Function Secret Sharing And Private Information Retrieval

CS 598 DH

Today’s objectives

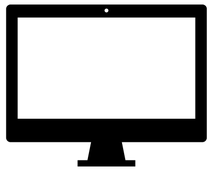
Define 2-server private information retrieval

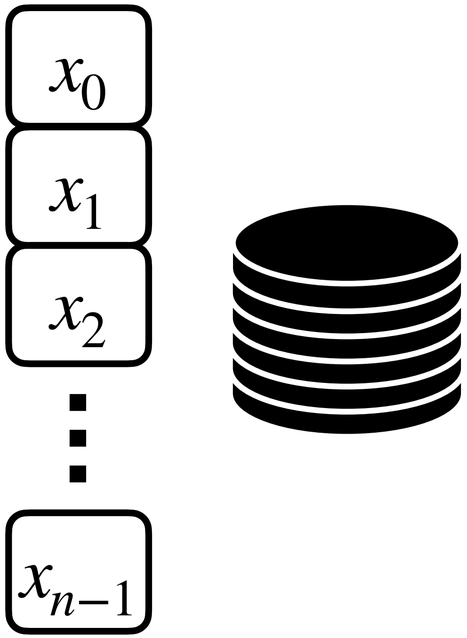
Introduce notion of function secret sharing

Construct distributed point function

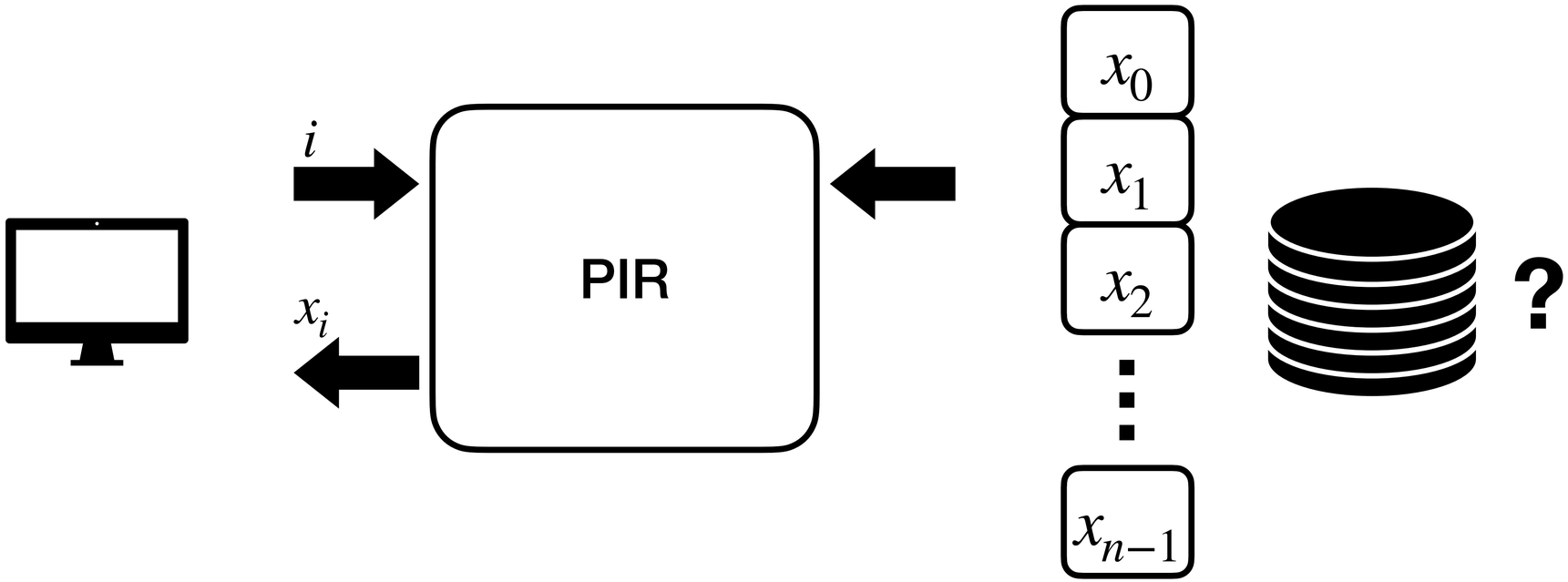
Show improved 2-server PIR protocol

Private Information Retrieval



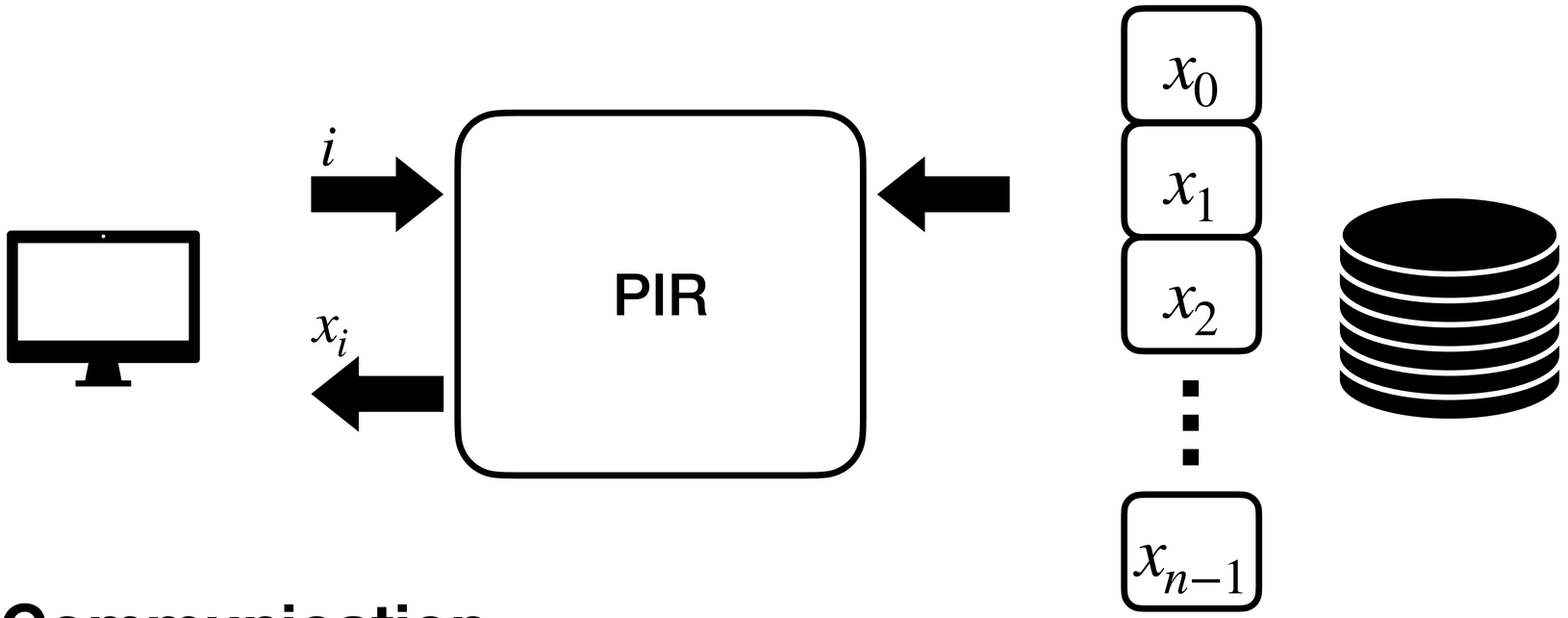


Private Information Retrieval



Client wishes to privately query one element from a large database

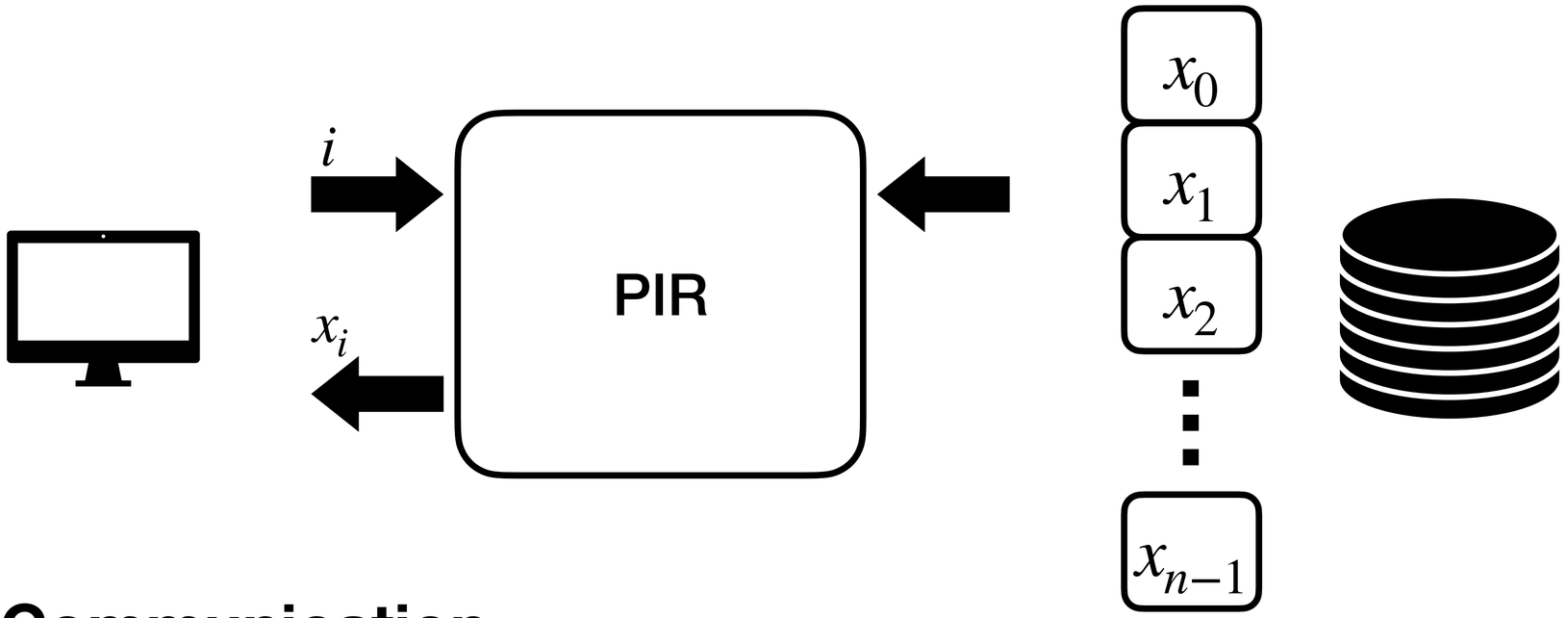
Private Information Retrieval



Goal: Low Communication
Semi-honest server

Question: What about ORAM?

