtt{M}_0$

So, what does the problem boil down to?

Bob

$\overrightarrow{\mathtt{D}}_1 \quad \mathtt{i}^*_1 \quad \mathtt{M}_1$

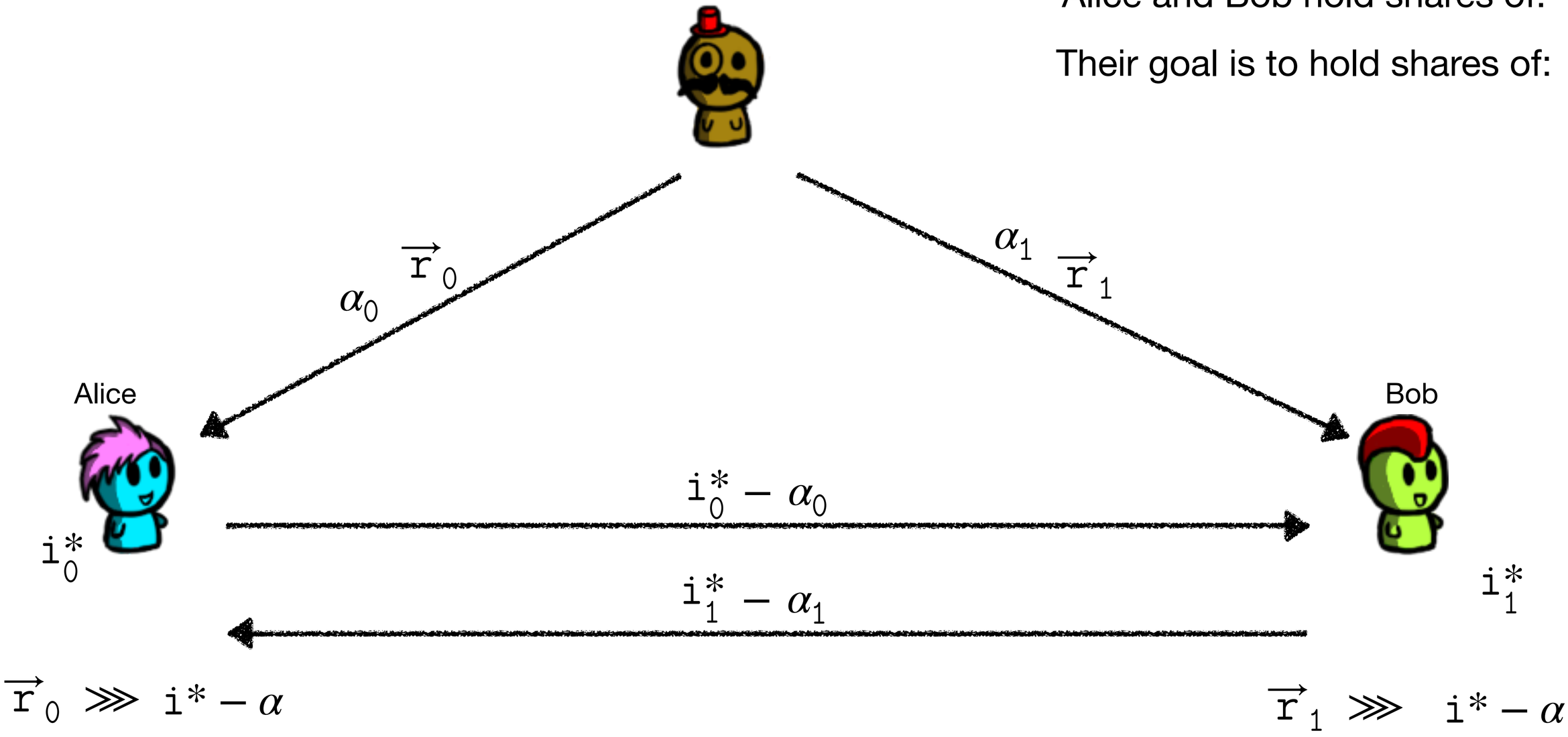
The Rotation Trick

$\vec{r}_0 + \vec{r}_1 = \vec{e}_\alpha$ → *Standard-basis vector at* α

$\alpha_0 + \alpha_1 = \alpha$ → *some random value*

Alice and Bob hold shares of: i^*

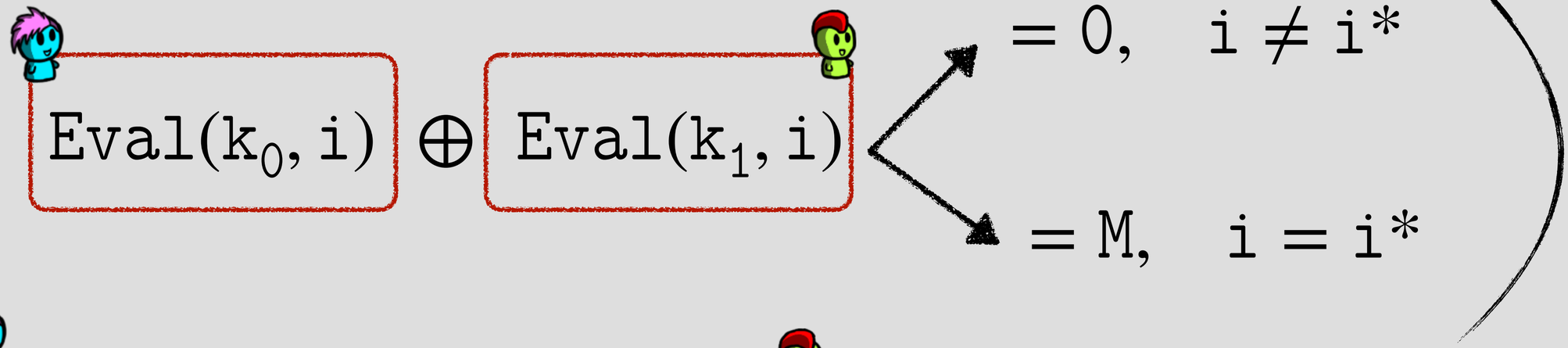
Their goal is to hold shares of: $\vec{e}_{i^*}$

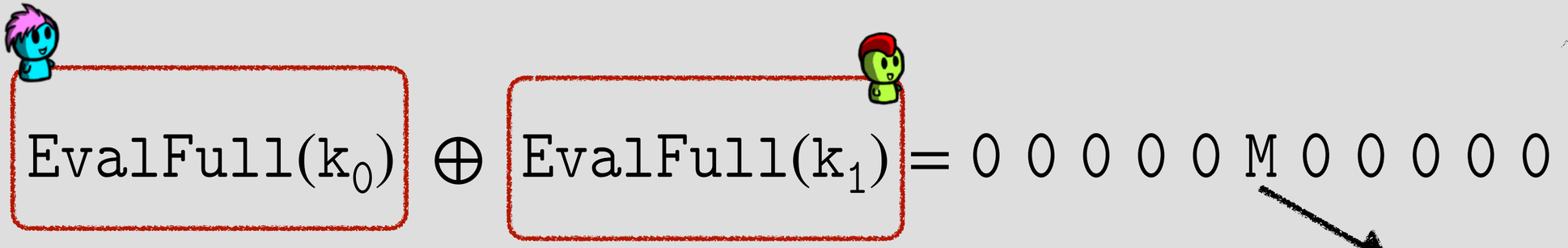


A Detour: Distributed Point Functions

The client (who is trying to distribute the shares)

$$(k_0, k_1) \leftarrow \mathtt{Gen}(i^*, M, 1^\lambda)$$





Oblivious Write

Alice

$\overrightarrow{D}_0$ i^*_0 M_0

Bob

$\overrightarrow{D}_1$ i^*_1 M_1

Do you have any ideas on how to solve this problem now?

The parties need to run an MPC on: DPF generation and evaluations

Back to the Detour

$$(\mathtt{k}_0, \mathtt{k}_1) \leftarrow \mathtt{Gen}(\mathtt{i}*, \mathtt{M}, 1^\lambda)$$

Some ideas on the Gen Function?

