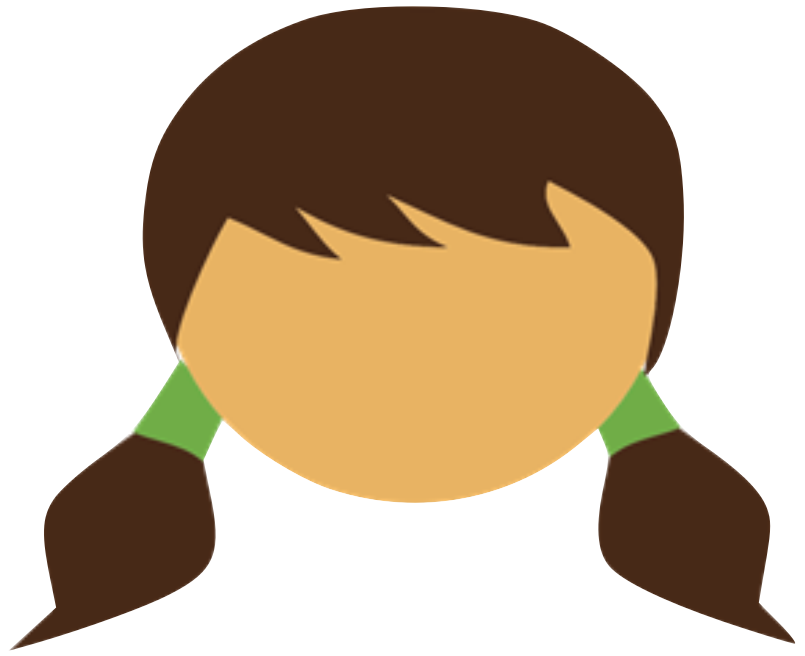
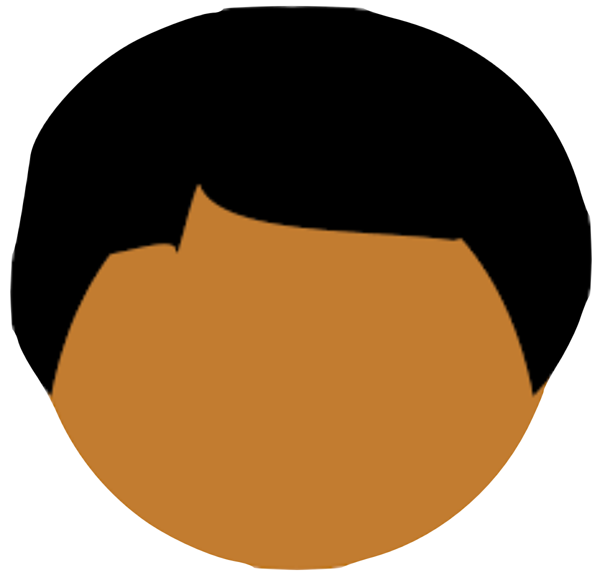
PSI

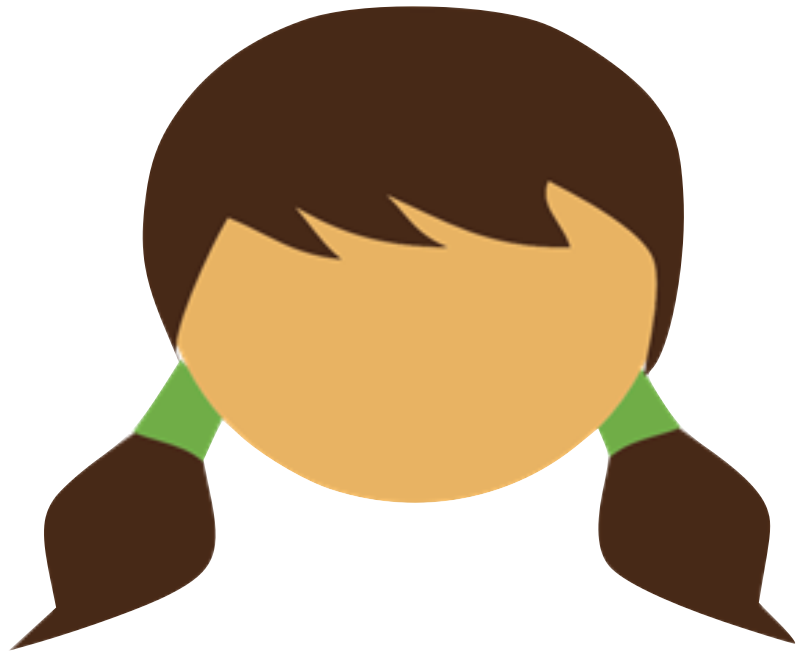


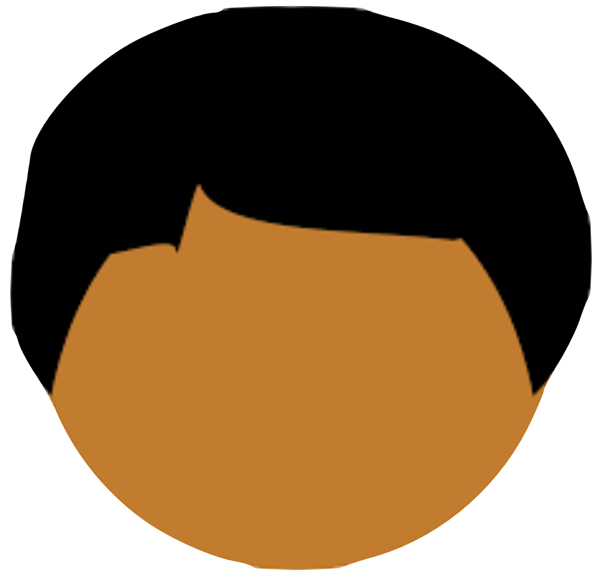


{13, 17, 25, 45, 52, 101}

{1, 4, 17, 19, 21, 45, 100}

PSI



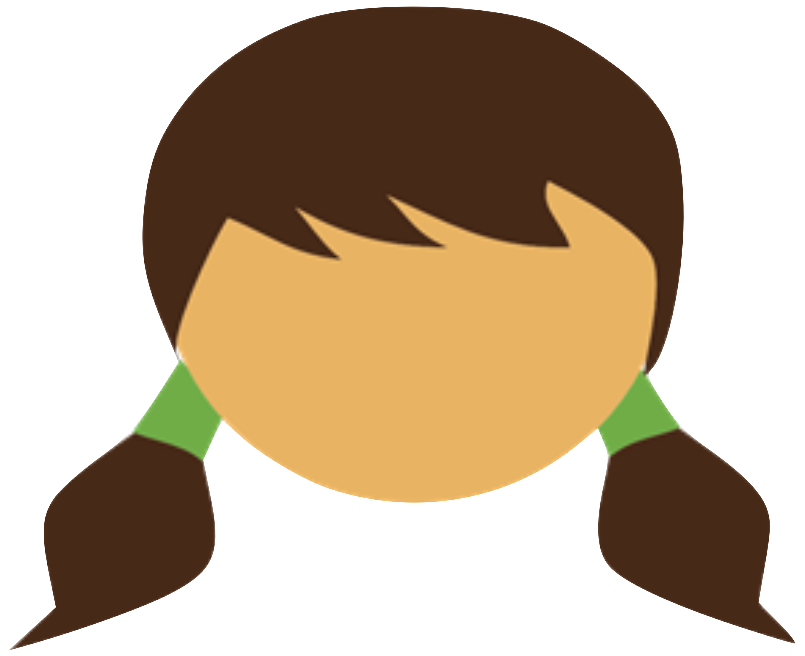


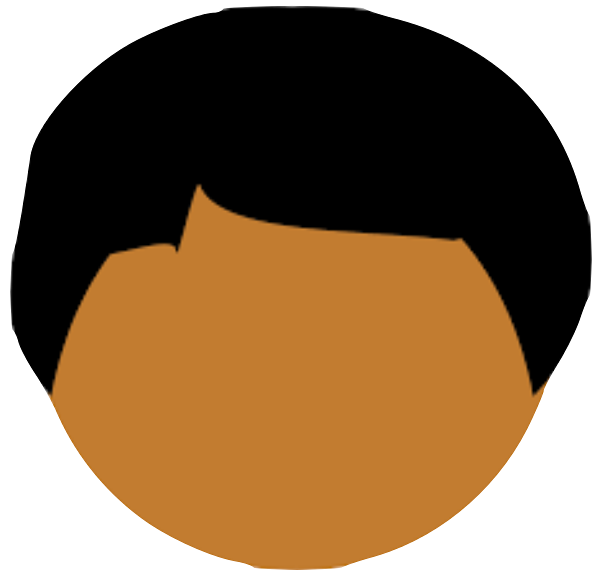
{13, **17**, 25, **45**, 52, 101}

{1, 4, **17**, 19, 21, **45**, 100}

{17,45}

PSI





{13, **17**, 25, **45**, 52, 101}

{1, 4, **17**, 19, 21, **45**, 100}

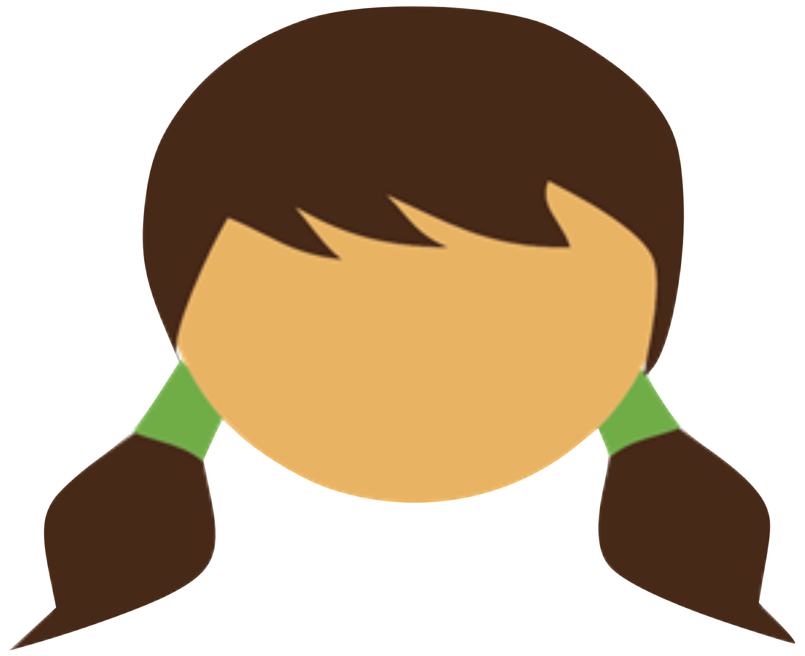
Special case of MPC

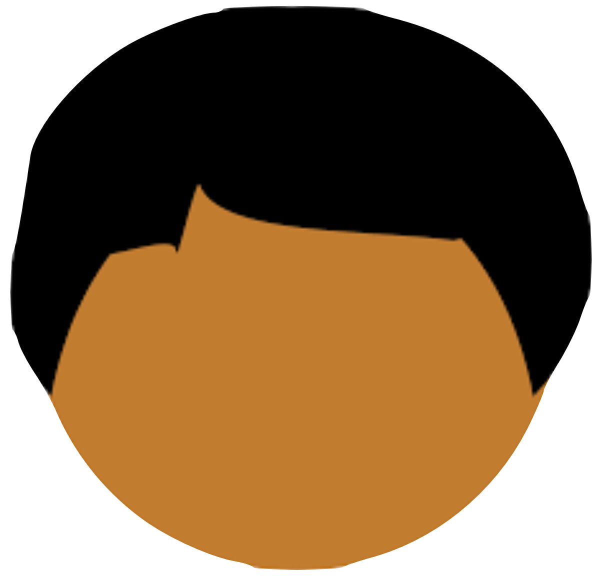
“Just use MPC”