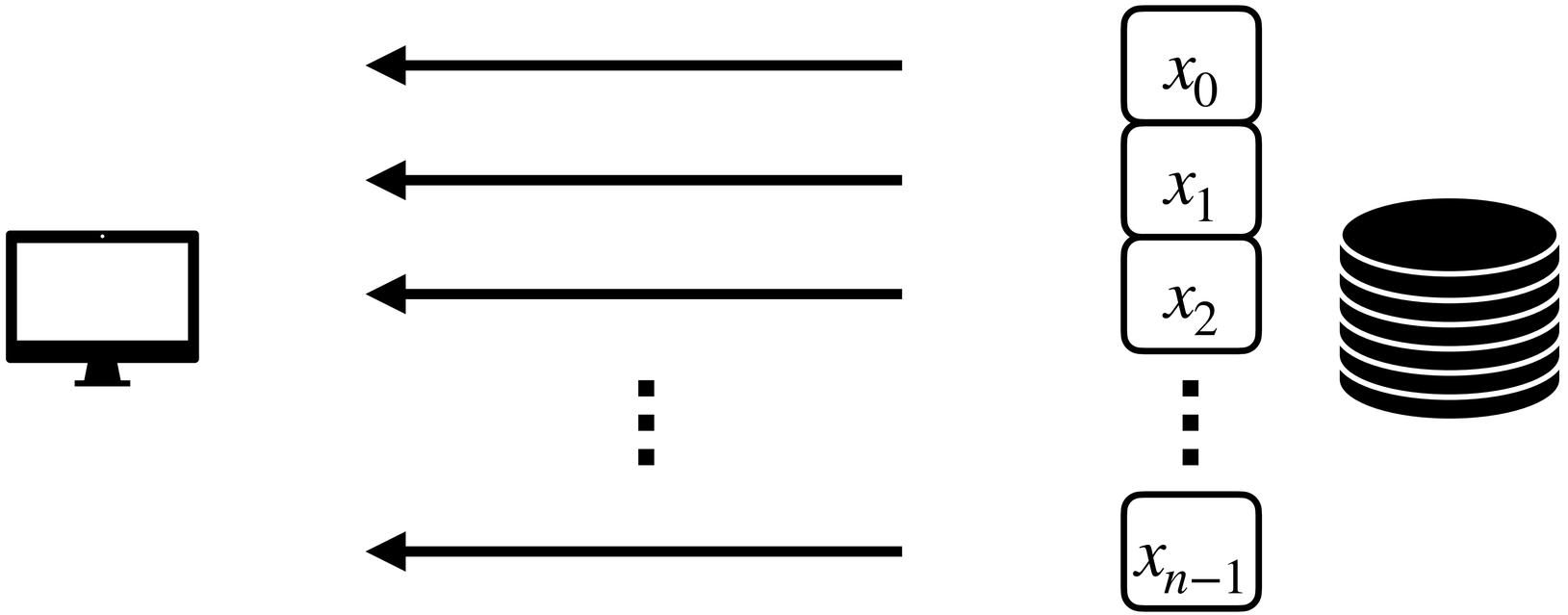
Private Information Retrieval



Goal: Low Communication
Semi-honest server

Question: What about ORAM?

ORAM achieves efficiency only in an amortized sense



Naive approach,
unacceptable overhead

Private Information Retrieval

BENNY CHOR

Technion, Haifa, Israel

ODED GOLDREICH

Weizmann Institute of Science, Rehovot, Israel

EYAL KUSHILEVITZ

Technion, Haifa, Israel

AND

MADHU SUDAN

Massachusetts Institute of Technology, Cambridge, Massachusetts

Abstract. Publicly accessible databases are an indispensable resource for retrieving up-to-date information. But they also pose a significant risk to the privacy of the user, since a curious database operator can follow the user's queries and infer what the user is after. Indeed, in cases where the users' intentions are to be kept secret, users are often cautious about accessing the database. It can be shown that when accessing a single database, to completely guarantee the privacy of the user, the whole database should be down-loaded; namely n bits should be communicated (where n is the number of bits in the database).

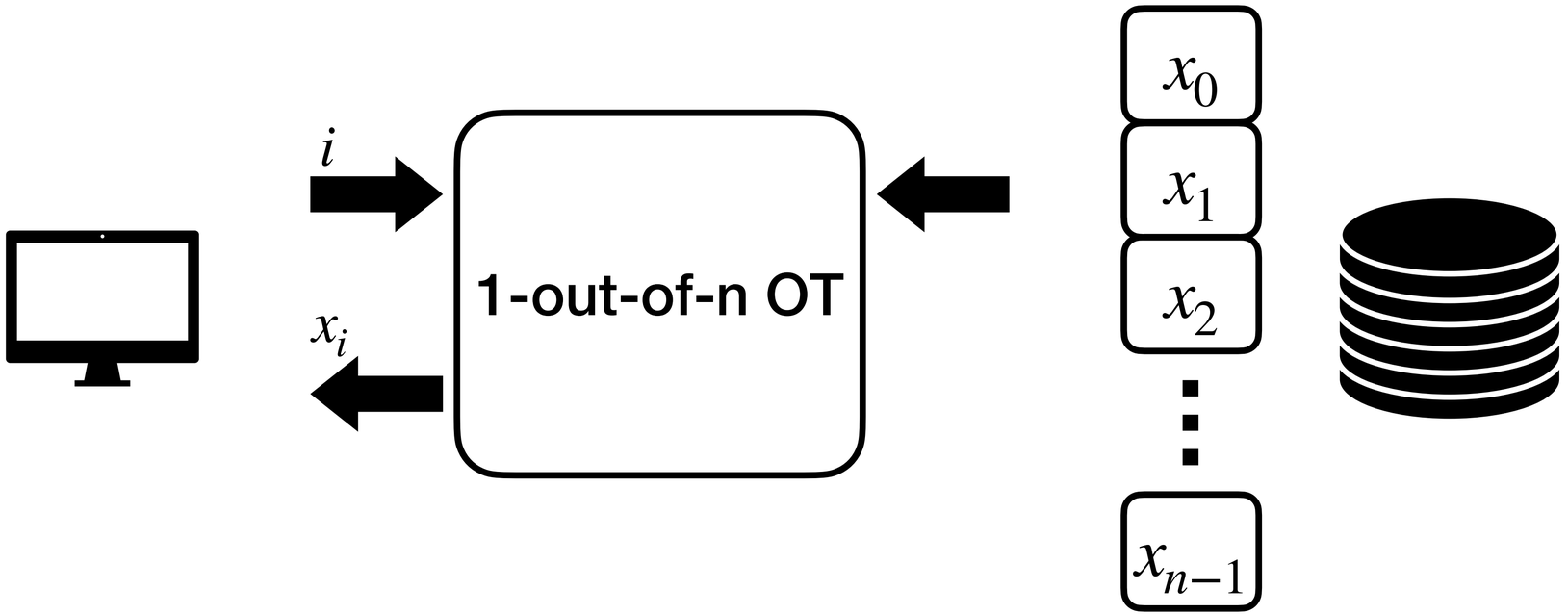
In this work, we investigate whether by replicating the database, more efficient solutions to the private retrieval problem can be obtained. We describe schemes that enable a user to access k replicated copies of a database ($k \geq 2$) and *privately* retrieve information stored in the database. This means that each individual server (holding a replicated copy of the database) gets no information on the identity of the item retrieved by the user. Our schemes use the replication to gain substantial saving. In particular, we present a two-server scheme with communication complexity $O(n^{1/3})$.

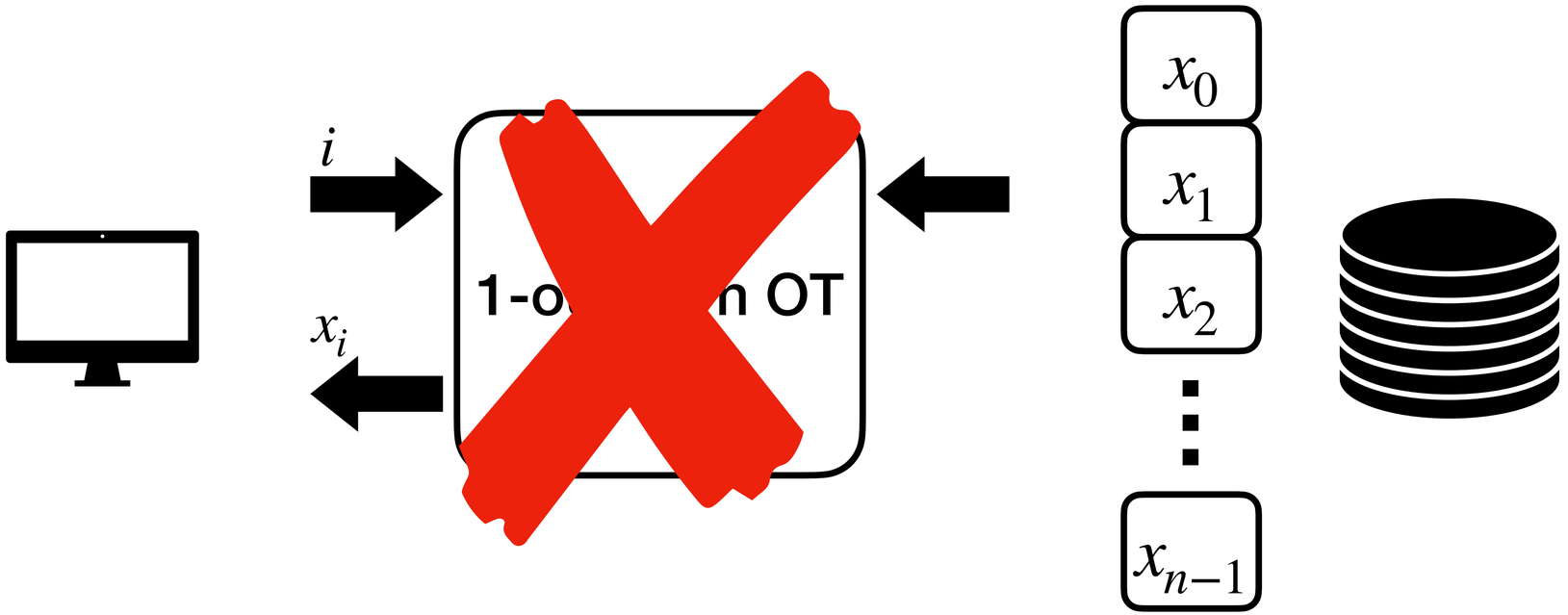
A preliminary version of this paper appeared in *Proceedings of the 36th Annual IEEE Conference on Foundations of Computer Science* (Oct.). IEEE, New York, 1995, pp. 41–50.

Some of this work was done while M. Sudan was at IBM.

The work of O. Goldreich was supported by grant No. 92-00226 from the Israel-US Binational Science Foundation (BSF), Jerusalem, Israel.