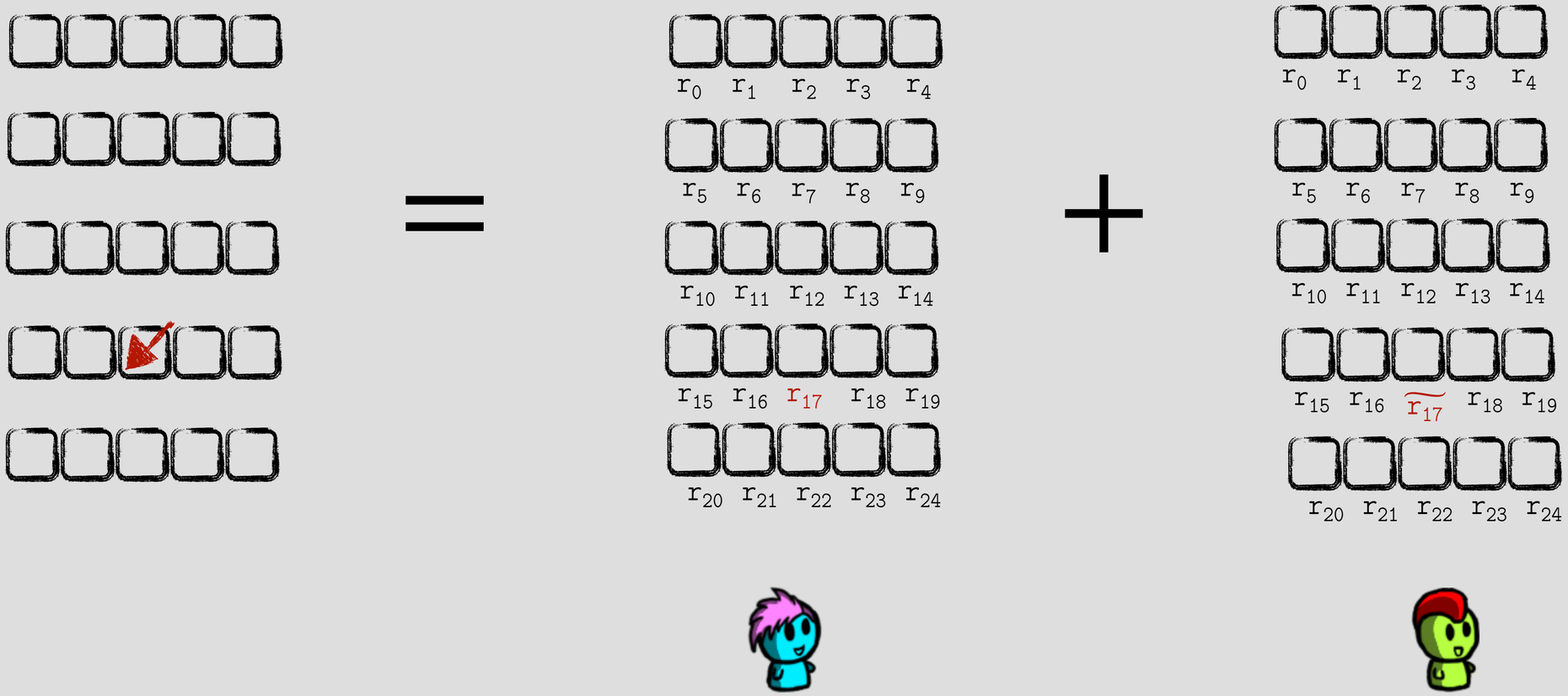
Any naive idea?