{17,45}

Because it is a special case, we can hope for much more efficiency

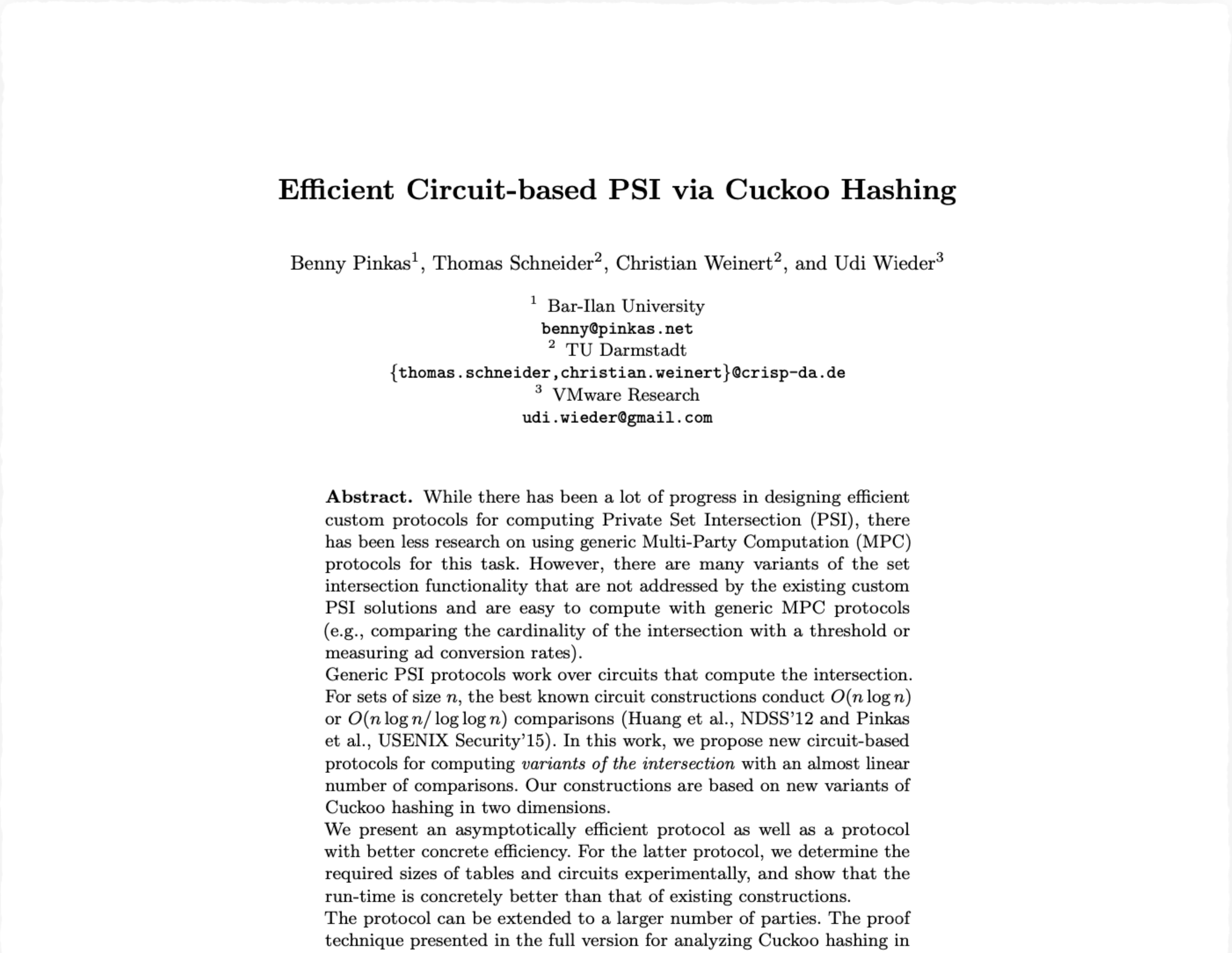
PSI





{13, **17**, 25, **45**, 52, 101}

{1, 4, **17**, 19, 21, **45**, 100}



Efficient Circuit-based PSI via Cuckoo Hashing

Benny Pinkas[1], Thomas Schneider[2], Christian Weinert[2], and Udi Wieder[3]

[1] Bar-Ilan University
benny@pinkas.net
[2] TU Darmstadt
{thomas.schneider,christian.weinert}@crisp-da.de
[3] VMware Research
udi.wieder@gmail.com

Abstract. While there has been a lot of progress in designing efficient custom protocols for computing Private Set Intersection (PSI), there has been less research on using generic Multi-Party Computation (MPC) protocols for this task. However, there are many variants of the set intersection functionality that are not addressed by the existing custom PSI solutions and are easy to compute with generic MPC protocols (e.g., comparing the cardinality of the intersection with a threshold or measuring ad conversion rates).

Generic PSI protocols work over circuits that compute the intersection. For sets of size n, the best known circuit constructions conduct $O(n \log n)$ or $O(n \log n / \log \log n)$ comparisons (Huang et al., NDSS'12 and Pinkas et al., USENIX Security'15). In this work, we propose new circuit-based protocols for computing *variants of the intersection* with an almost linear number of comparisons. Our constructions are based on new variants of Cuckoo hashing in two dimensions.

We present an asymptotically efficient protocol as well as a protocol with better concrete efficiency. For the latter protocol, we determine the required sizes of tables and circuits experimentally, and show that the run-time is concretely better than that of existing constructions.

The protocol can be extended to a larger number of parties. The proof technique presented in the full version for analyzing Cuckoo hashing in

↓

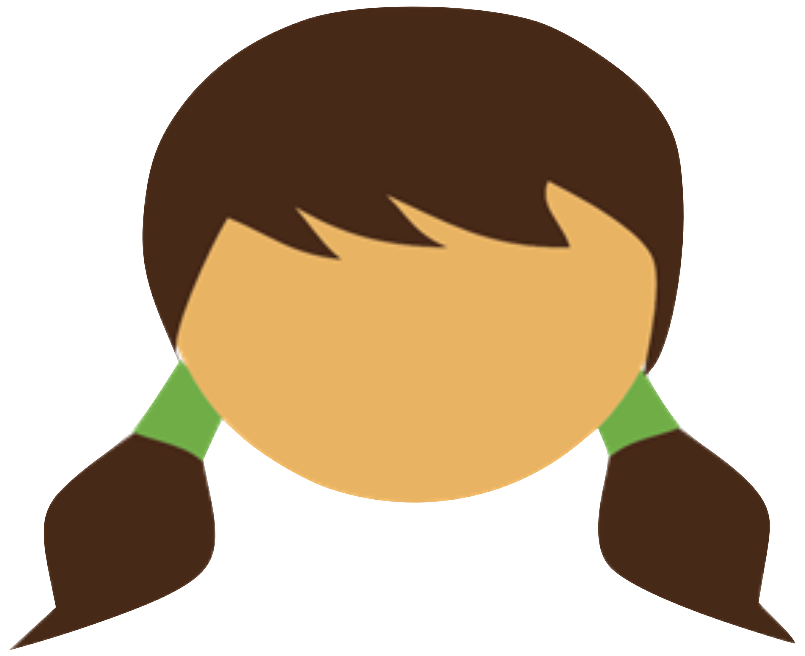
{17,45}

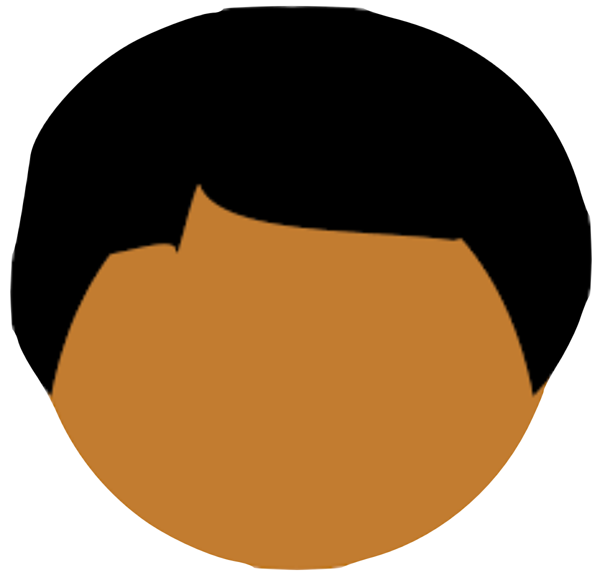
Special case of MPC

“Just use MPC”

Because it is a special case, we can hope for much more efficiency

PSI

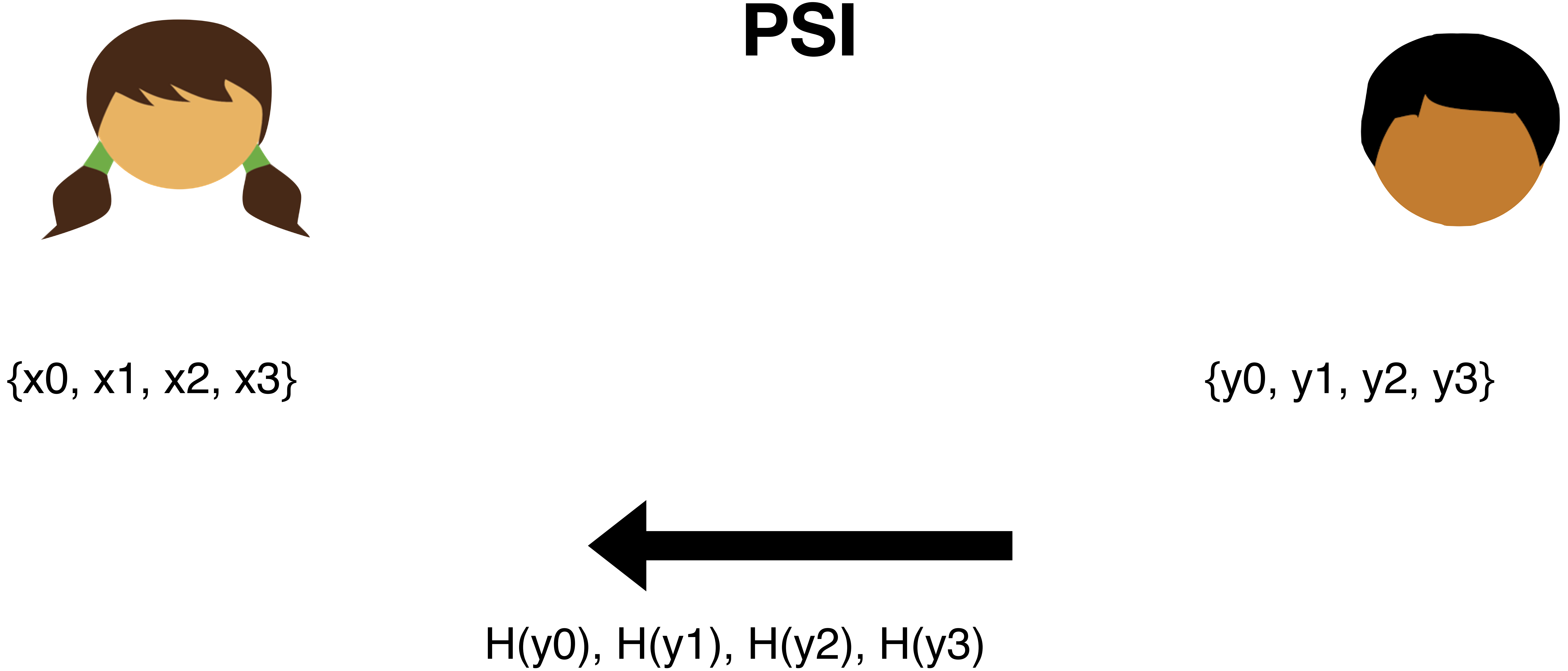




{x0, x1, x2, x3}

{y0, y1, y2, y3}

PSI

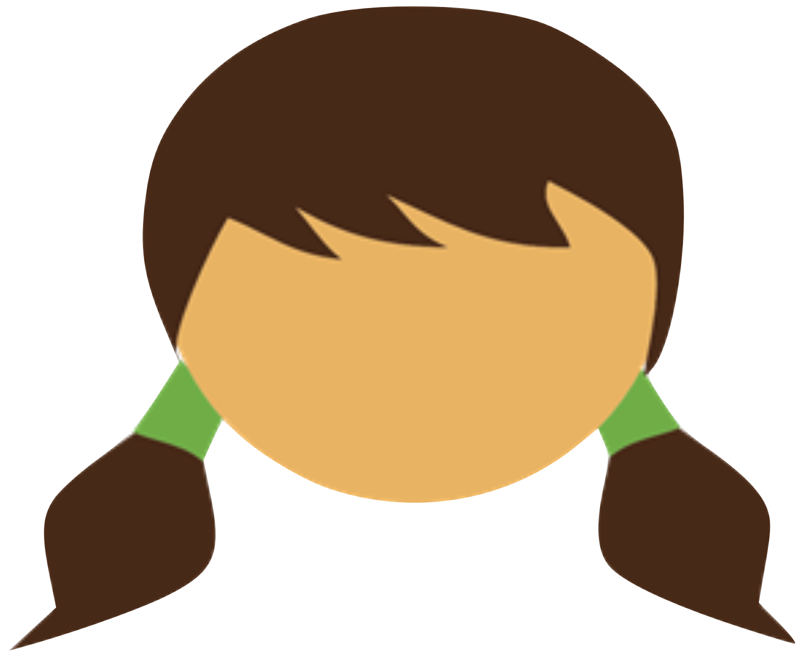


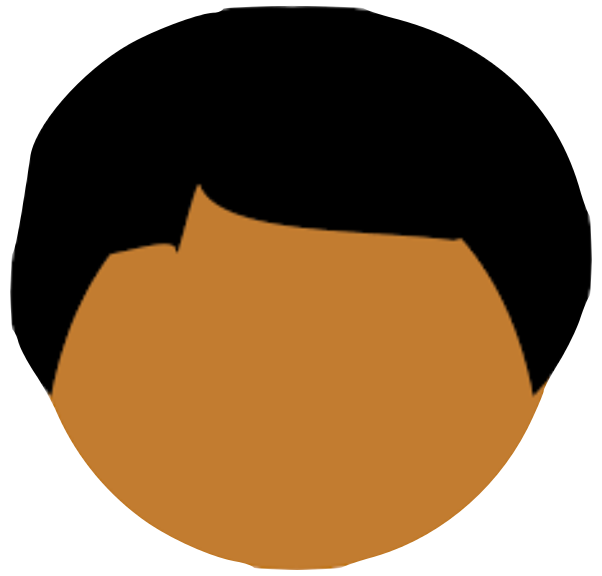
{x0, x1, x2, x3}

{y0, y1, y2, y3}

H(y0), H(y1), H(y2), H(y3)