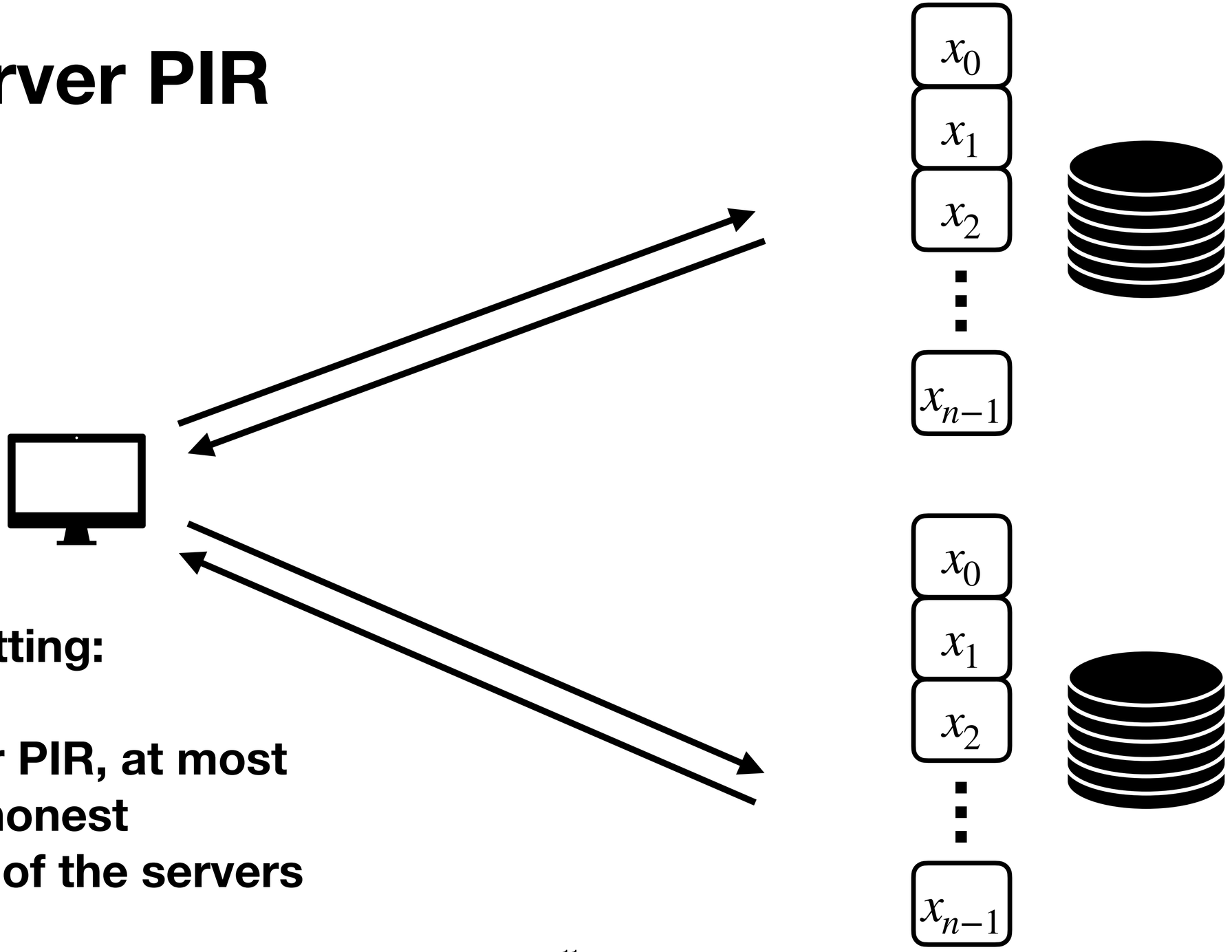
Authors' addresses: B. Chor, Computer Science Dept., Technion, Haifa, Israel, e-mail: {benny, eyalk}@cs.technion.ac.il, internet: http://www.cs.technion.ac.il/~eyalk; O. Goldreich, Computer Science and Applied Mathematics Dept., Weizmann Institute of Science, Rehovot, Israel, e-mail: oded@wisdom.weizmann.ac.il; M. Sudan, Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139, e-mail: madhu@theory.lcs.mit.edu.


i
x_i
1-out-of-n OT
x_0
x_1
x_2
x_{n-1}



OT requires sending something proportional to all n messages

Two Server PIR

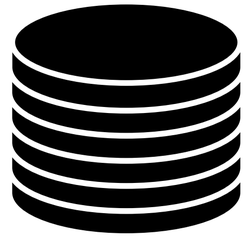


Simpler setting:

Two-server PIR, at most one semi-honest corruption of the servers

XOR Secret Sharing

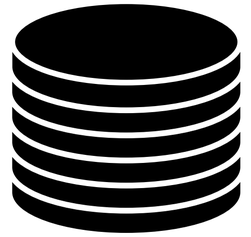
a_0



$[a]$

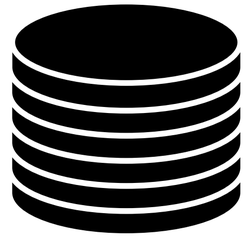
We can imagine trying to secret share the database, but it is not clear how it will help

a_1



Function Secret Sharing

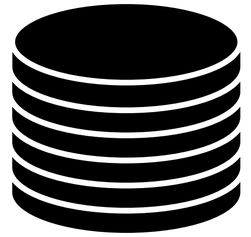
f_0



$[f]$

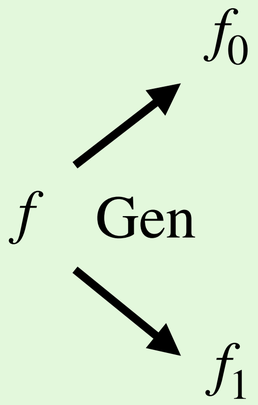
Instead, today we will show how
how to “secret share” the
client’s **query**