Can we get better than a naive

=

+

Can we get better than a naive



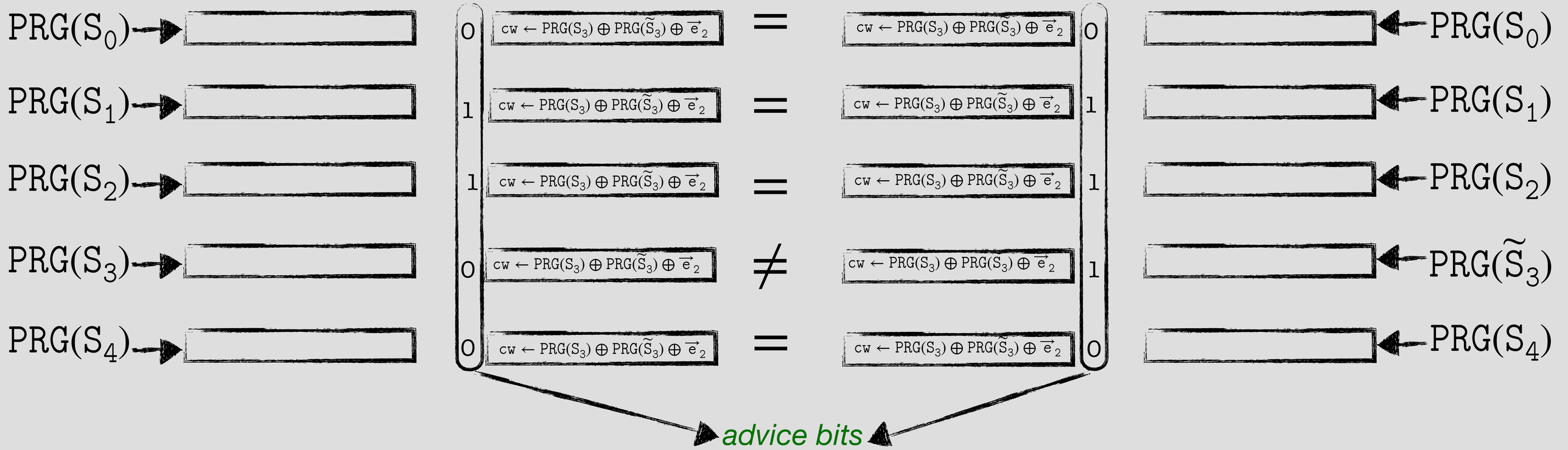
Back to the Detour

$$(k_0, k_1) \leftarrow \mathtt{Gen}(i^*, M, 1^\lambda)$$

This *almost* gives us what we want!

Why almost?

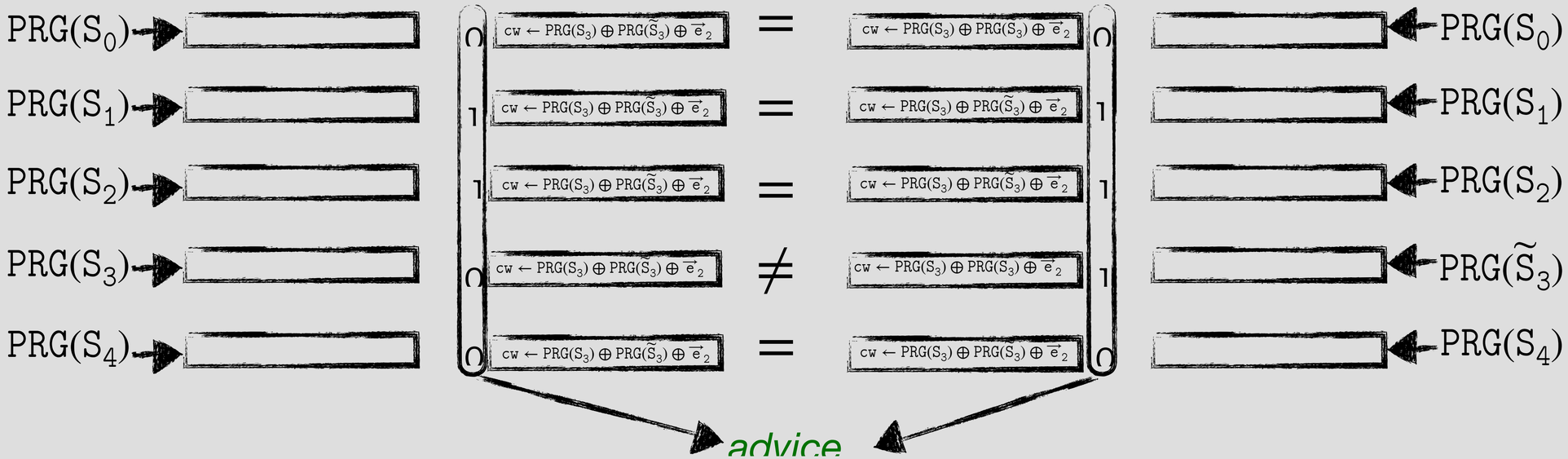
How can we fix it?