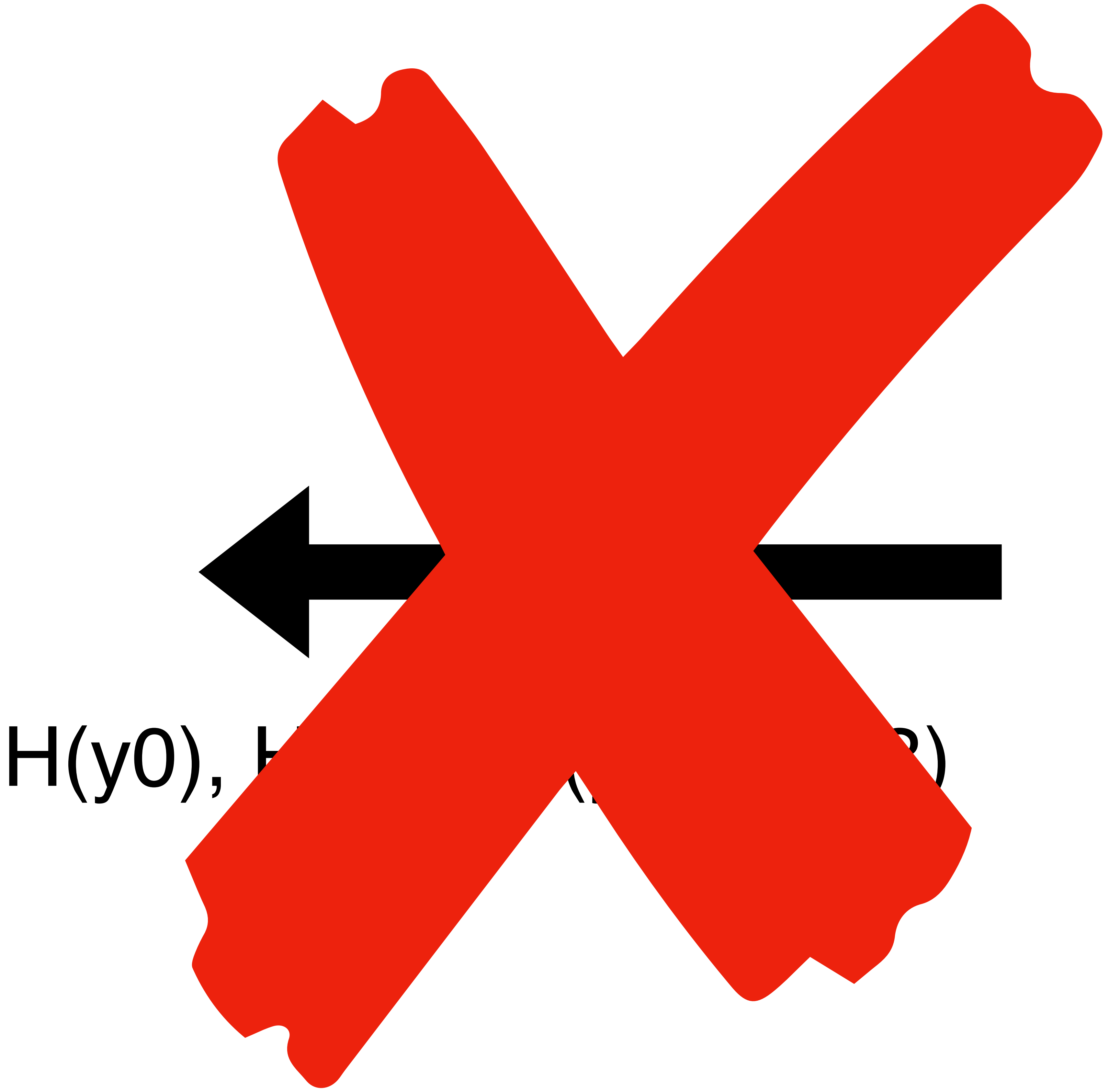
PSI





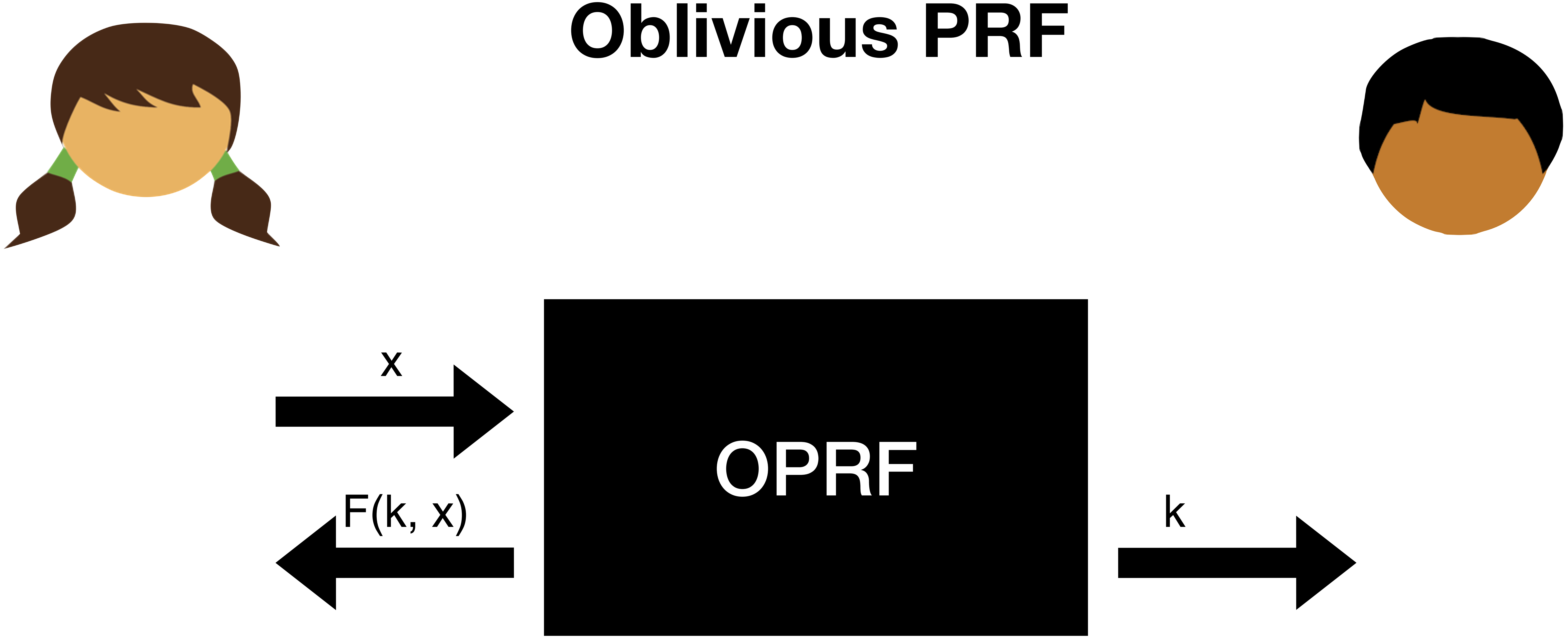
{x0, x1, x2, x3}

{y0, y1, y2, y3}



We cannot simulate semi-honest Alice

Oblivious PRF



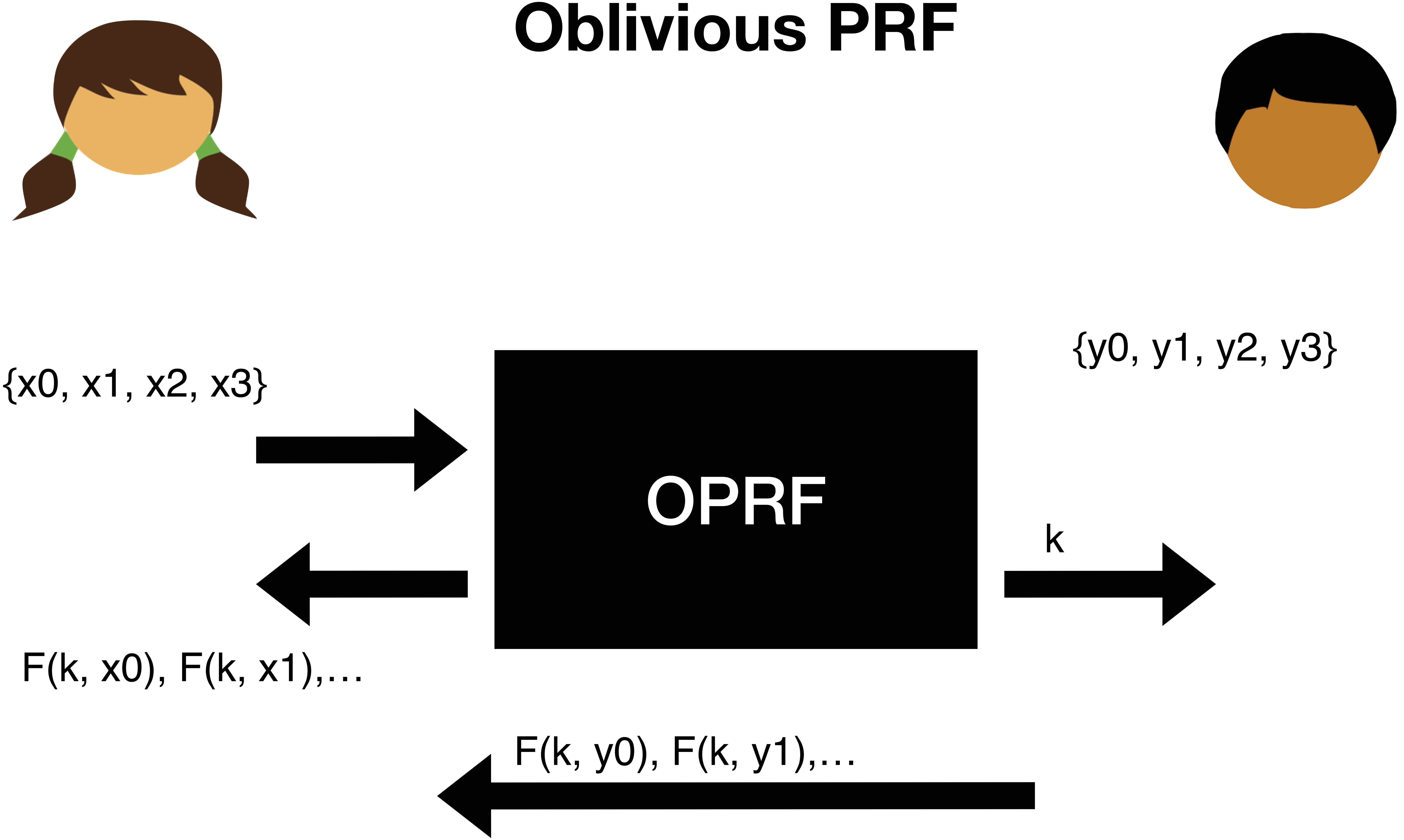
x

OPRF

F(k, x)