f_1

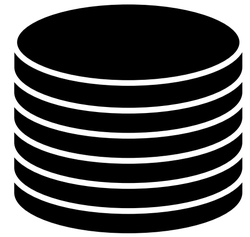


Function Secret Sharing

$f \in \mathscr{F}$

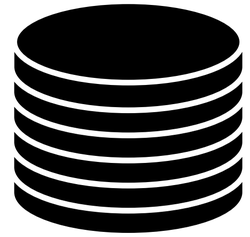


f_0

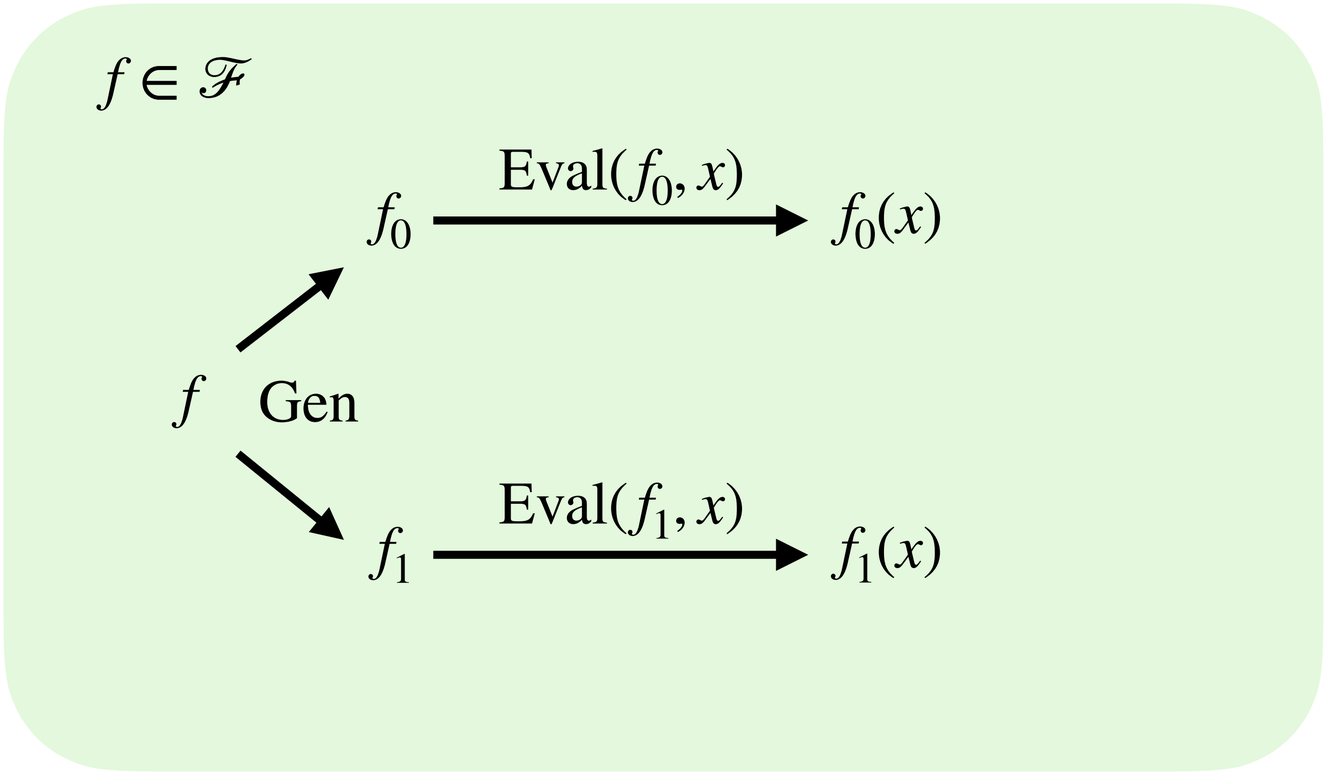


$[f]$

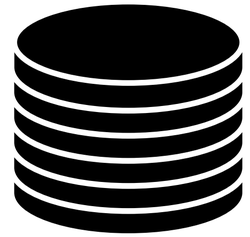
f_1



Function Secret Sharing

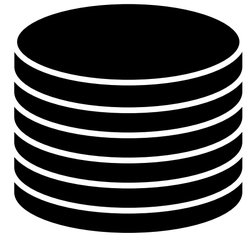


f_0

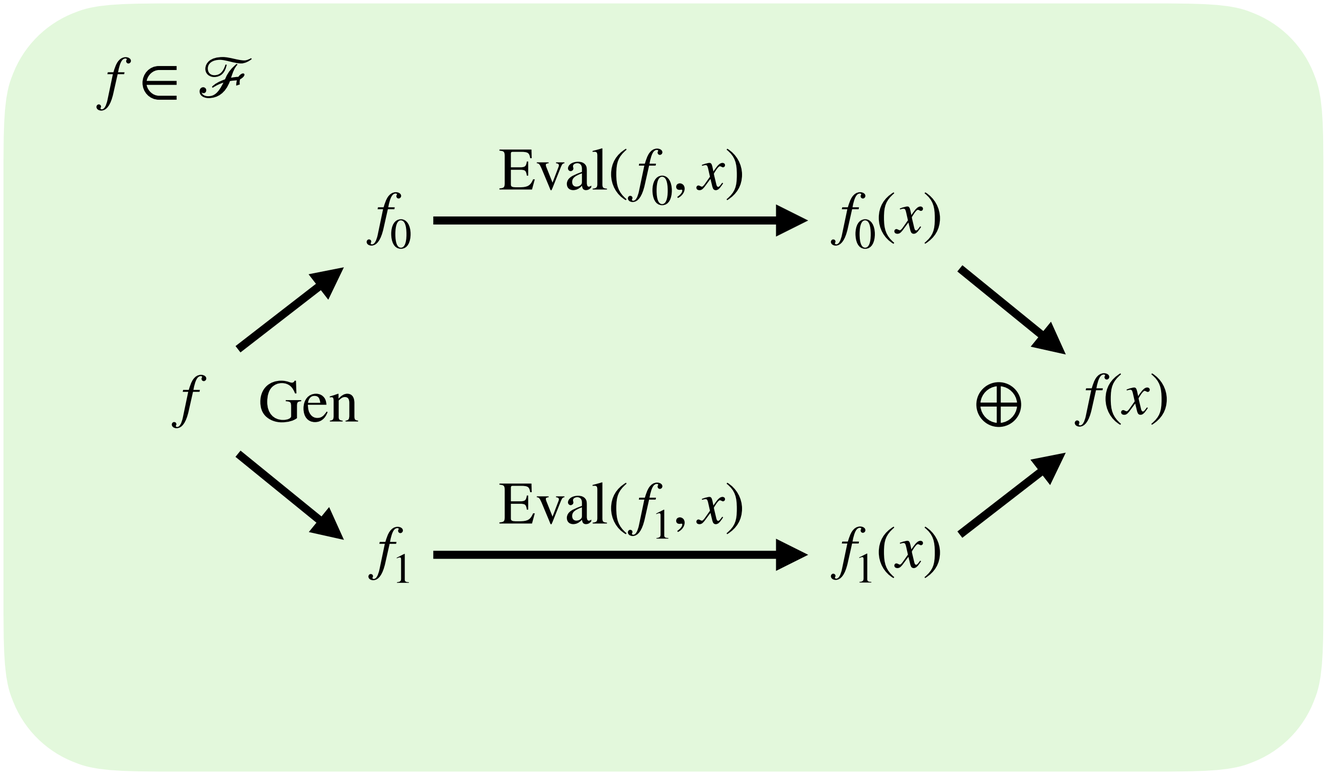


$[f]$

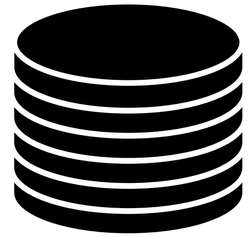
f_1



Function Secret Sharing

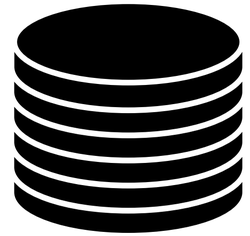


f_0

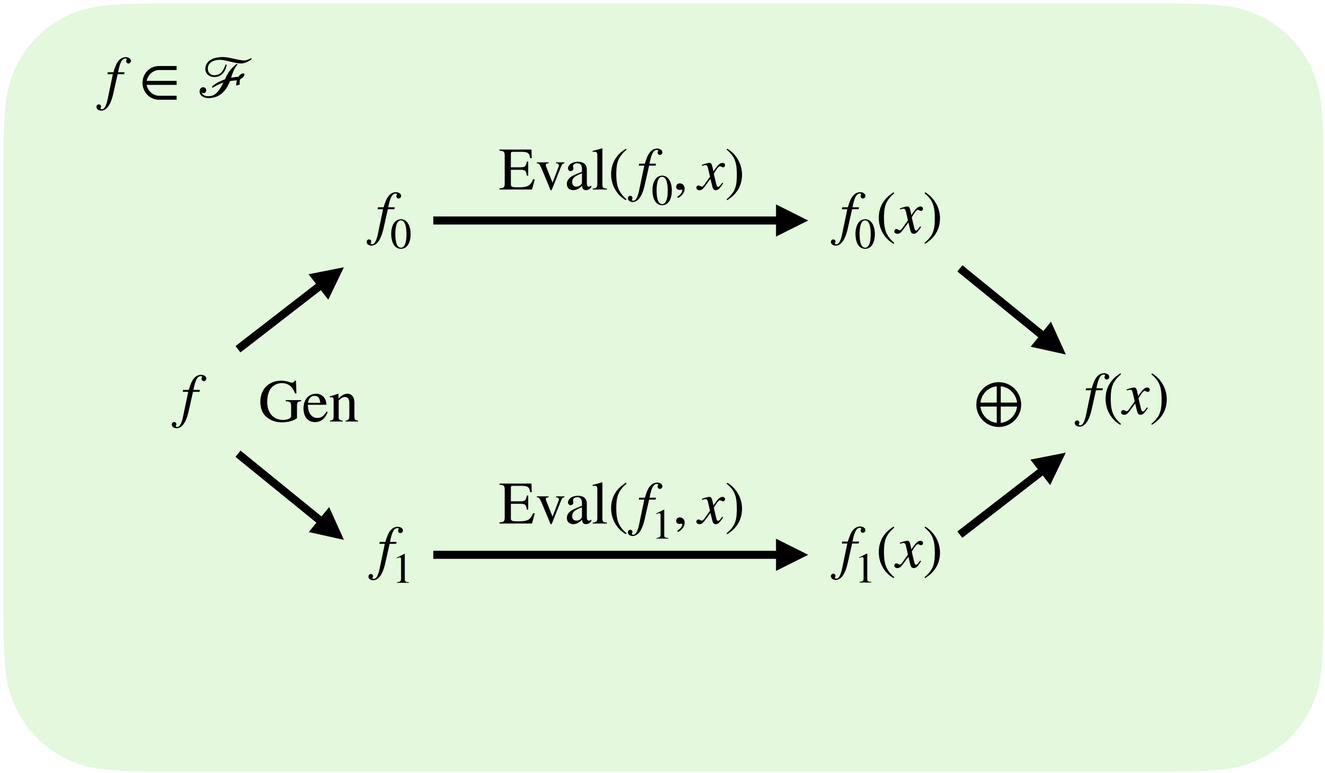


$[f]$

f_1

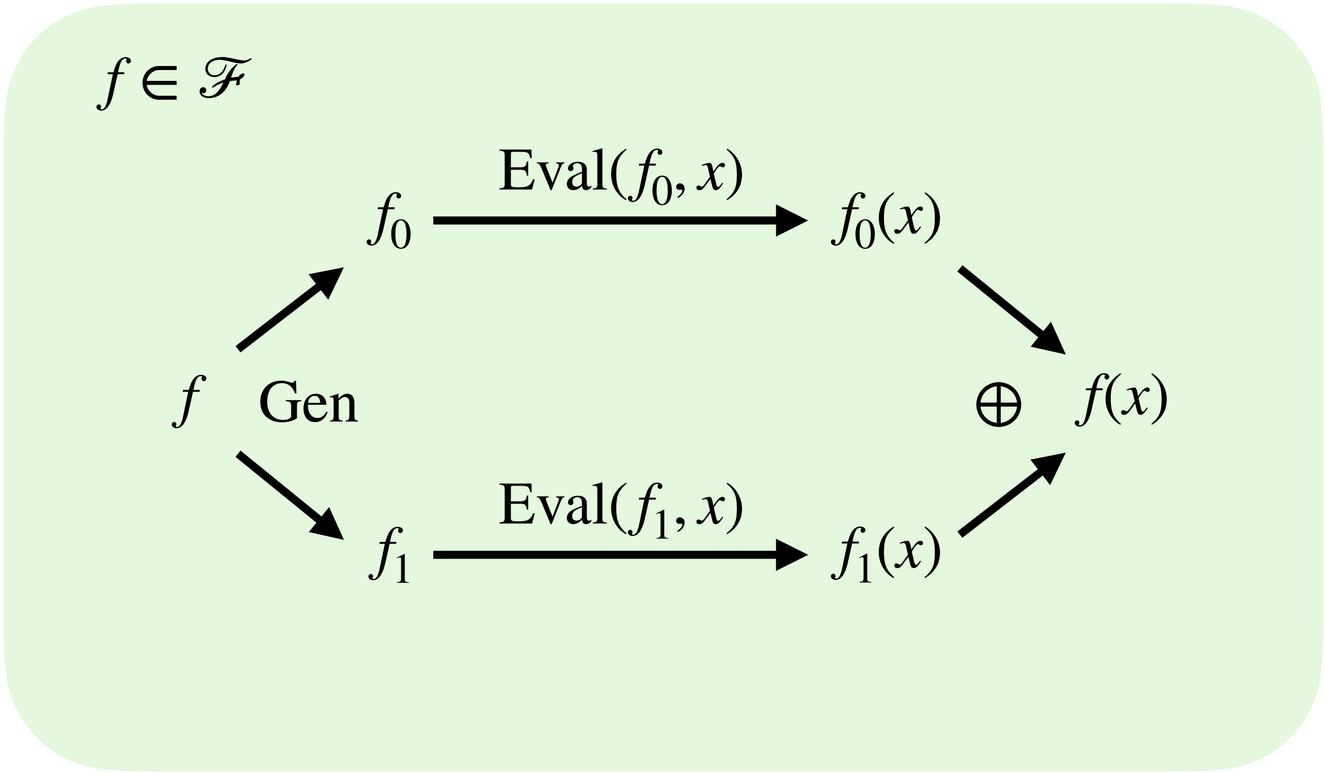


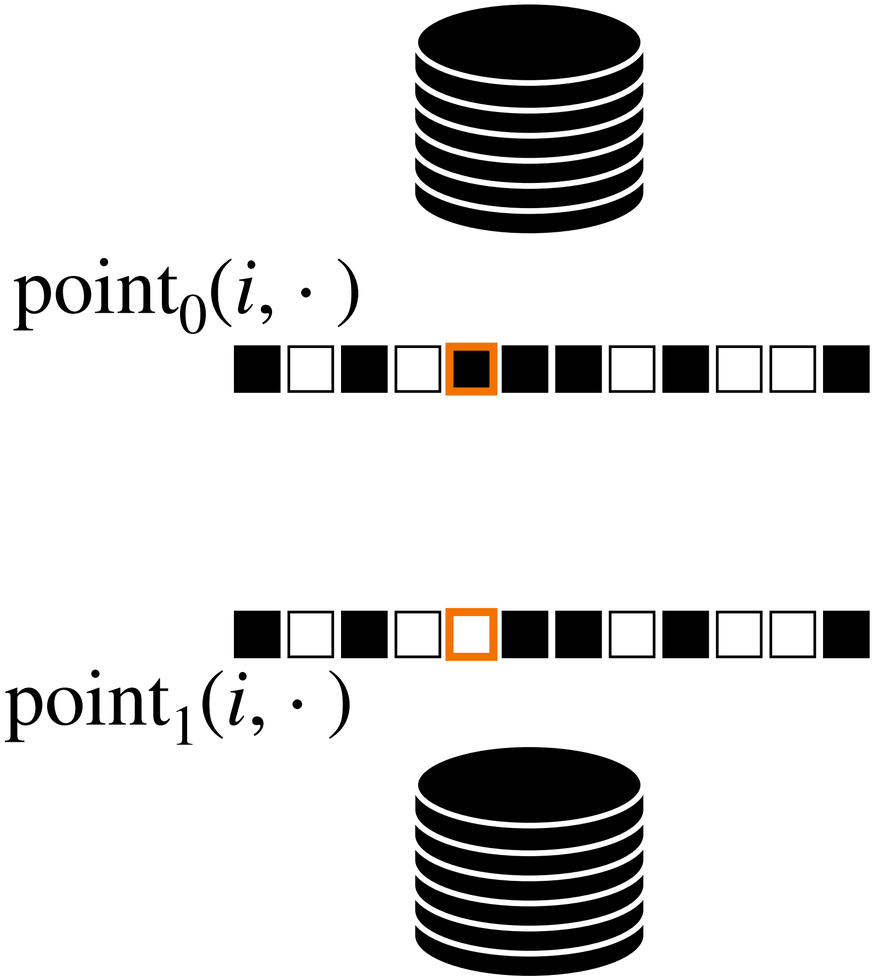
Distributed Point Function



$$\text{point}(i, x) = \begin{cases} 1 & \text{if } x = i \\ 0 & \text{otherwise} \end{cases}$$