$$\mathtt{cw} \leftarrow \mathrm{PRG}(S_3) \oplus \mathrm{PRG}(\widetilde{S}_3) \oplus \vec{e}_2$$

Back to the Detour

$(k_0, k_1) \leftarrow \mathrm{Gen}(i^*, M, 1^\lambda)$



$\mathtt{cw} \leftarrow \mathrm{PRG}(S_3) \oplus \mathrm{PRG}(\widetilde{S}_3) \oplus \vec{e}_2$

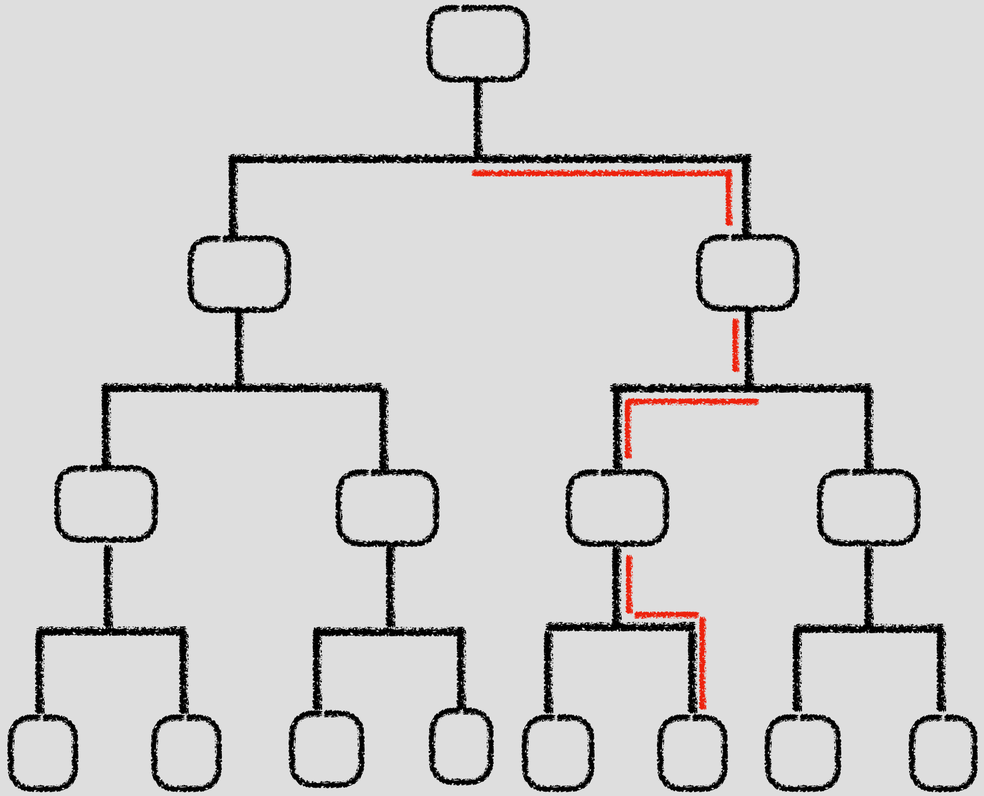
$k_0 = (s_0, s_1, s_2, s_3, s_4, \mathtt{cw}, b_0, b_1, b_2, b_3, b4)$

$k_1 = (s_0, s_1, s_2, \widetilde{s_3}, s_4, \mathtt{cw}, b_0, b_1, b_2, \widetilde{b_3}, b4)$