k

Oblivious PRF



{x0, x1, x2, x3}

{y0, y1, y2, y3}

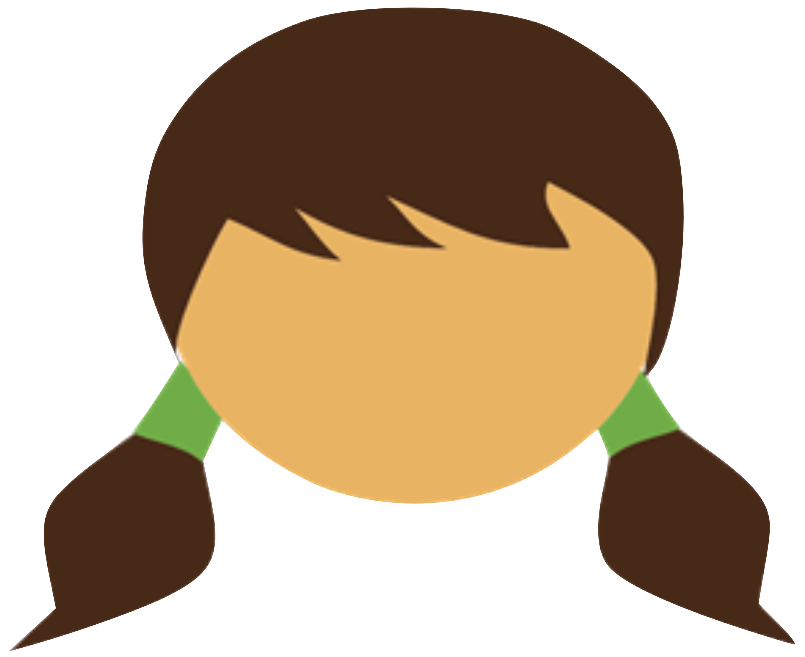
OPRF

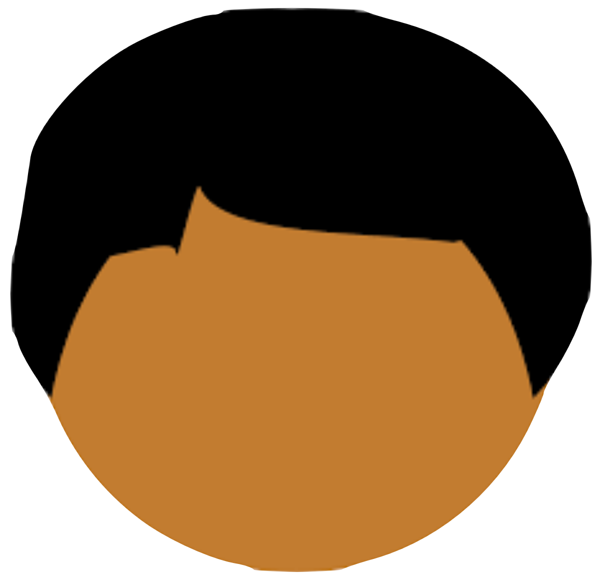
k

F(k, x0), F(k, x1),…

F(k, y0), F(k, y1),…

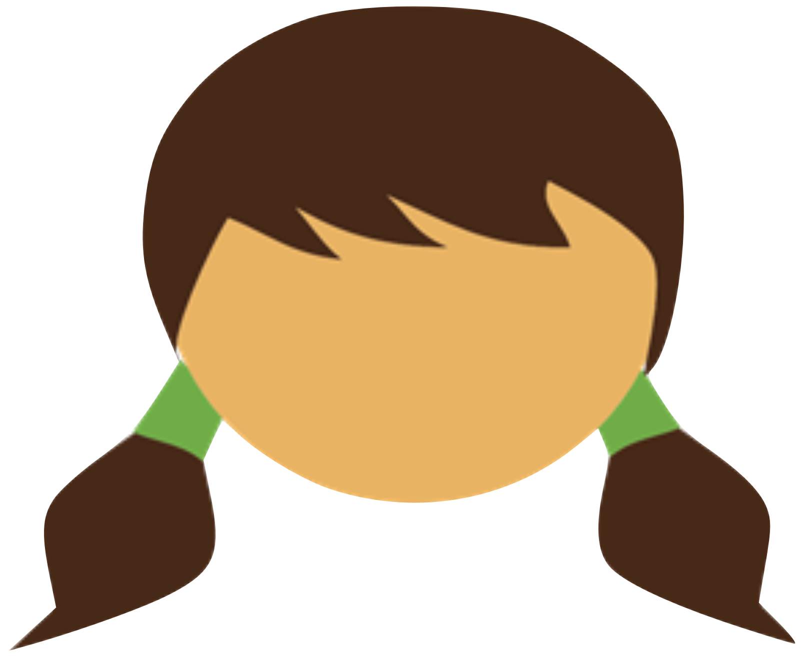
Batched Oblivious PRF

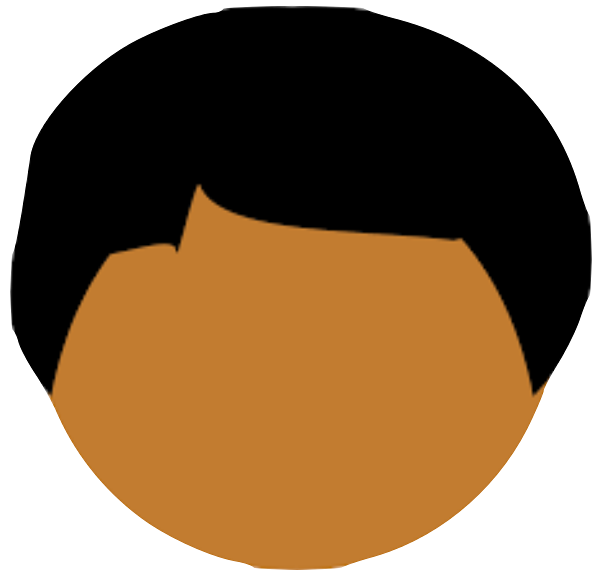




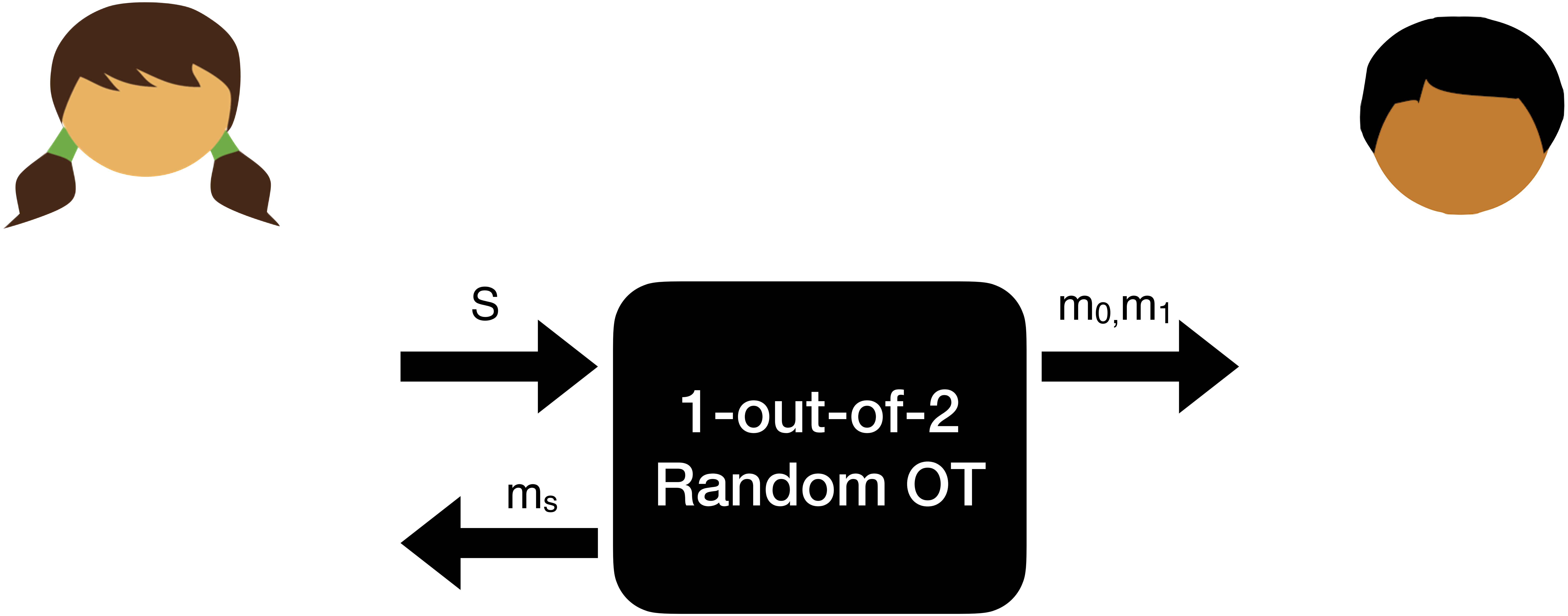
x0	0
x1	1
x2	2
x3	3
x4	4
x5	5

Batched Oblivious PRF





x0	F(k0, x0)	0	k0
x1	F(k1, x1)	1	k1
x2	F(k2, x2)	2	k2
x3	F(k3, x3)	3	k3
x4	F(k4, x4)	4	k4
x5	F(k5, x5)	5	k5