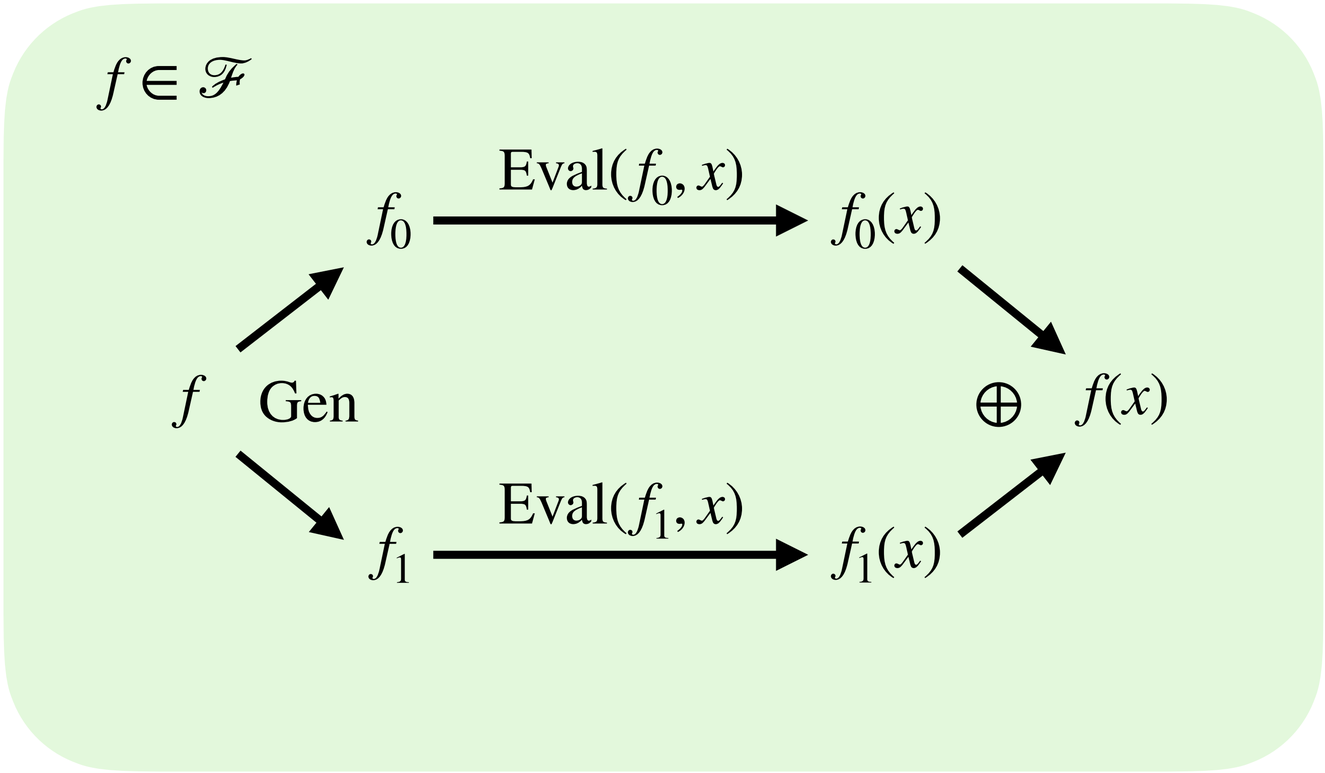
Distributed Point Function

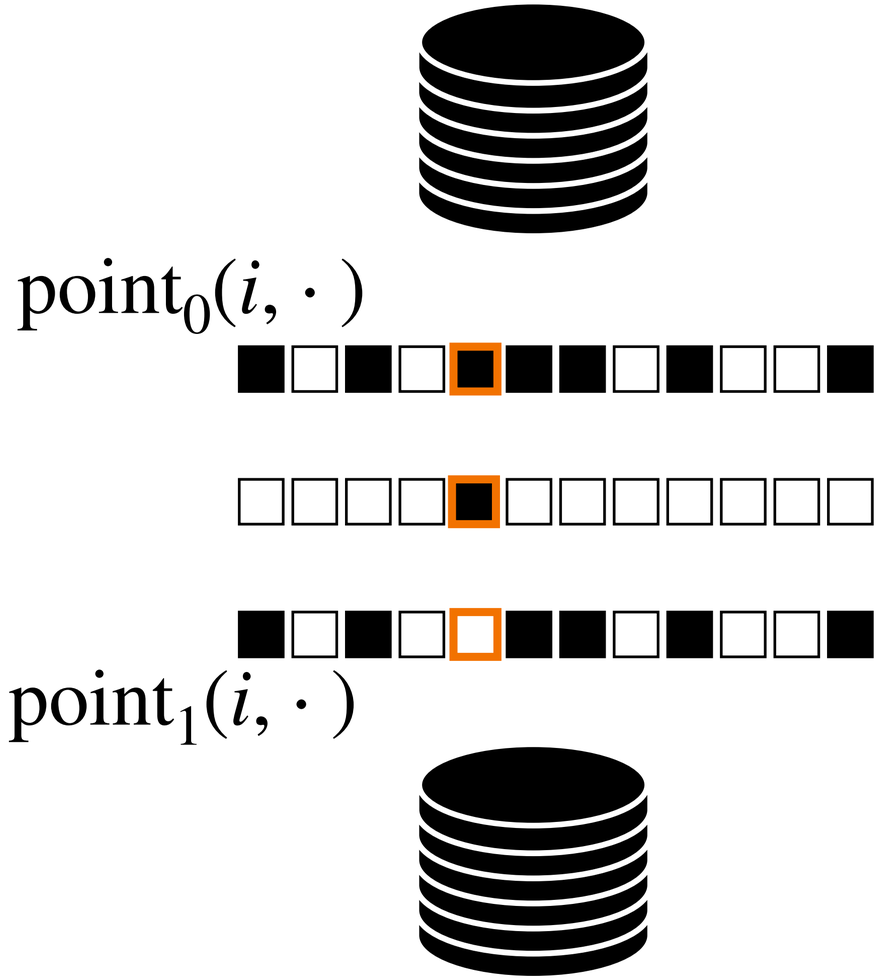




$$\text{point}(i, x) = \begin{cases} 1 & \text{if } x = i \\ 0 & \text{otherwise} \end{cases}$$

Distributed Point Function





$$\text{point}(i, x) = \begin{cases} 1 & \text{if } x = i \\ 0 & \text{otherwise} \end{cases}$$

DPF helps build state-of-the-art OT

Efficient Pseudorandom Correlation Generators: Silent OT Extension and More*

Elette Boyle[1], Geoffroy Couteau[2], Niv Gilboa[3], Yuval Ishai[4], Lisa Kohl[5], and Peter Scholl[5]

[1] IDC Herzliya
[2] Karlsruhe Institute of Technology
[3] Ben-Gurion University of the Negev
[4] Technion
[5] Aarhus University

Abstract. Secure multiparty computation (MPC) often relies on sources of correlated randomness for better efficiency and simplicity. This is particularly useful for MPC with no honest majority, where input-independent correlated randomness enables a lightweight "non-cryptographic" online phase once the inputs are known. However, since the amount of correlated randomness typically scales with the circuit size of the function being computed, securely generating correlated randomness forms an efficiency bottleneck, involving a large amount of communication and storage.
A natural tool for addressing the above limitations is a *pseudorandom correlation generator* (PCG). A PCG allows two or more parties to securely generate long sources of useful correlated randomness via a local expansion of correlated short seeds and no interaction. PCGs enable MPC with *silent preprocessing*, where a small amount of interaction used for securely sampling the seeds is followed by silent local generation of correlated pseudorandomness.
A concretely efficient PCG for Vector OLE correlations was recently obtained by Boyle et al. (CCS 2018) based on variants of the learning parity with noise (LPN) assumption over large fields. In this work, we initiate a systematic study of PCGs and present concretely efficient constructions for several types of useful MPC correlations. We obtain the following main contributions:

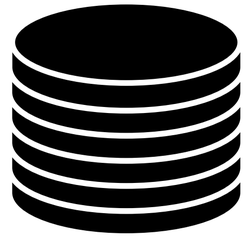
- **PCG foundations.** We give a general security definition for PCGs. Our definition suffices for any MPC protocol satisfying a stronger security requirement that is met by existing protocols. We prove that a stronger security requirement is indeed necessary, and justify our PCG definition by ruling out a stronger and more natural definition.
- **Silent OT extension.** We present the first concretely efficient PCG for oblivious transfer correlations. Its security is based on a variant of the binary LPN assumption and any correlation-robust hash function. We expect it to provide a faster alternative to the IKNP OT extension protocol (Crypto '03) when communication is the bottleneck. We present several applications, including protocols for non-interactive zero-knowledge with bounded-reusable preprocessing from binary LPN, and concretely efficient related-key oblivious pseudorandom functions.
- **PCGs for simple 2-party correlations.** We obtain PCGs for several other types of useful 2-party correlations, including (authenticated) one-time truth-tables and Beaver triples. While the latter PCGs are slower than our PCG for OT, they are still practically feasible. These PCGs are based on a host of assumptions and techniques, including specialized homomorphic secret sharing schemes and pseudorandom generators tailored to their structure.
- **Multiparty correlations.** We obtain PCGs for multiparty correlations that can be used to make the circuit-dependent communication of MPC protocols scale *linearly* (instead of quadratically) with the number of parties.

1 Introduction

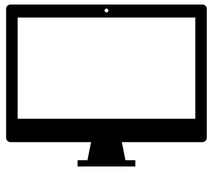
Correlated secret randomness is a valuable resource for secure multi-party computation (MPC). A simple example is a common random key that is given to two parties, who can later use it as a one-time pad for secure message transmission. In the context of MPC, a more useful example is a random *oblivious transfer* (OT) correlation, in which one party is given a pair of random bits (more generally, strings) (s_0, s_1) and the other party is given the pair (r, s_r) for a random bit r. The OT correlation can serve as a basis for general MPC protocols with no honest majority [GMW87, Kil88, IPS08]. Other kinds of two-party correlations that are useful

* This is a full version of [BCG+19].