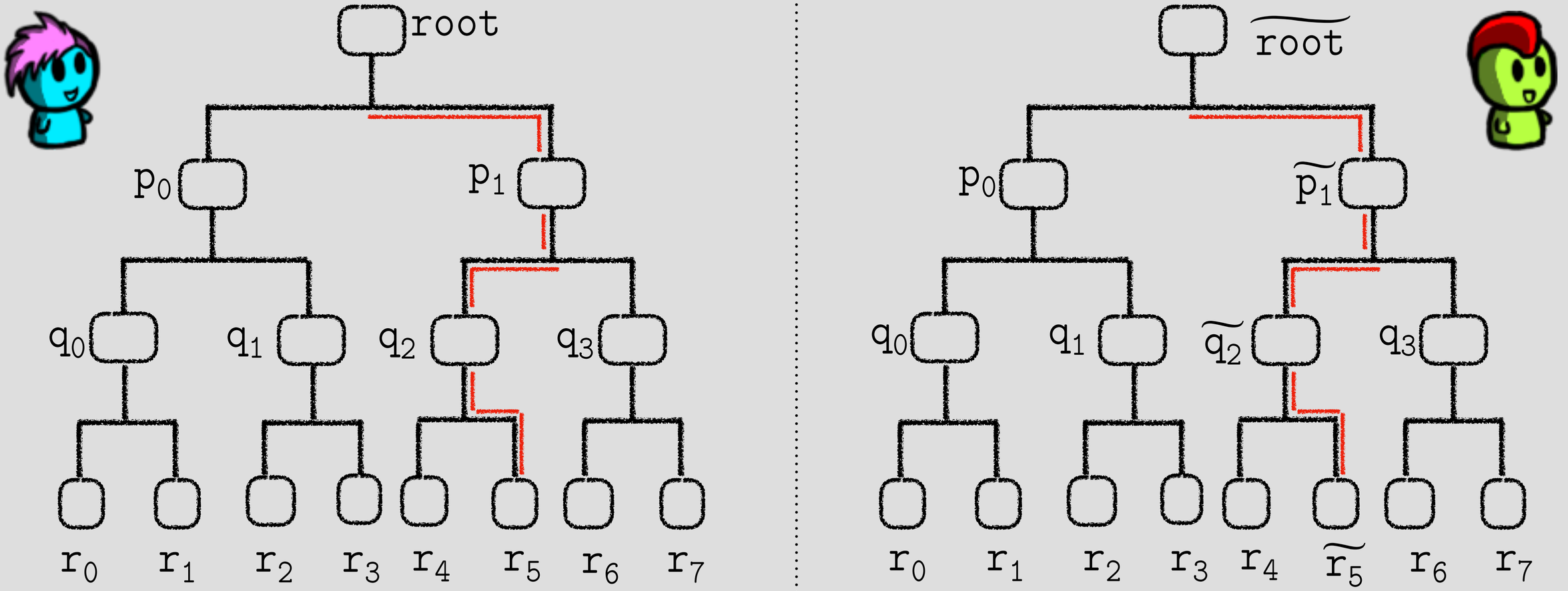
advice bits

Point Function as binary tree

101 = target point



Distributing a Point Function among two parties



101 = target point

The more efficient DPF

101 = target point

$scw \leftarrow s_0^{(1)} \oplus s_1^{(1)}$
$fcw_0 \leftarrow f_0^{(0)} \oplus f_1^{(0)}$
$fcw_1 \leftarrow f_0^{(1)} \oplus f_1^{(1)} \oplus 1$

$scw \leftarrow s_0^{(10)} \oplus s_1^{(10)}$
$fcw_0 \leftarrow f_0^{(11)} \oplus f_1^{(11)}$
$fcw_1 \leftarrow f_0^{(01)} \oplus f_1^{(01)} \oplus 1$

$scw \leftarrow s_0^{(101)} \oplus s_1^{(101)}$
$fcw_0 \leftarrow f_0^{(100)} \oplus f_1^{(100)}$
$fcw_1 \leftarrow f_0^{(101)} \oplus f_1^{(101)} \oplus 1$