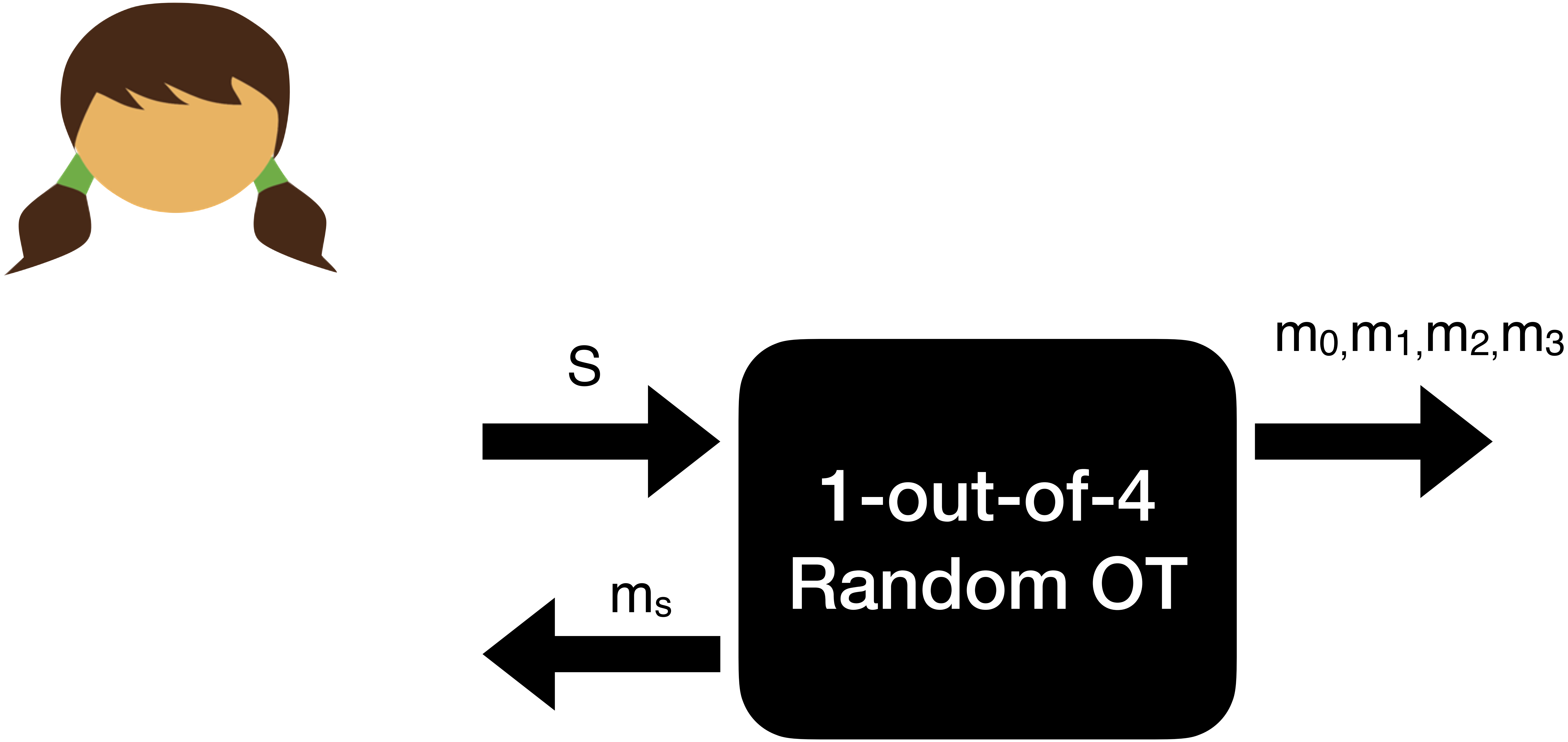


S

1-out-of-2
Random OT

m_0,m_1

m_s

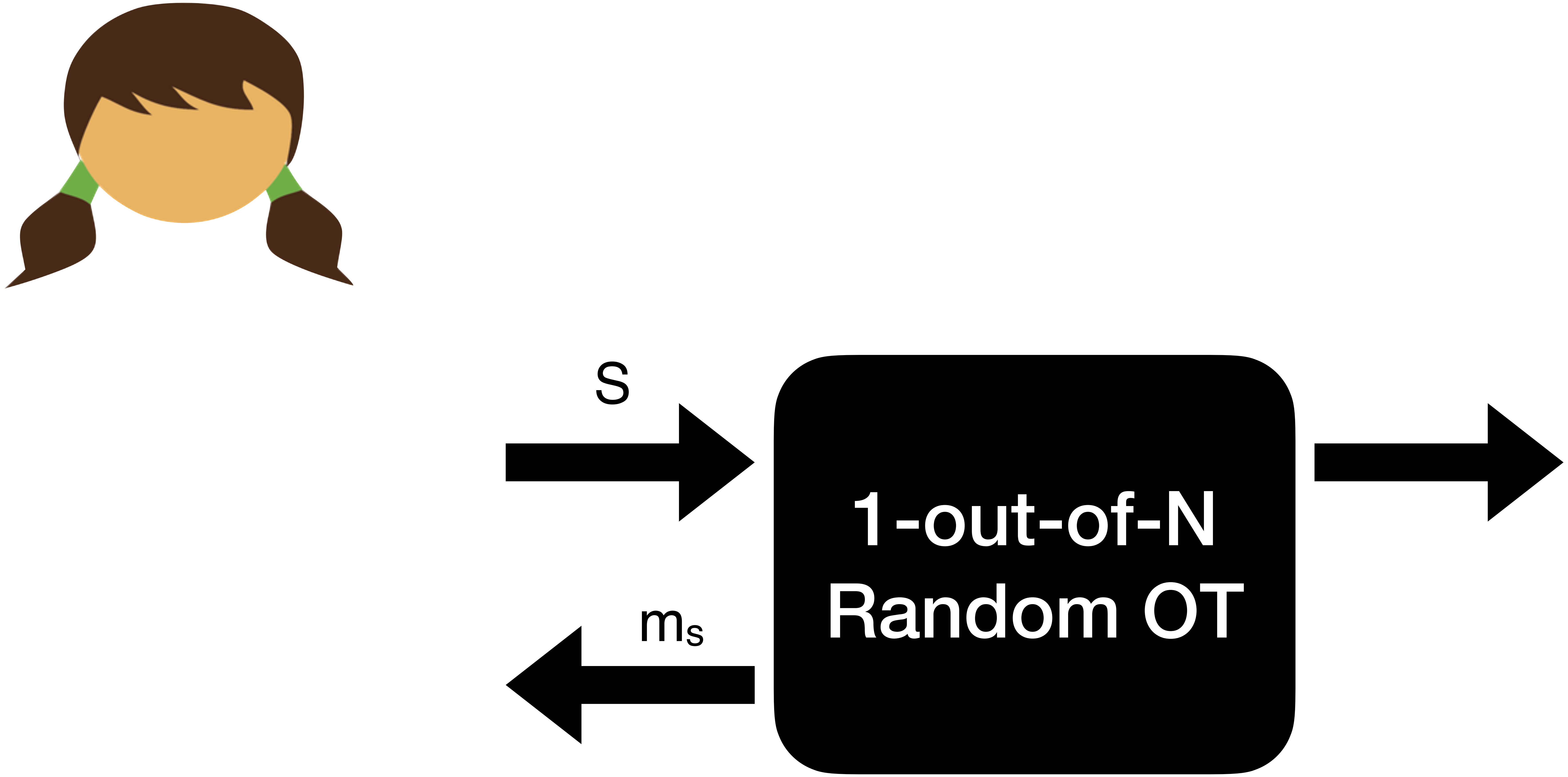


s

1-out-of-4 Random OT

m_s

m_0, m_1, m_2, m_3

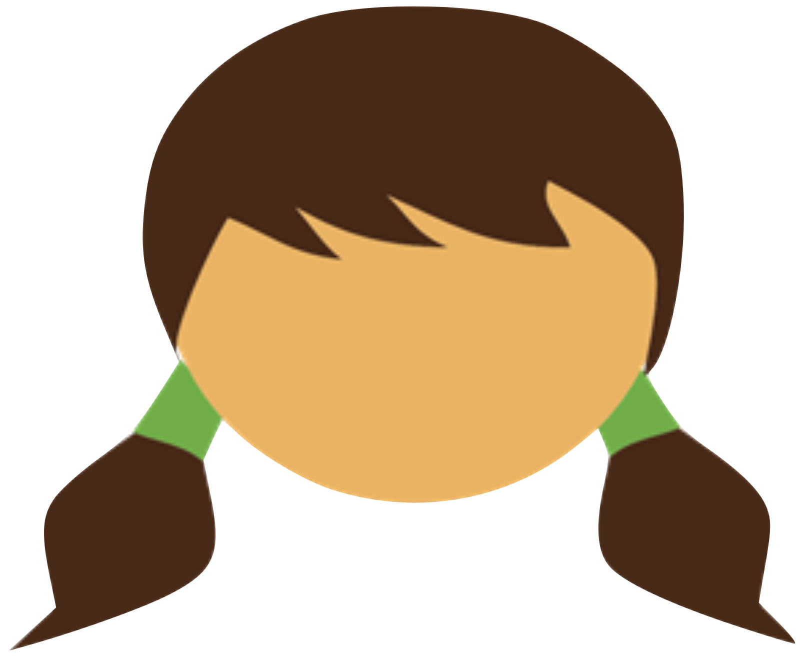


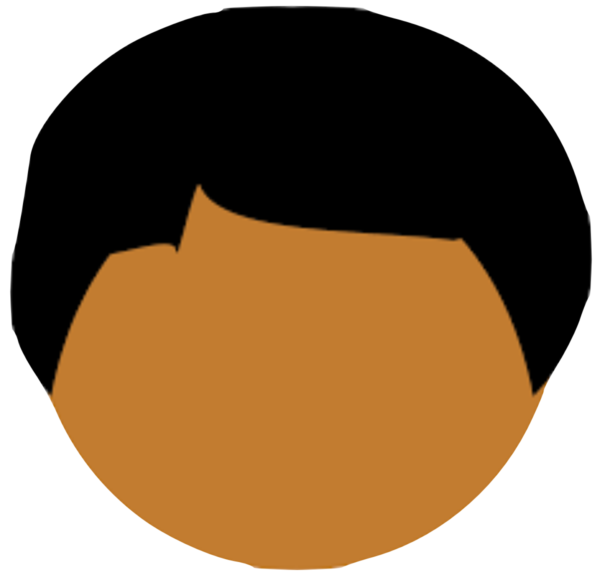
s

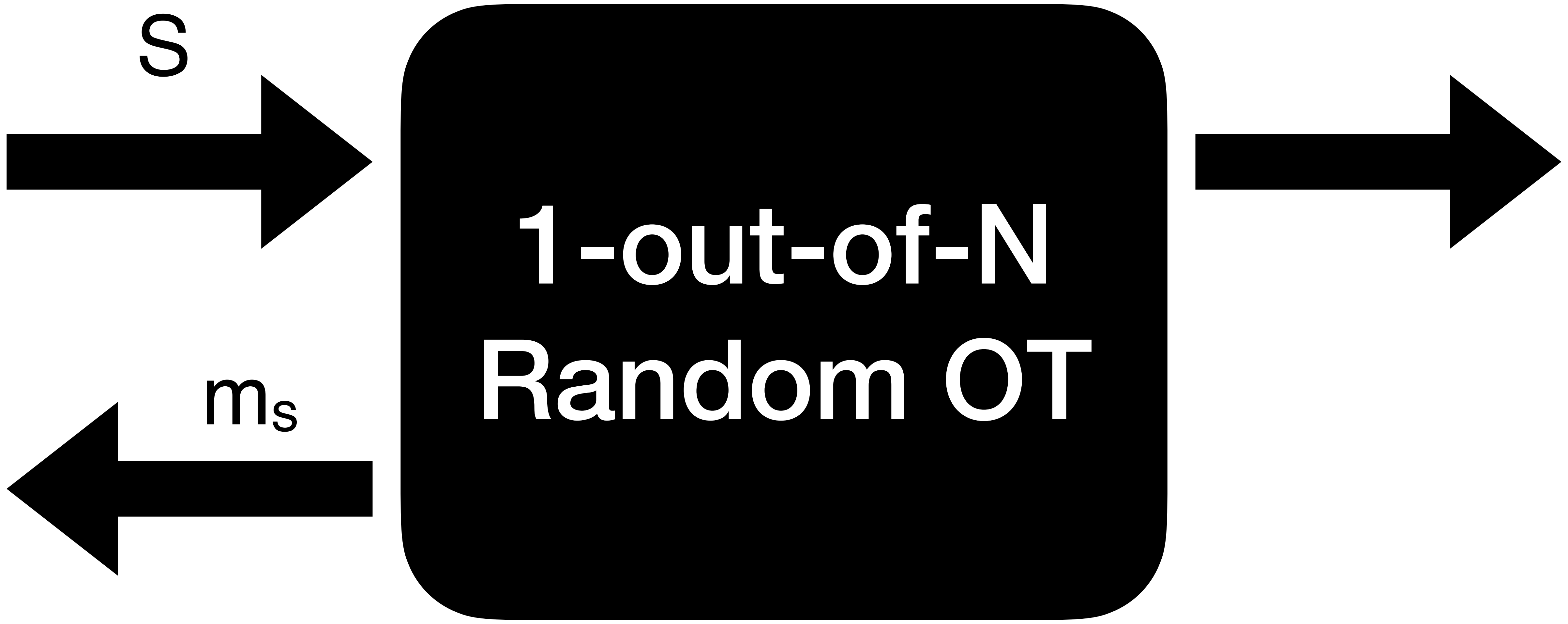
1-out-of-N Random OT

m_s

Use log n OTs

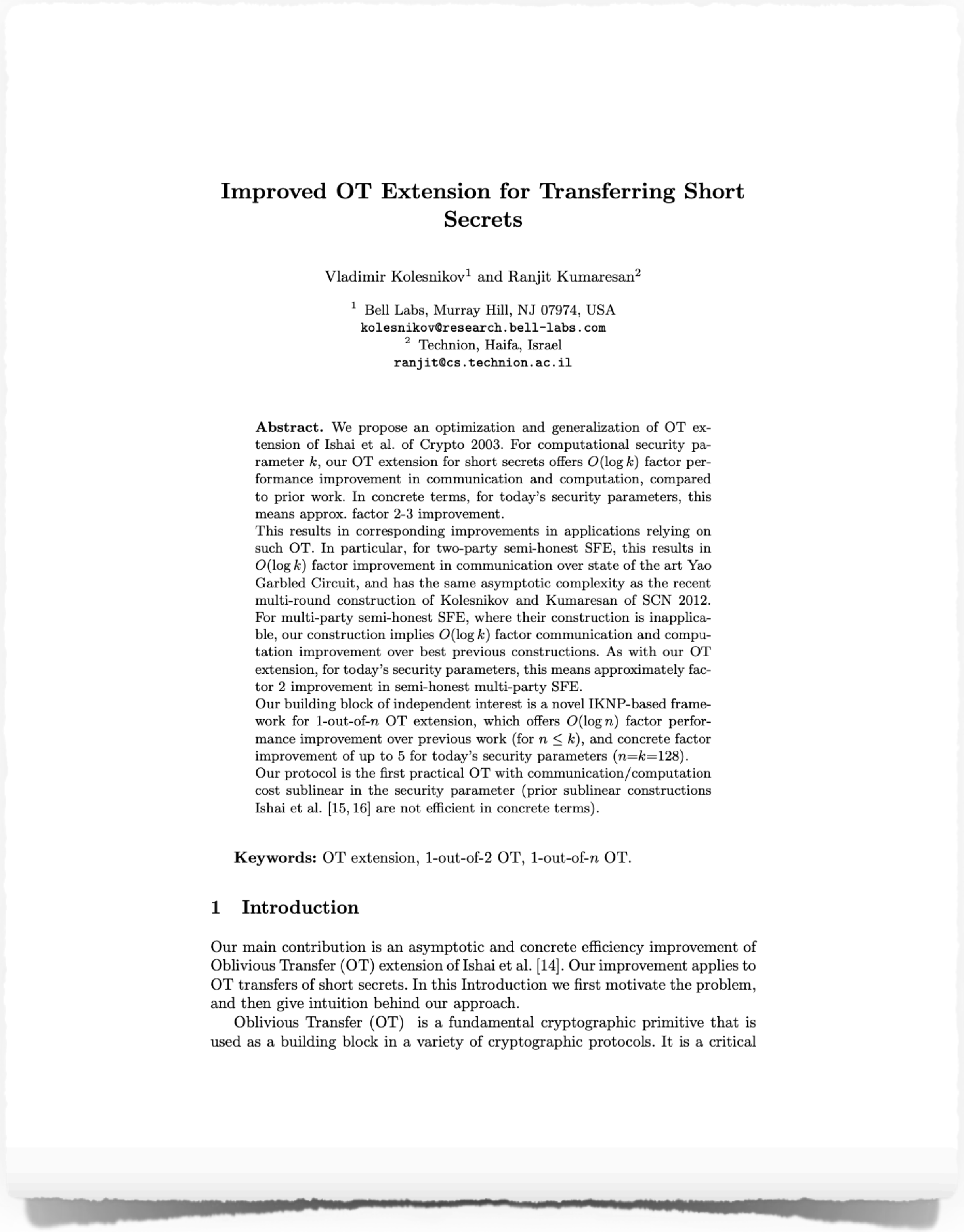






Use log n OTs

… or do something smarter



Improved OT Extension for Transferring Short Secrets

Vladimir Kolesnikov[1] and Ranjit Kumaresan[2]

[1] Bell Labs, Murray Hill, NJ 07974, USA
kolesnikov@research.bell-labs.com
[2] Technion, Haifa, Israel
ranjit@cs.technion.ac.il

Abstract. We propose an optimization and generalization of OT extension of Ishai et al. of Crypto 2003. For computational security parameter k, our OT extension for short secrets offers $O(\log k)$ factor performance improvement in communication and computation, compared to prior work. In concrete terms, for today's security parameters, this means approx. factor 2-3 improvement.
This results in corresponding improvements in applications relying on such OT. In particular, for two-party semi-honest SFE, this results in $O(\log k)$ factor improvement in communication over state of the art Yao Garbled Circuit, and has the same asymptotic complexity as the recent multi-round construction of Kolesnikov and Kumaresan of SCN 2012. For multi-party semi-honest SFE, where their construction is inapplicable, our construction implies $O(\log k)$ factor communication and computation improvement over best previous constructions. As with our OT extension, for today's security parameters, this means approximately factor 2 improvement in semi-honest multi-party SFE.
Our building block of independent interest is a novel IKNP-based framework for 1-out-of-n OT extension, which offers $O(\log n)$ factor performance improvement over previous work (for $n \leq k$), and concrete factor improvement of up to 5 for today's security parameters (n=k=128).
Our protocol is the first practical OT with communication/computation cost sublinear in the security parameter (prior sublinear constructions Ishai et al. [15, 16] are not efficient in concrete terms).

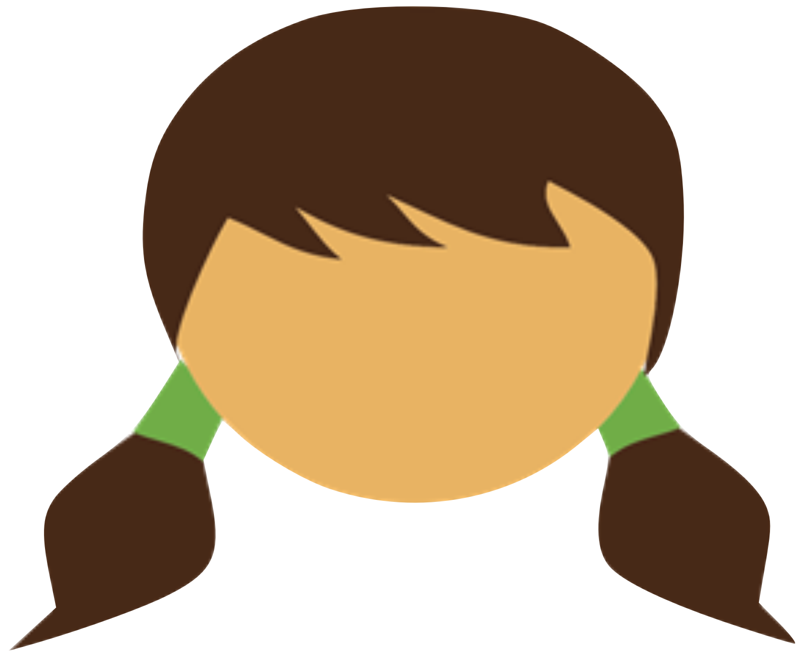
Keywords: OT extension, 1-out-of-2 OT, 1-out-of-n OT.