Two Server PIR

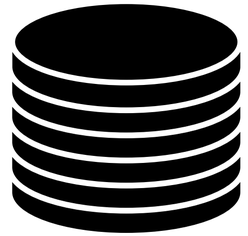


$x_0\, x_1\, x_2\, x_3\, x_4\, x_5\, x_6\, x_7$

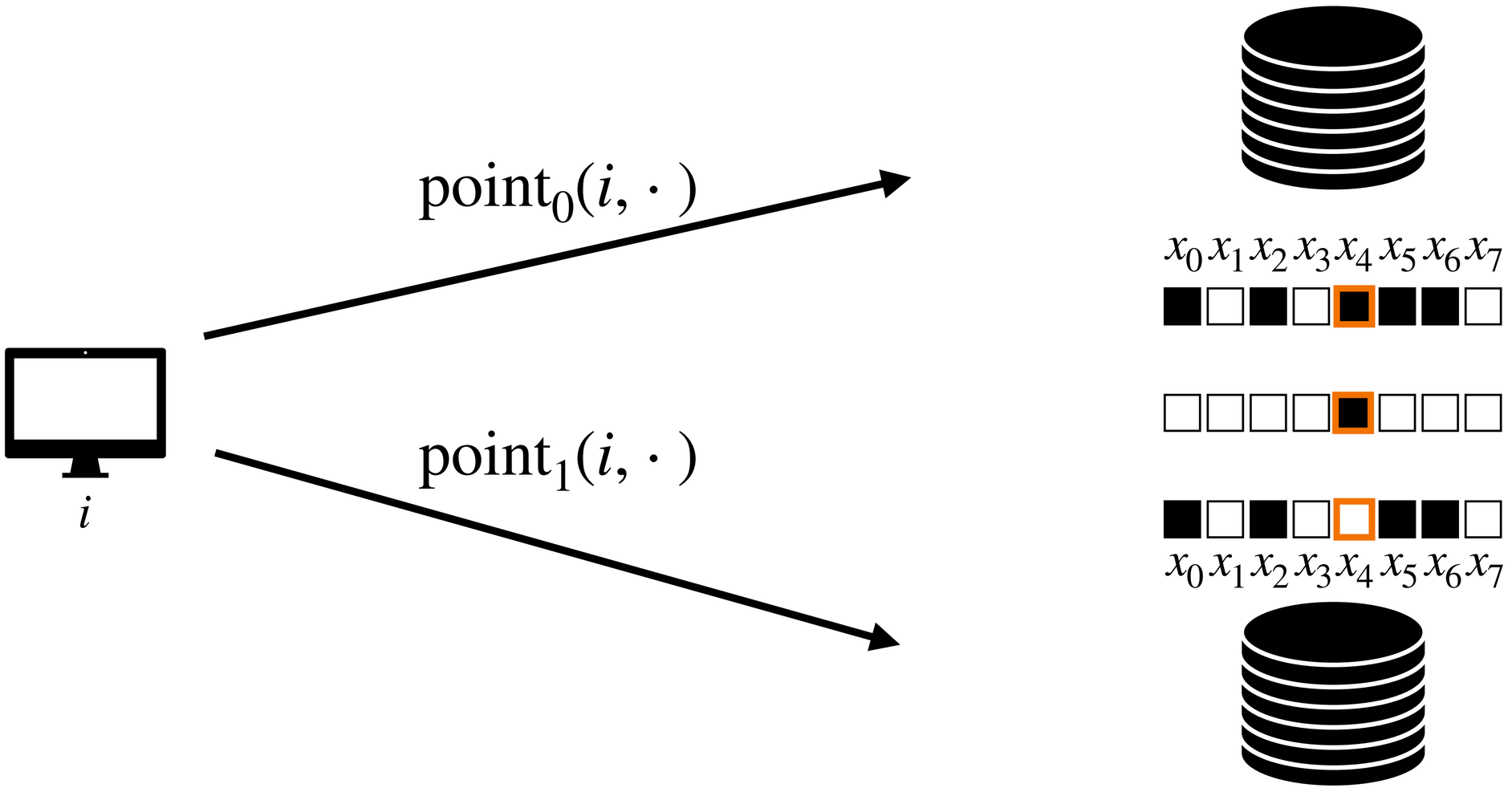


i

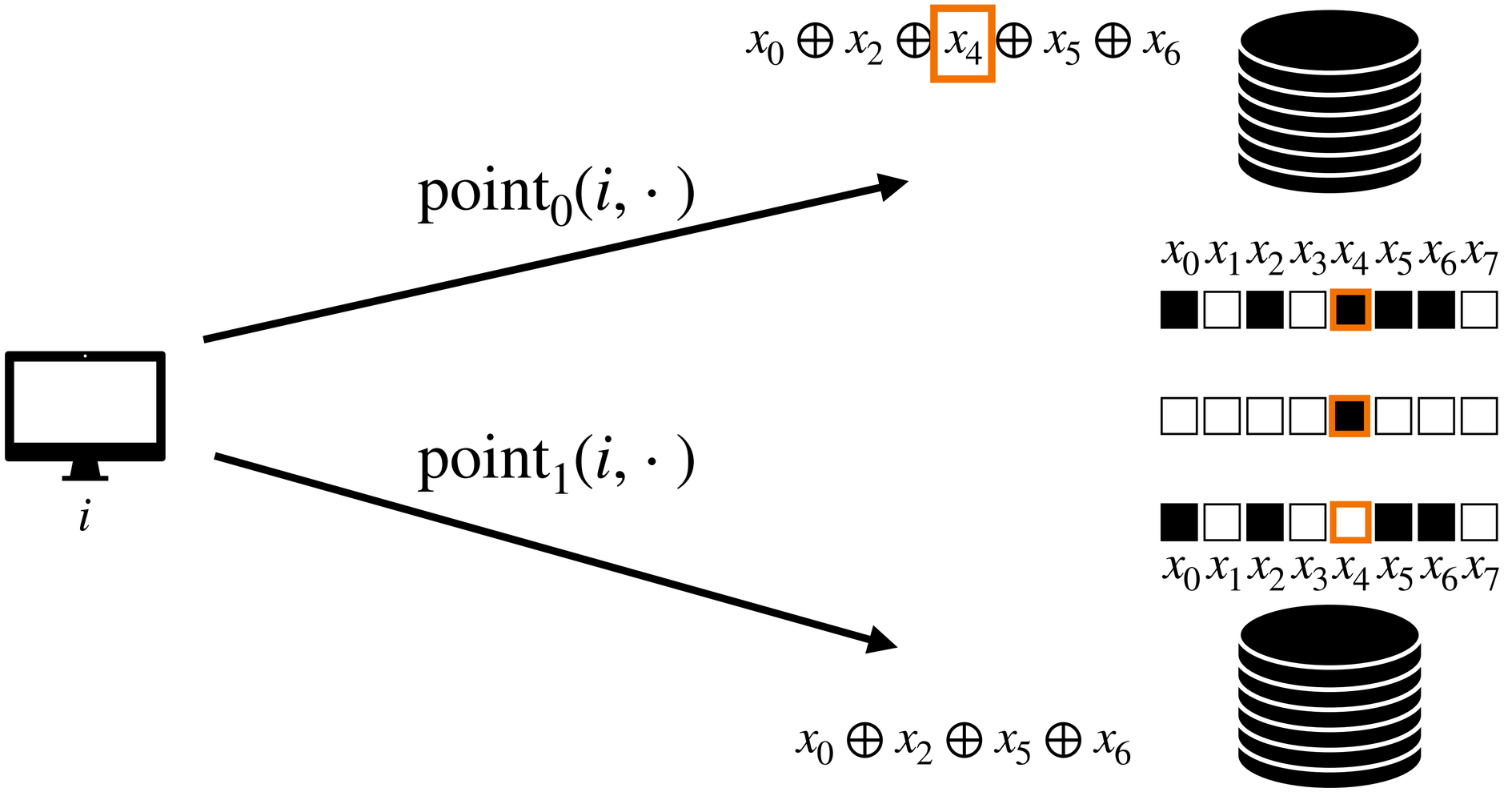
$x_0\, x_1\, x_2\, x_3\, x_4\, x_5\, x_6\, x_7$



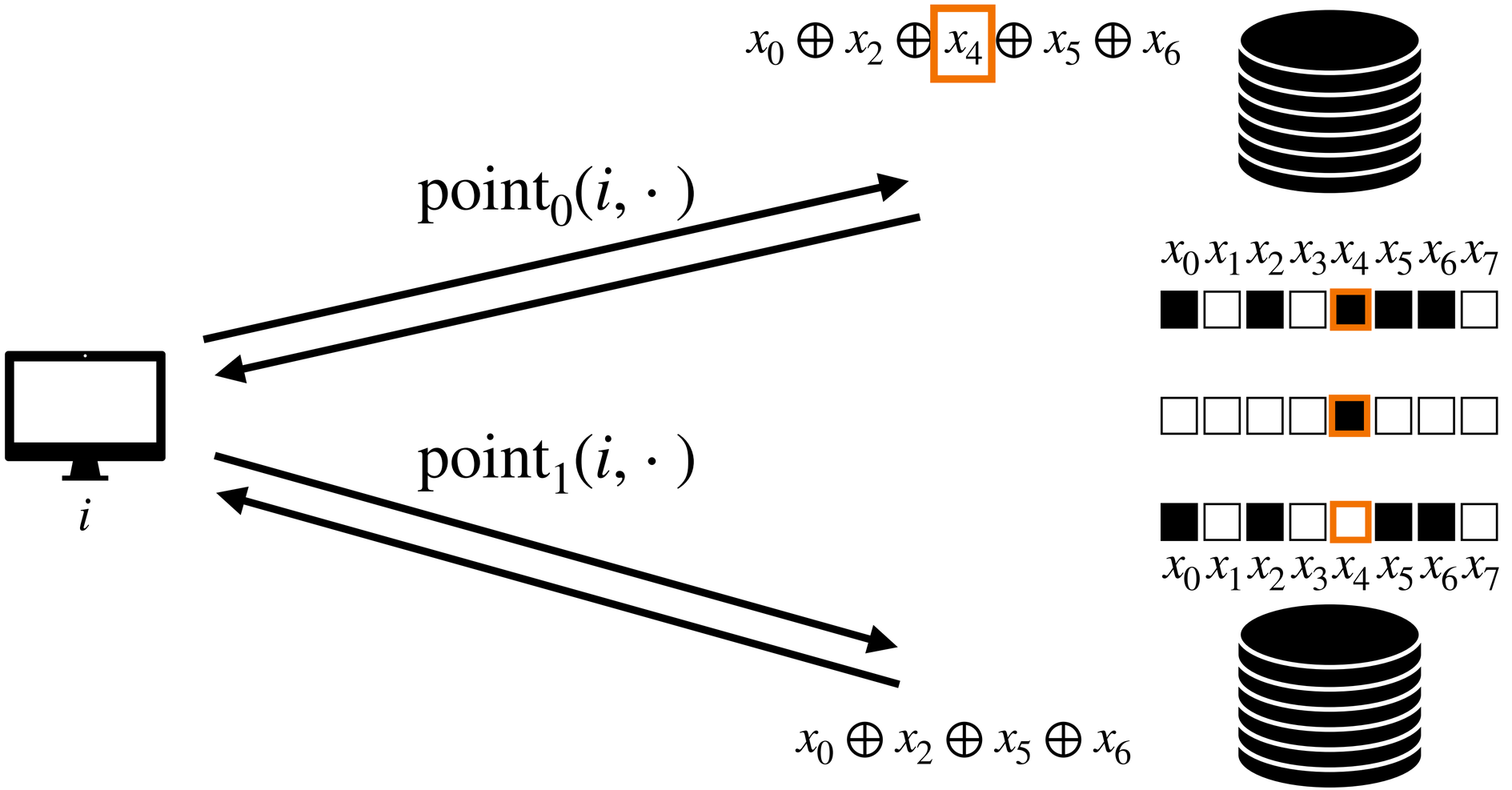
Two Server PIR

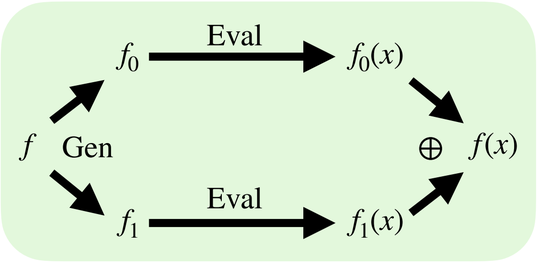


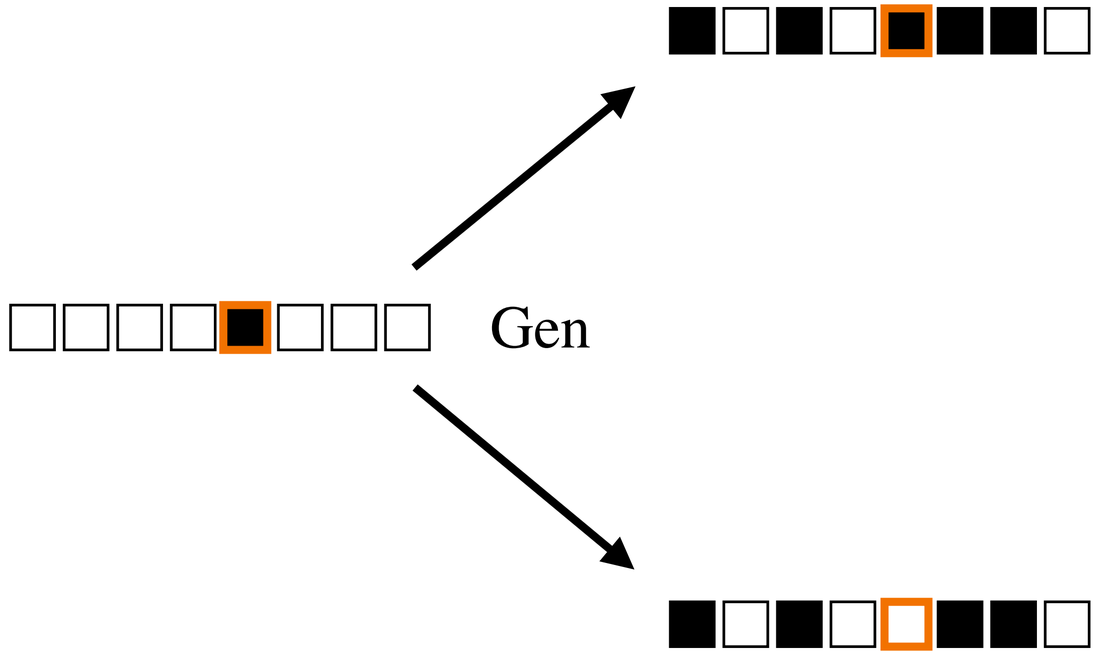
Two Server PIR



Two Server PIR

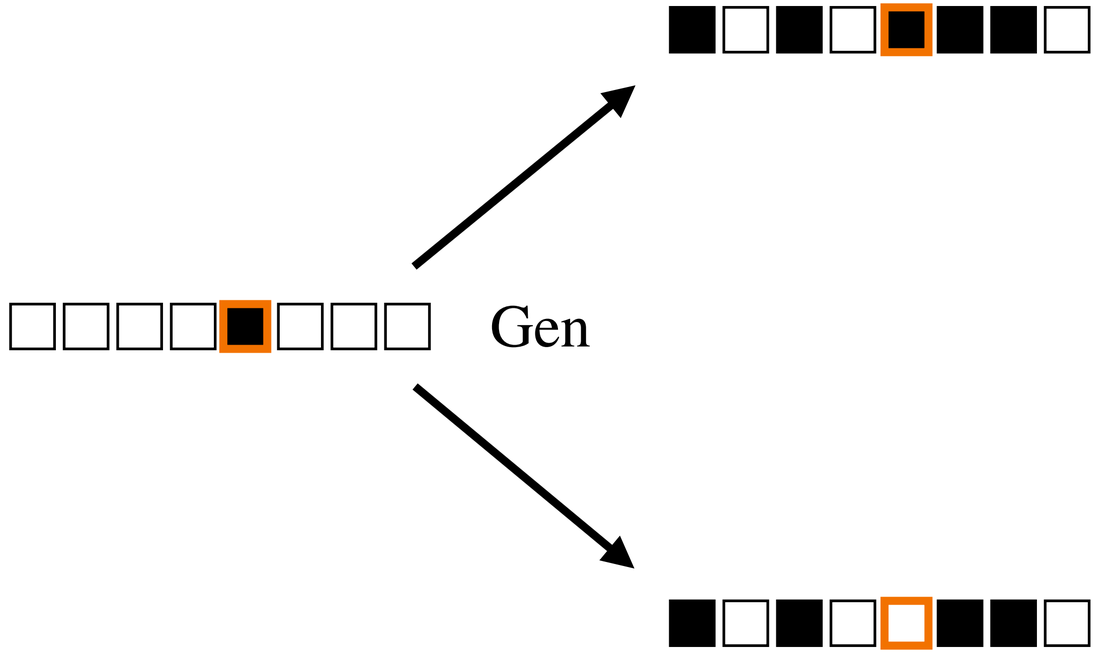






Naive: Shares are n bits long

Goal: Small function shares



Function Secret Sharing: Improvements and Extensions*

Elette Boyle† Niv Gilboa‡ Yuval Ishai§

July 24, 2018

Abstract

Function Secret Sharing (FSS), introduced by Boyle et al. (Eurocrypt 2015), provides a way for additively secret-sharing a function from a given function family $\mathcal{F}$. More concretely, an m-party FSS scheme splits a function $f : \{0,1\}^n \to \mathbb{G}$, for some abelian group $\mathbb{G}$, into functions $f_1, \ldots, f_m$, described by keys $k_1, \ldots, k_m$, such that $f = f_1 + \ldots + f_m$ and every strict subset of the keys hides f. A Distributed Point Function (DPF) is a special case where $\mathcal{F}$ is the family of point functions, namely functions $f_{\alpha,\beta}$ that evaluate to β on the input α and to 0 on all other inputs.

FSS schemes are useful for applications that involve privately reading from or writing to distributed databases while minimizing the amount of communication. These include different flavors of private information retrieval (PIR), as well as a recent application of DPF for large-scale anonymous messaging.

We improve and extend previous results in several ways:

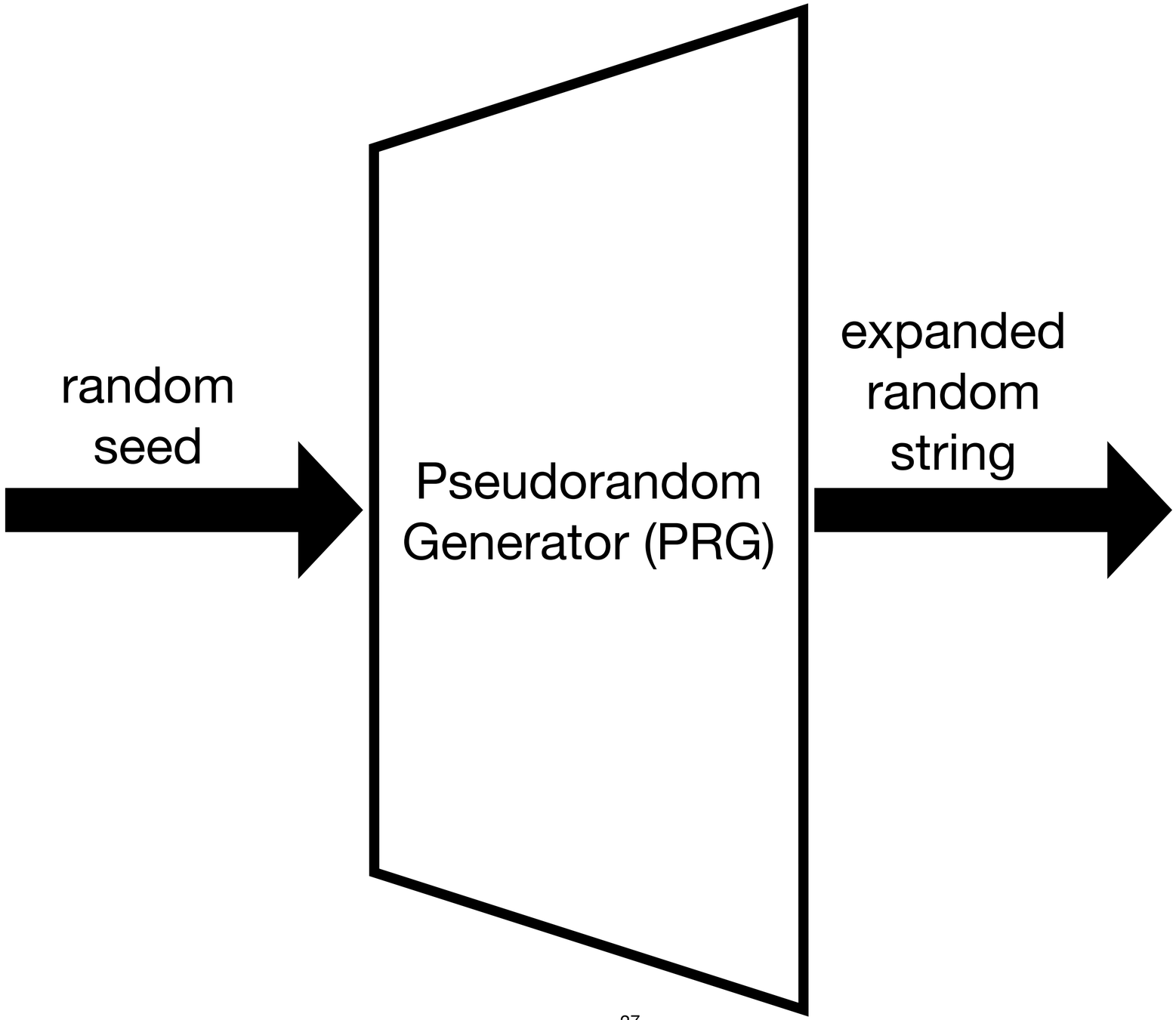
- **Simplified FSS constructions.** We introduce a tensoring operation for FSS which is used to obtain a conceptually simpler derivation of previous constructions and present our new constructions.
- **Improved 2-party DPF.** We reduce the key size of the PRG-based DPF scheme of Boyle et al. roughly by a factor of 4 and optimize its computational cost. The optimized DPF significantly improves the concrete costs of 2-server PIR and related primitives.
- **FSS for new function families.** We present an efficient PRG-based 2-party FSS scheme for the family of *decision trees*, leaking only the topology of the tree and the internal node labels. We apply this towards FSS for multi-dimensional intervals. We also present a general technique for obtaining more expressive FSS schemes by increasing the number of parties.
- **Verifiable FSS.** We present efficient protocols for verifying that keys $(k_1^*, \ldots, k_m^*)$, obtained from a potentially malicious user, are consistent with some $f \in \mathcal{F}$. Such a verification may be critical for applications that involve private writing or voting by many users.

Keywords: Function secret sharing, private information retrieval, secure multiparty computation, homomorphic encryption

*This is a full version of [10].

†IDC Herzliya, Israel, eboyle@alum.mit.edu.
‡Ben Gurion University, Israel gilboan@bgu.ac.il.
§Technion, Israel, and UCLA, USA. yuvali@cs.technion.ac.il.

$$\approx \log n \cdot \lambda \text{ bits}$$


random
seed
Pseudorandom
Generator (PRG)
expanded
random
string

How to Construct Random Functions

ODED GOLDREICH, SHAFI GOLDWASSER,
AND SILVIO MICALI

Massachusetts Institute of Technology, Cambridge, Massachusetts

Abstract. A constructive theory of randomness for functions, based on computational complexity, is developed, and a pseudorandom function generator is presented. This generator is a deterministic polynomial-time algorithm that transforms pairs (g, r), where g is *any* one-way function and r is a random k-bit string, to polynomial-time computable functions $f_r: \{1, \ldots, 2^k\} \to \{1, \ldots, 2^k\}$. These f_r's cannot be distinguished from *random* functions by any probabilistic polynomial-time algorithm that asks and receives the value of a function at arguments of its choice. The result has applications in cryptography, random constructions, and complexity theory.

Categories and Subject Descriptors: F.0 [**Theory of Computation**]: General; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*computability theory*; G.0 [**Mathematics of Computing**]: General; G.3 [**Mathematics of Computing**]: Probability and Statistics—*probabilistic algorithms; random number generation*

General Terms: Algorithms, Security, Theory

Additional Key Words and Phrases: Cryptography, one-way functions, prediction problems, randomness

I have set up on a Manchester computer a small programme using only 1000 units of storage, whereby the machine supplied with one sixteen figure number replies with another within two seconds. I would defy anyone to learn from these replies sufficient about the programme to be able to predict any replies to untried values.

A. TURING

1. *Introduction*

What is meant by saying that certain functions "behave randomly"?

In this paper we provide a precise answer to the above question. We then present an efficient way to construct functions that behave randomly, if one-way functions exist. We conclude by demonstrating applications of our construction.

Randomness has attracted much attention in the second half of this century. However, most of the previous work focused on measuring the randomness of *strings*.