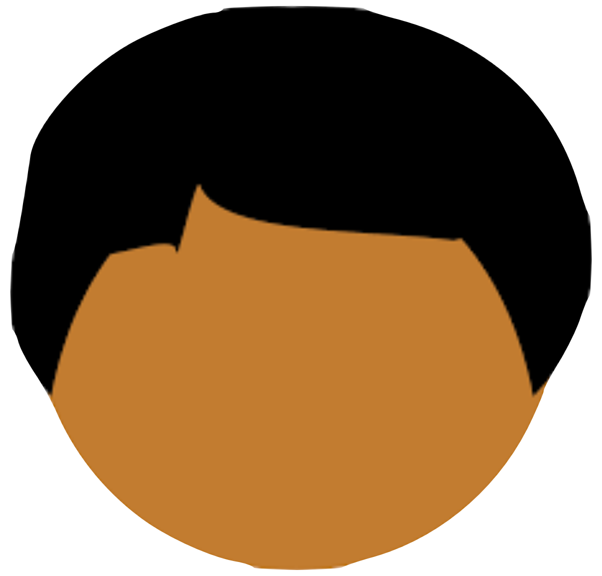
1 Introduction

Our main contribution is an asymptotic and concrete efficiency improvement of Oblivious Transfer (OT) extension of Ishai et al. [14]. Our improvement applies to OT transfers of short secrets. In this Introduction we first motivate the problem, and then give intuition behind our approach.

Oblivious Transfer (OT) is a fundamental cryptographic primitive that is used as a building block in a variety of cryptographic protocols. It is a critical

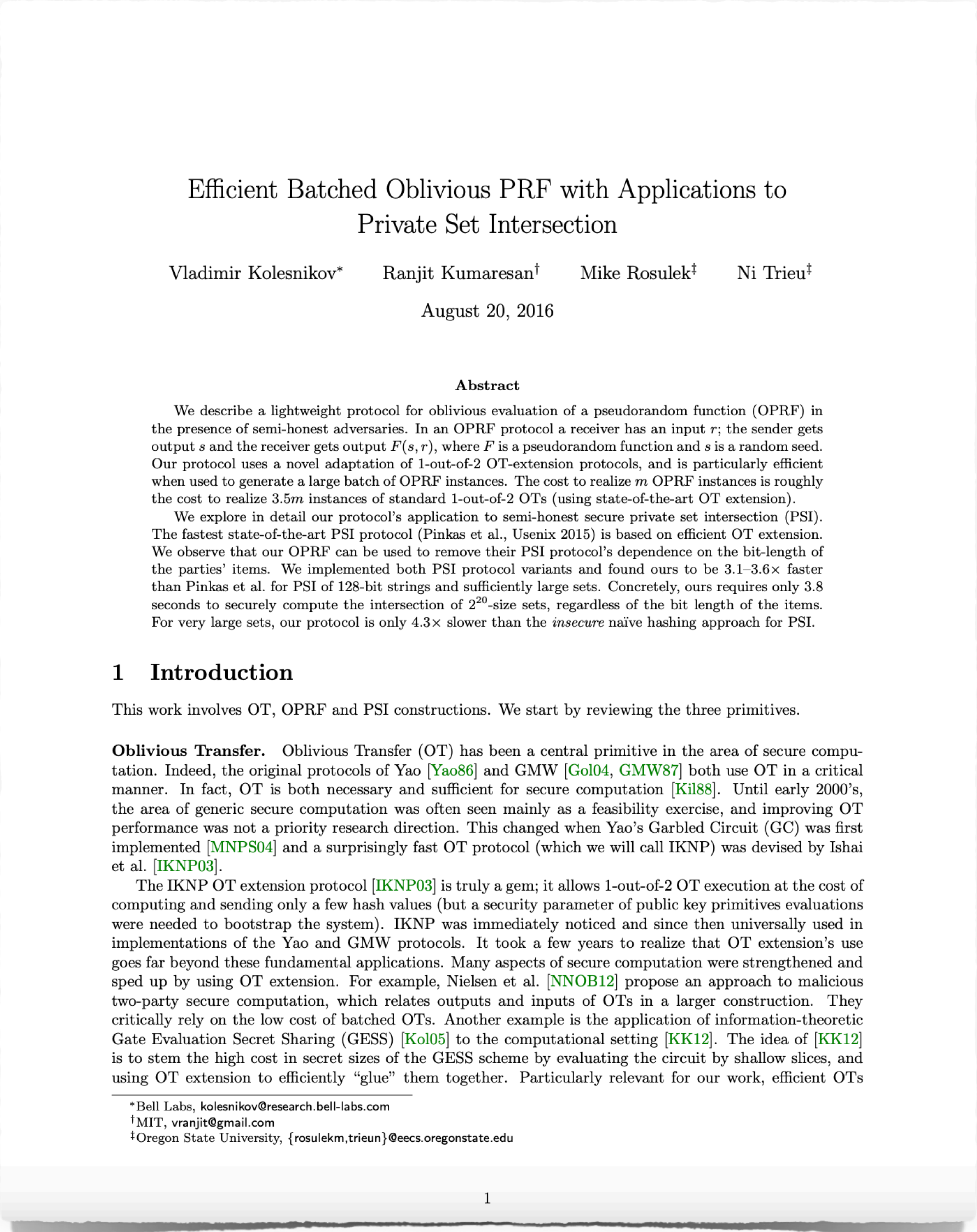
Batched Oblivious PRF





Essentially batched 1-out-of-N OT

x0	F(k0, x0)	0	k0
x1	F(k1, x1)	1	k1
x2	F(k2, x2)	2	k2
x3	F(k3, x3)	3	k3
x4	F(k4, x4)	4	k4
x5	F(k5, x5)	5	k5



Efficient Batched Oblivious PRF with Applications to Private Set Intersection

Vladimir Kolesnikov* Ranjit Kumaresan† Mike Rosulek‡ Ni Trieu‡

August 20, 2016

Abstract

We describe a lightweight protocol for oblivious evaluation of a pseudorandom function (OPRF) in the presence of semi-honest adversaries. In an OPRF protocol a receiver has an input r; the sender gets output s and the receiver gets output $F(s, r)$, where F is a pseudorandom function and s is a random seed. Our protocol uses a novel adaptation of 1-out-of-2 OT-extension protocols, and is particularly efficient when used to generate a large batch of OPRF instances. The cost to realize m OPRF instances is roughly the cost to realize $3.5m$ instances of standard 1-out-of-2 OTs (using state-of-the-art OT extension).

We explore in detail our protocol's application to semi-honest secure private set intersection (PSI). The fastest state-of-the-art PSI protocol (Pinkas et al., Usenix 2015) is based on efficient OT extension. We observe that our OPRF can be used to remove their PSI protocol's dependence on the bit-length of the parties' items. We implemented both PSI protocol variants and found ours to be 3.1–3.6× faster than Pinkas et al. for PSI of 128-bit strings and sufficiently large sets. Concretely, ours requires only 3.8 seconds to securely compute the intersection of 2^{20}-size sets, regardless of the bit length of the items. For very large sets, our protocol is only 4.3× slower than the *insecure* naïve hashing approach for PSI.

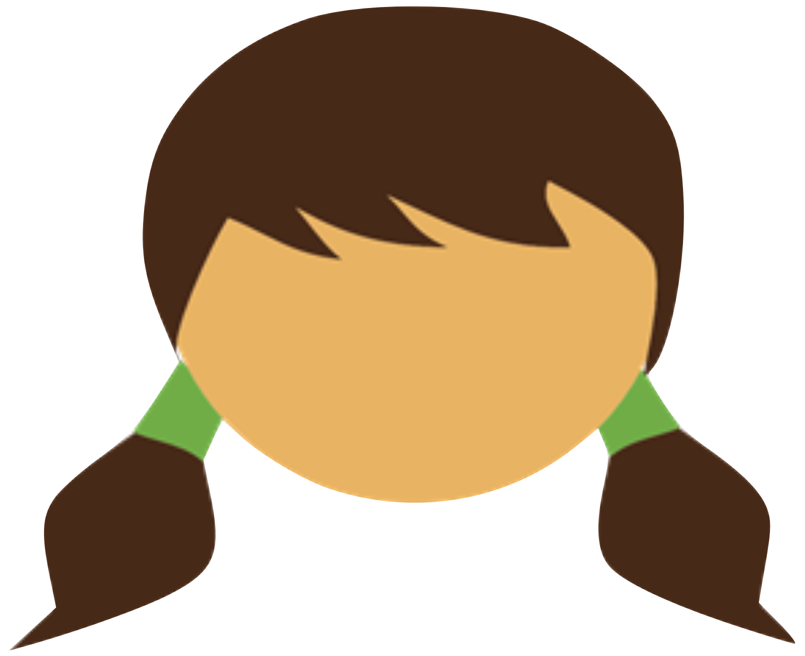
1 Introduction

This work involves OT, OPRF and PSI constructions. We start by reviewing the three primitives.

Oblivious Transfer. Oblivious Transfer (OT) has been a central primitive in the area of secure computation. Indeed, the original protocols of Yao [Yao86] and GMW [Gol04, GMW87] both use OT in a critical manner. In fact, OT is both necessary and sufficient for secure computation [Kil88]. Until early 2000's, the area of generic secure computation was often seen mainly as a feasibility exercise, and improving OT performance was not a priority research direction. This changed when Yao's Garbled Circuit (GC) was first implemented [MNPS04] and a surprisingly fast OT protocol (which we will call IKNP) was devised by Ishai et al. [IKNP03].

The IKNP OT extension protocol [IKNP03] is truly a gem; it allows 1-out-of-2 OT execution at the cost of computing and sending only a few hash values (but a security parameter of public key primitives evaluations were needed to bootstrap the system). IKNP was immediately noticed and since then universally used in implementations of the Yao and GMW protocols. It took a few years to realize that OT extension's use goes far beyond these fundamental applications. Many aspects of secure computation were strengthened and sped up by using OT extension. For example, Nielsen et al. [NNOB12] propose an approach to malicious two-party secure computation, which relates outputs and inputs of OTs in a larger construction. They critically rely on the low cost of batched OTs. Another example is the application of information-theoretic Gate Evaluation Secret Sharing (GESS) [Kol05] to the computational setting [KK12]. The idea of [KK12] is to stem the high cost in secret sizes of the GESS scheme by evaluating the circuit by shallow slices, and using OT extension to efficiently "glue" them together. Particularly relevant for our work, efficient OTs

*Bell Labs, kolesnikov@research.bell-labs.com
†MIT, vranjit@gmail.com
‡Oregon State University, {rosulekm,trieun}@eecs.oregonstate.edu



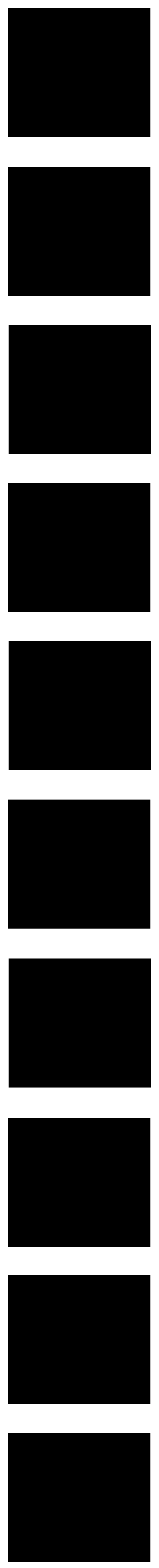
A

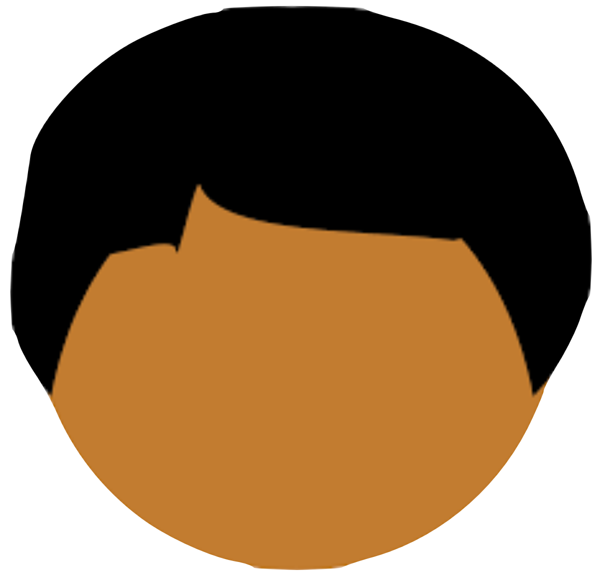
C

D

F

bins





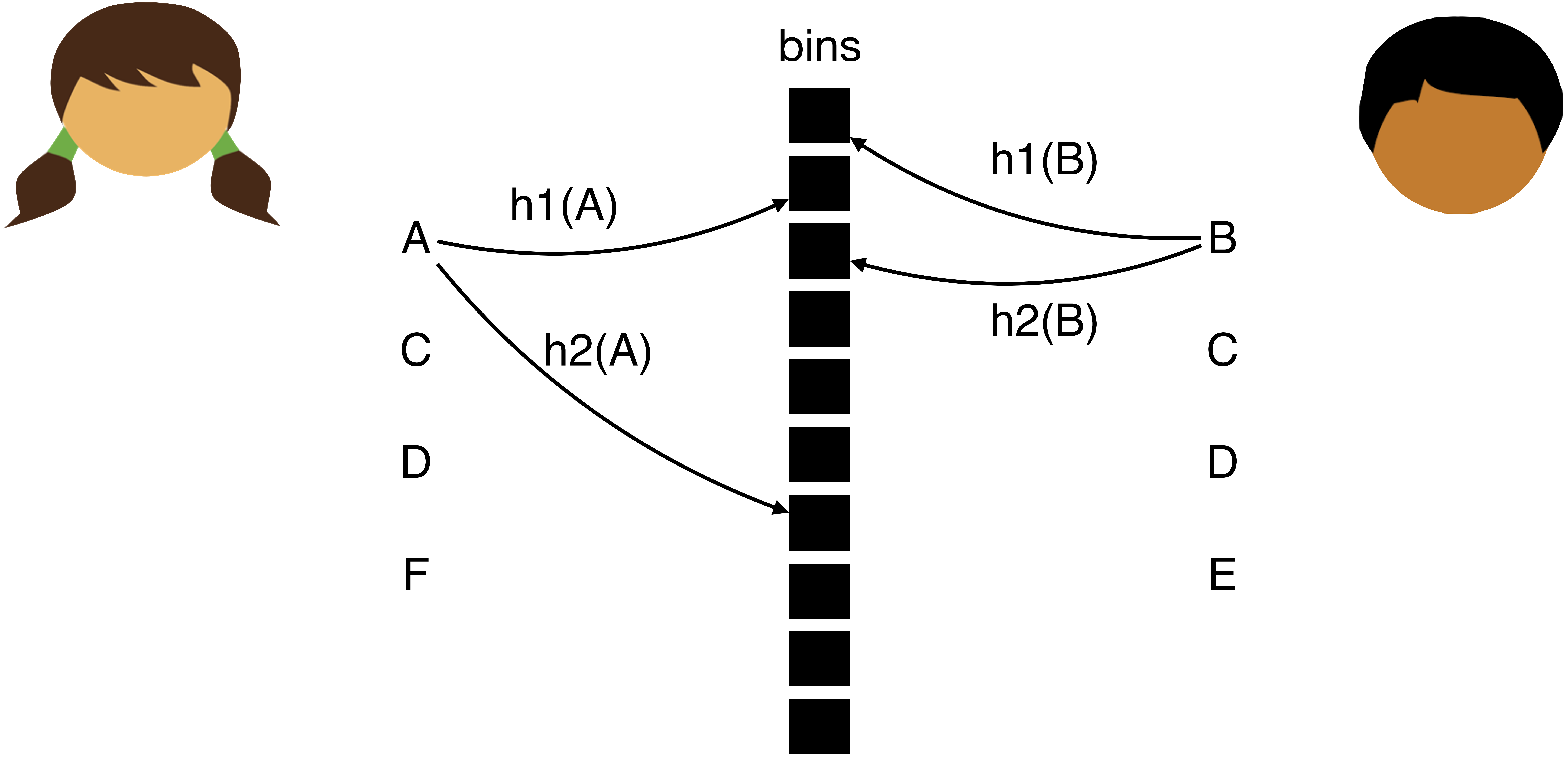
B

C

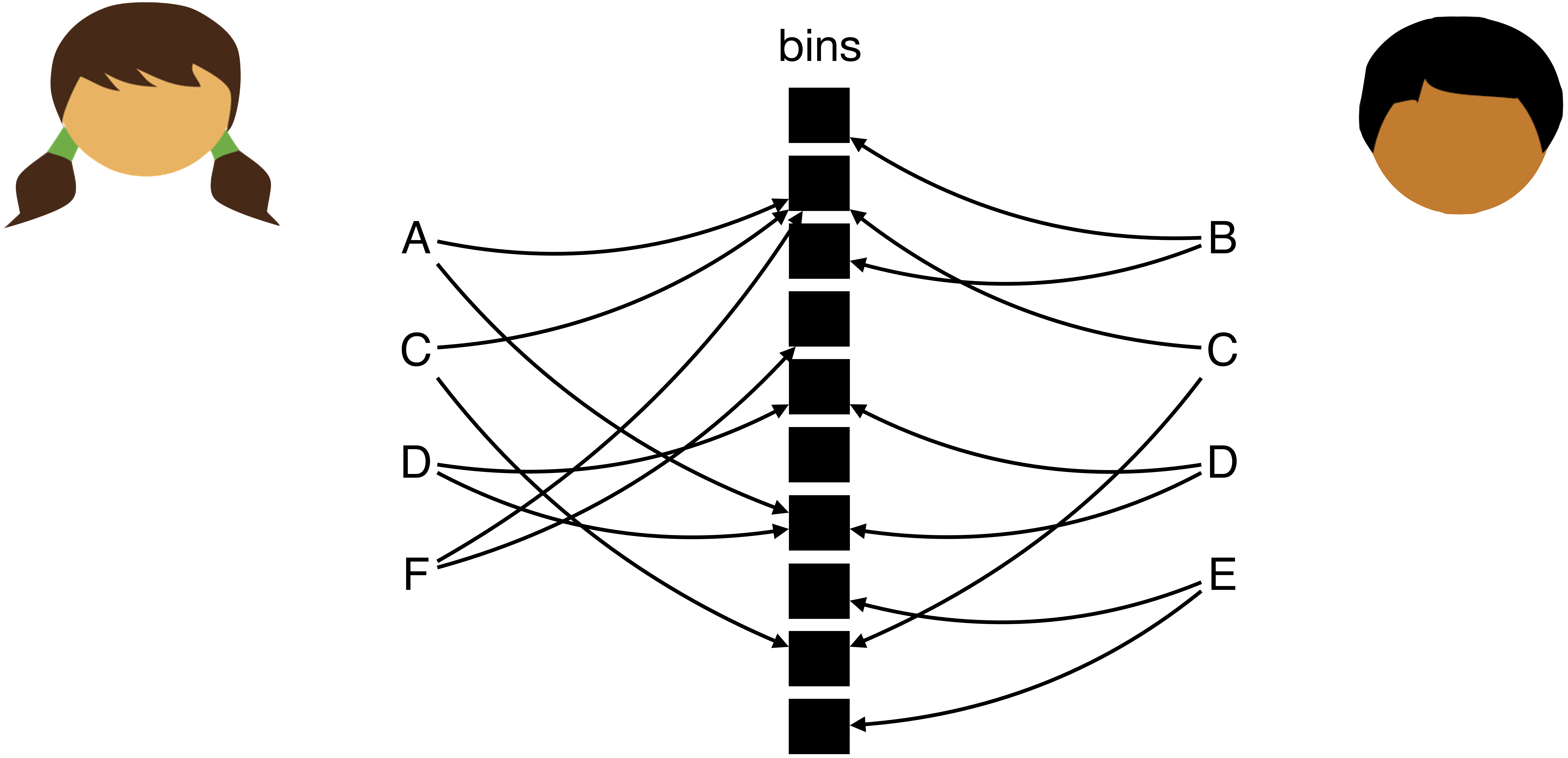
D

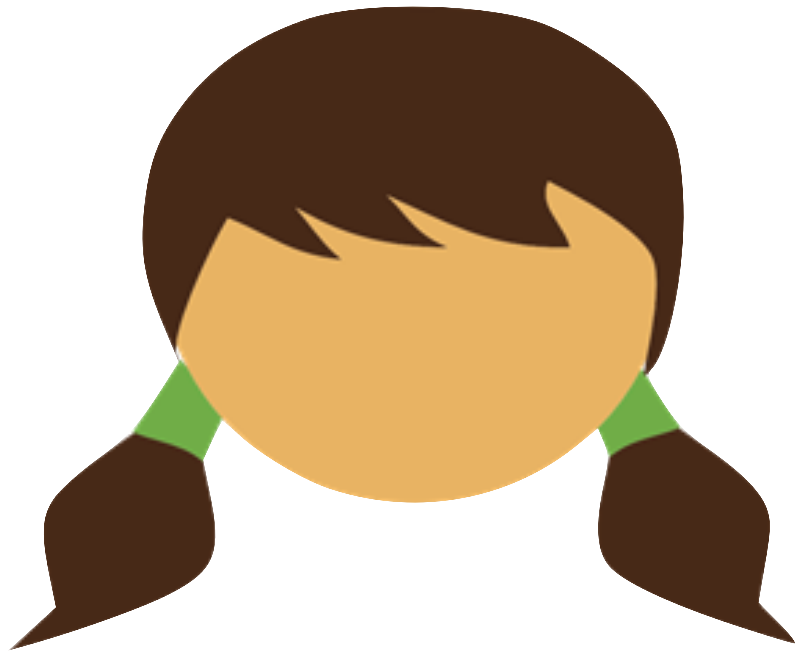
E

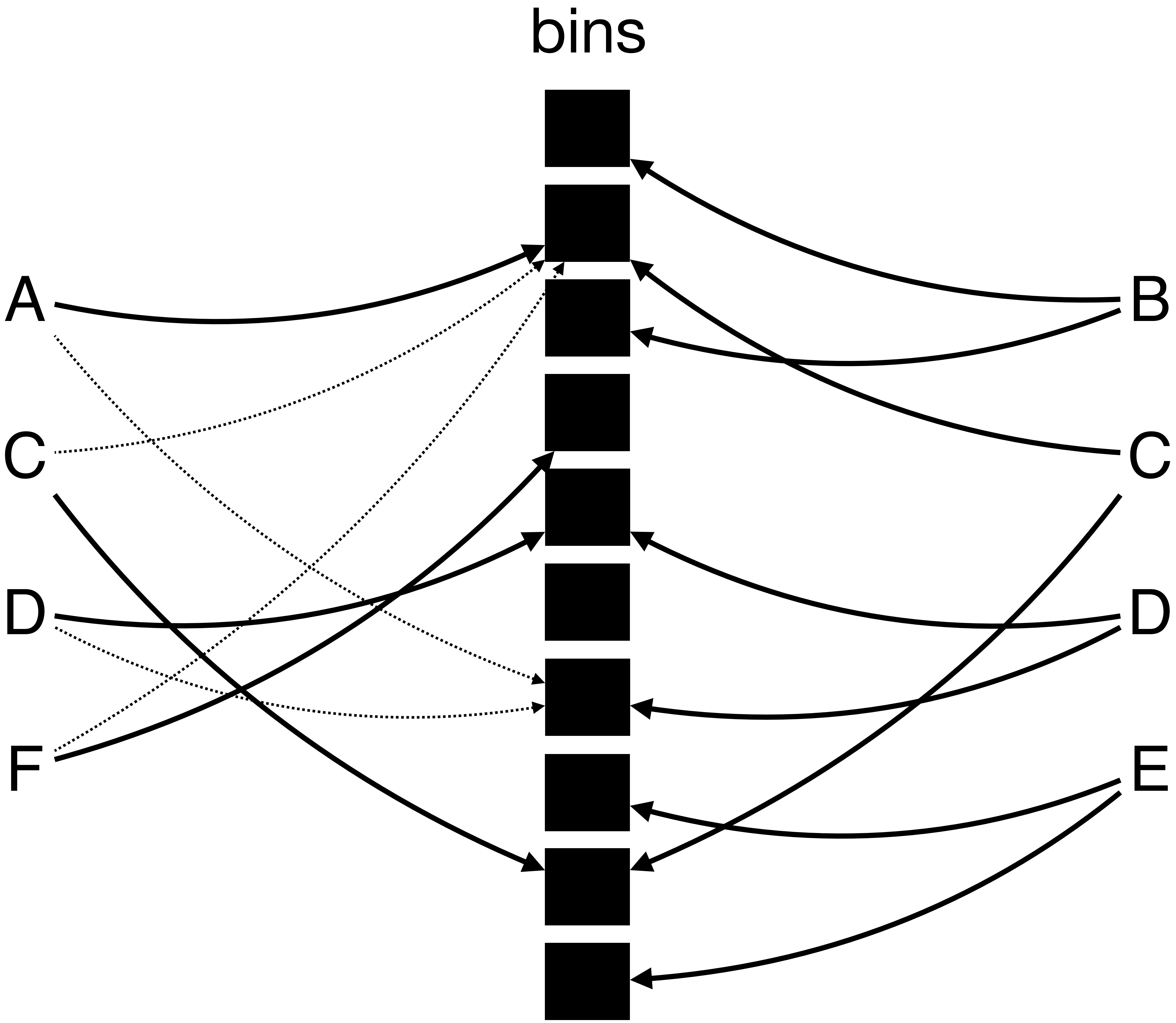
h1, h2 are two public hash functions

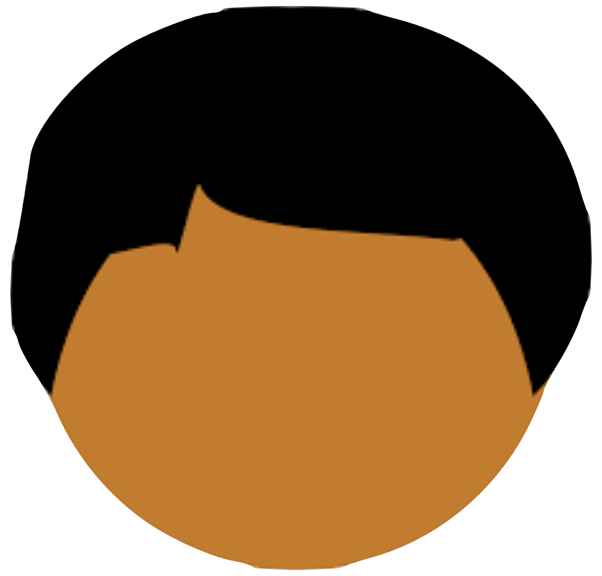


h1, h2 are two public hash functions

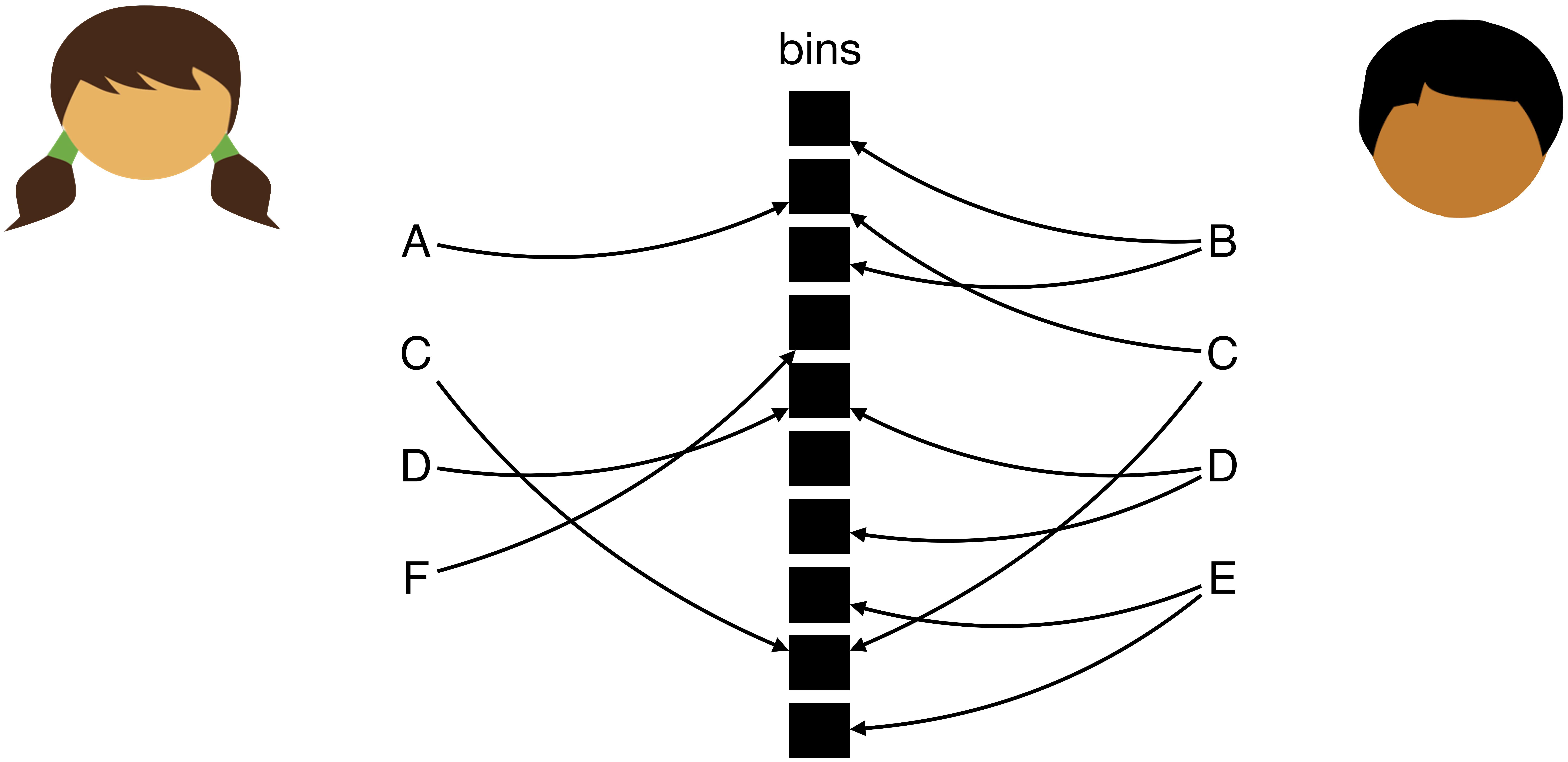

bins
A
C
D
F
B
C
D
E



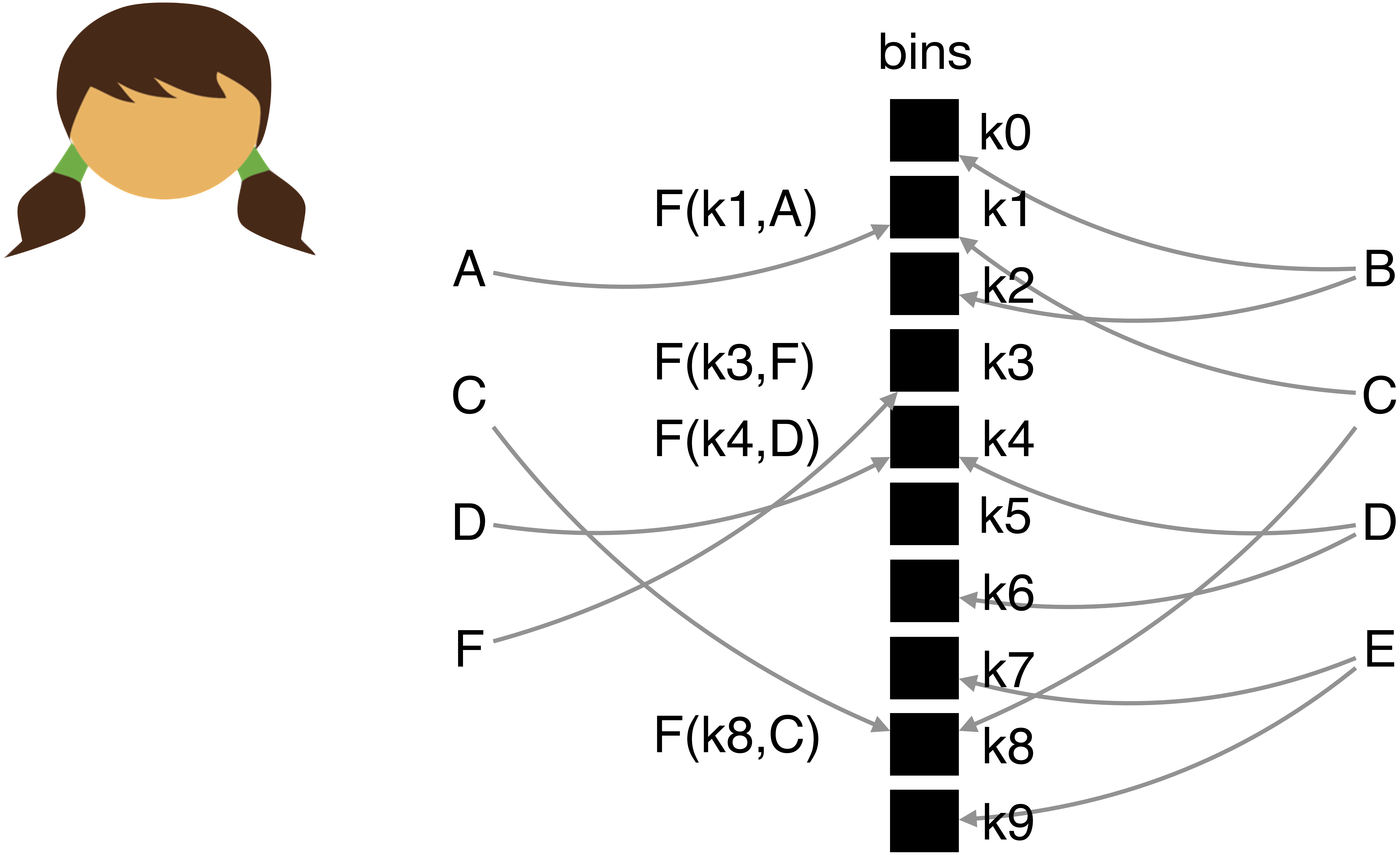


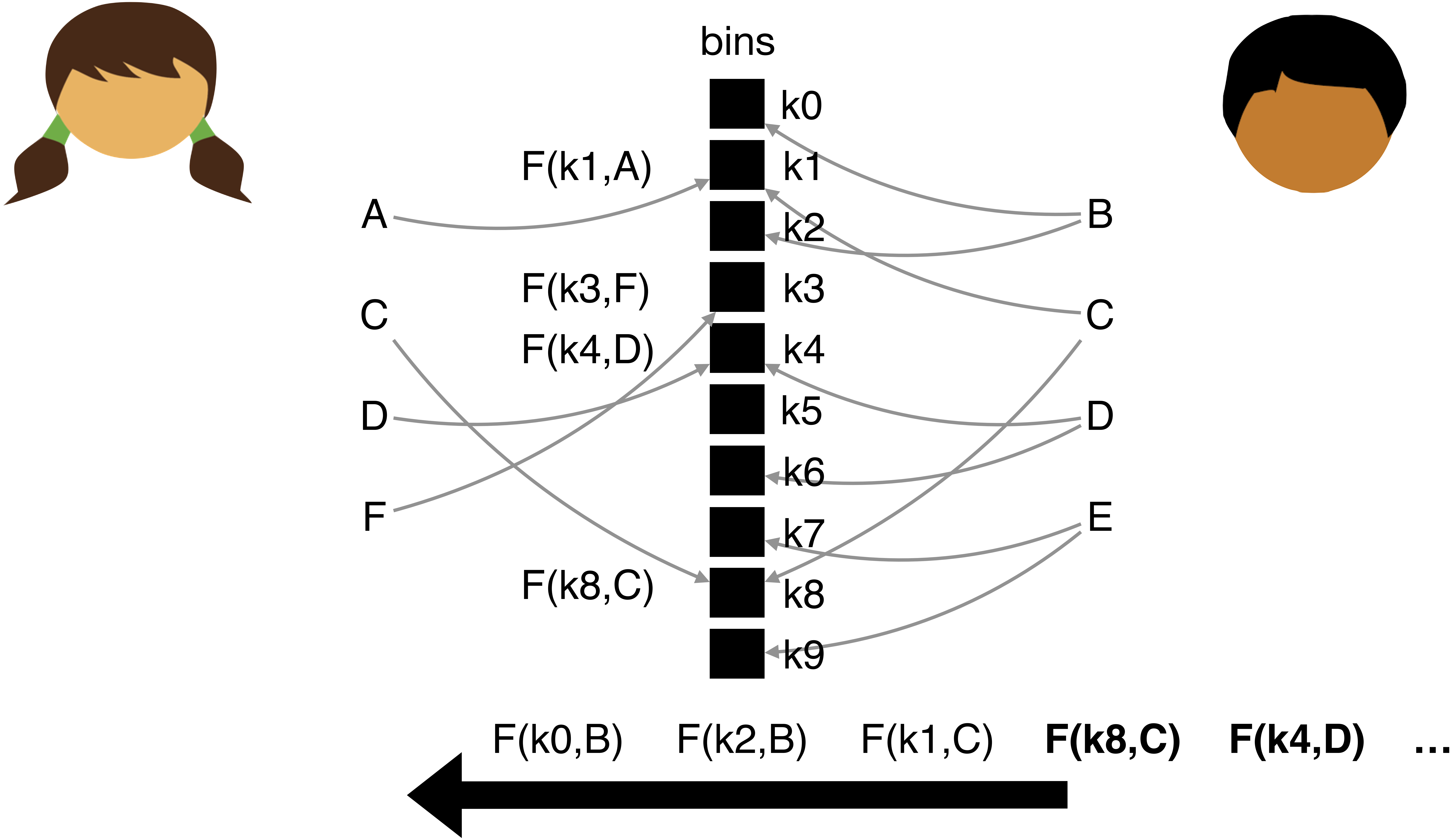


cuckoo hashing: if there are enough hash functions and enough bins, Alice can place at most one item in each bin (with very small overflow)



cuckoo hashing: if there are enough hash functions and enough bins, Alice can place at most one item in each bin (with very small overflow)


bins
k0
k1
k2
k3
k4
k5
k6
k7
k8
k9
F(k1,A)
F(k3,F)
F(k4,D)
F(k8,C)
A
C
D
F
B
C
D
E



bins

k0

k1

k2

k3

k4

k5

k6

k7

k8

k9

F(k1,A)

F(k3,F)

F(k4,D)

F(k8,C)

A

C

D

F

B

C

D

E

F(k0,B) F(k2,B) F(k1,C) **F(k8,C)** **F(k4,D)** …