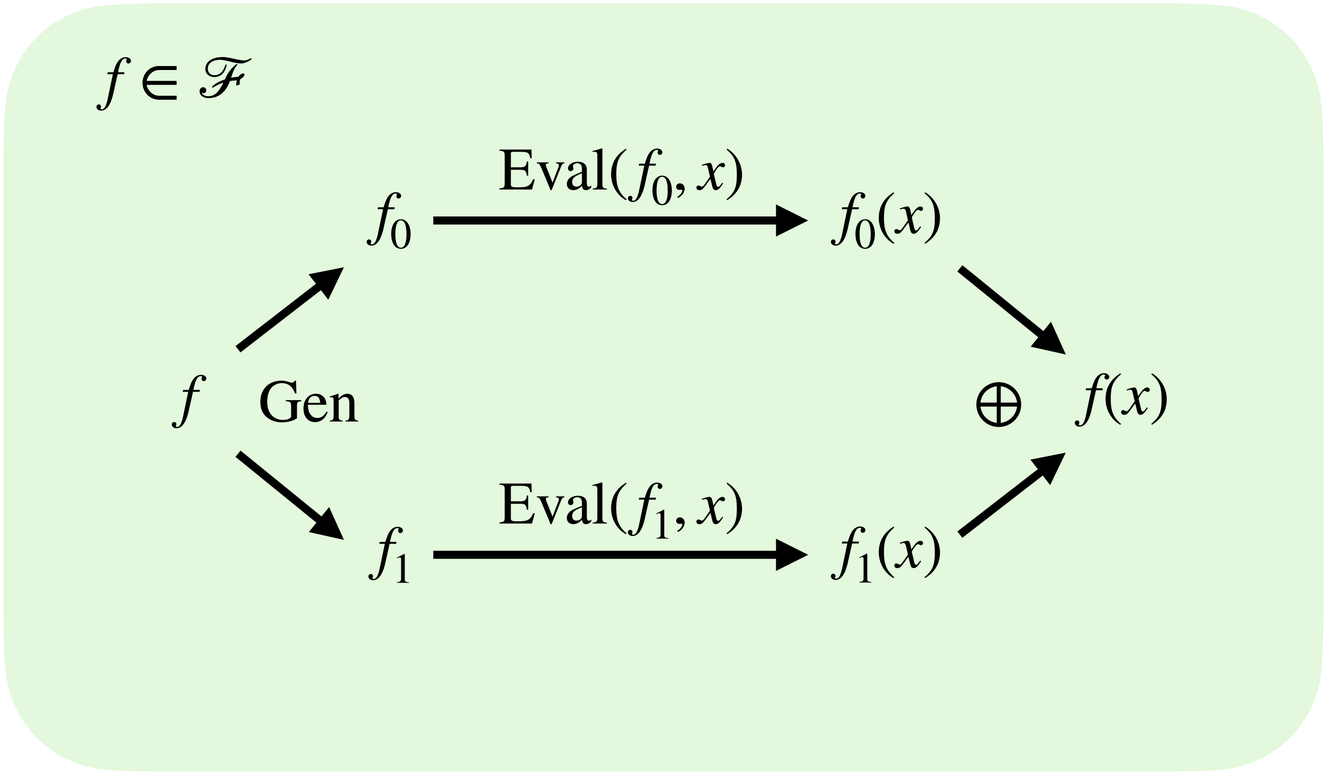
O. Goldreich was supported in part by a Weizmann postdoctoral fellowship; S. Goldwasser was supported in part by an IBM faculty development award (1983) and National Science Foundation grant DCR 85-09905; and S. Micali was supported by a National Science Foundation grant DCR 84-13577 and an IBM faculty development Award (1984).

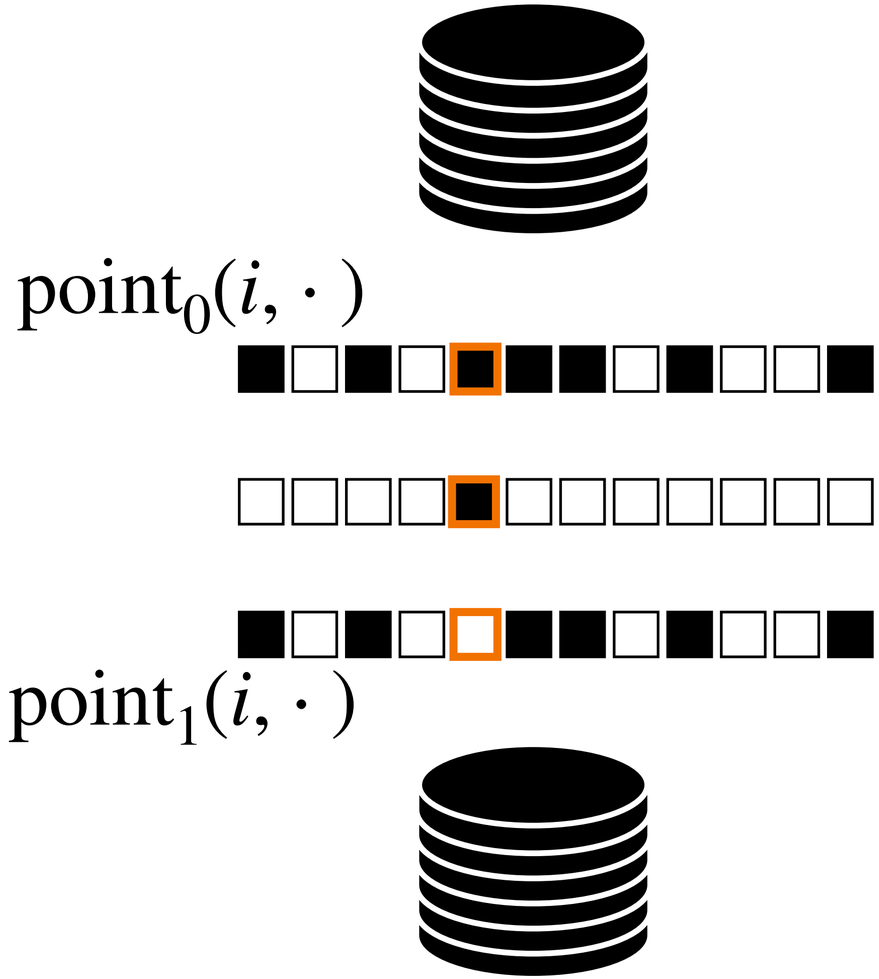
Authors' present addresses: O. Goldreich, Computer Science Department, Technion, Haifa 32000 Israel; S. Goldwasser and S. Micali, Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139.

Journal of the Association for Computing Machinery, Vol. 33, No. 4, October 1986, pp. 792-807.

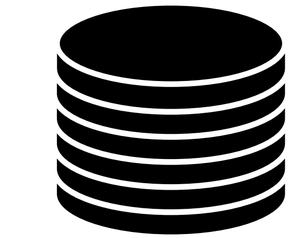
How to build a PRF from a PRG

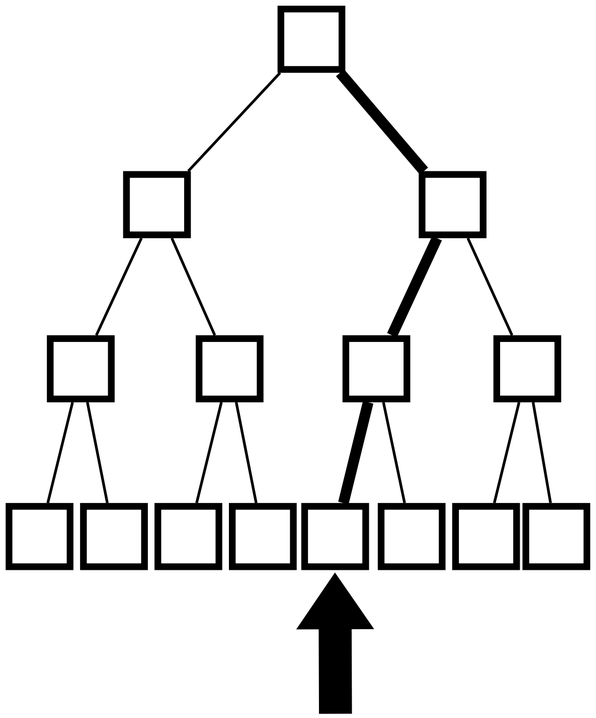
Goal: Distributed Point Function

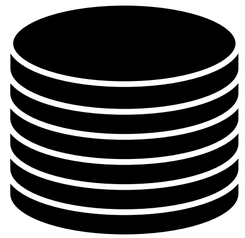


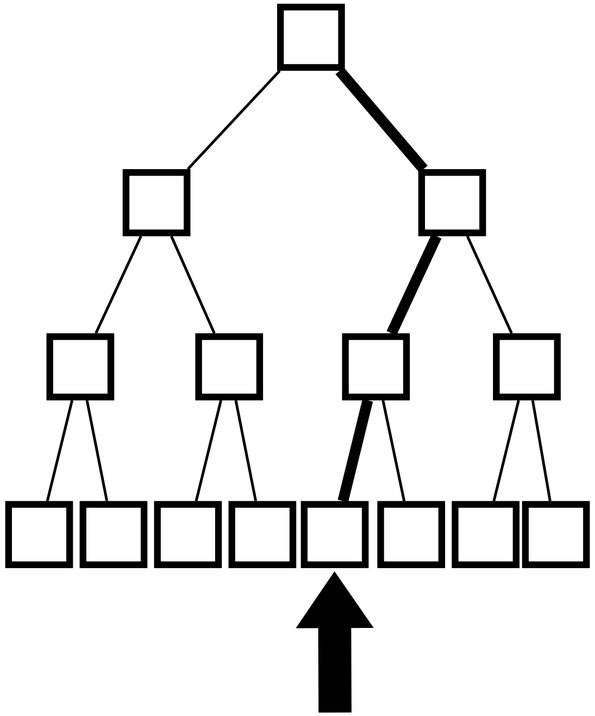


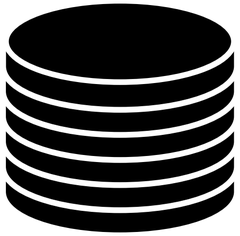
$$\text{point}(i, x) = \begin{cases} 1 & \text{if } x = i \\ 0 & \text{otherwise} \end{cases}$$



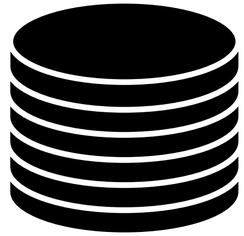


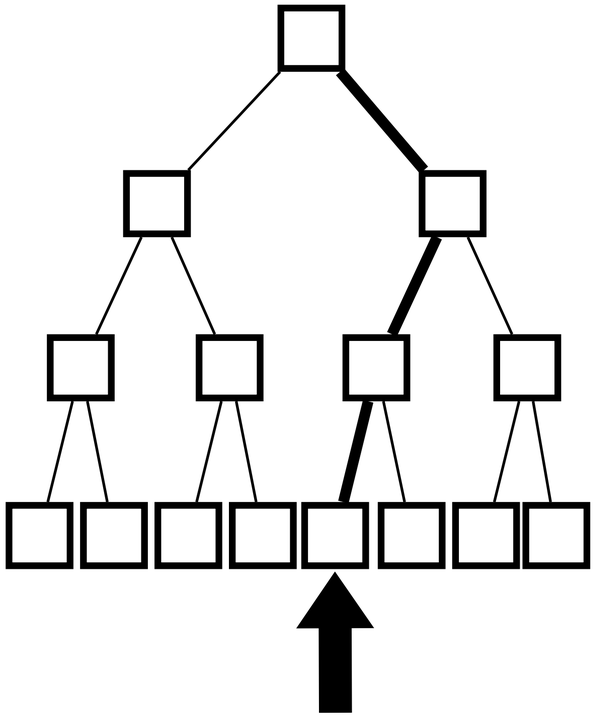


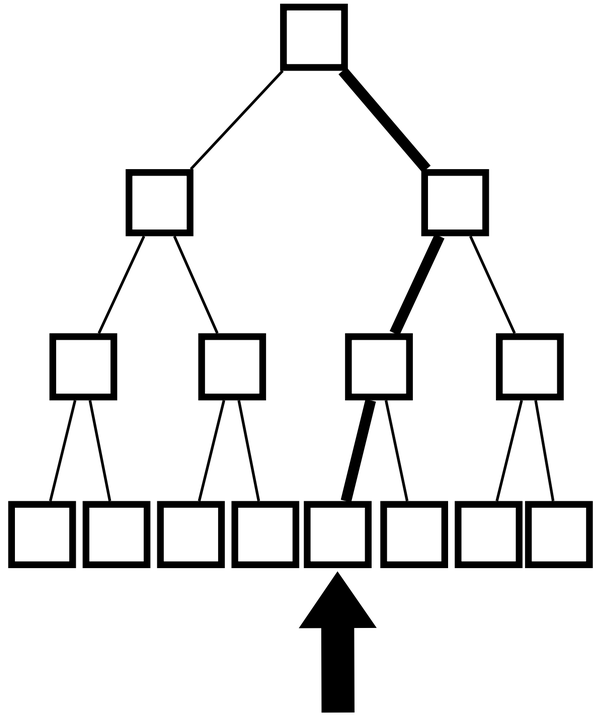


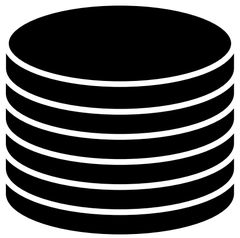


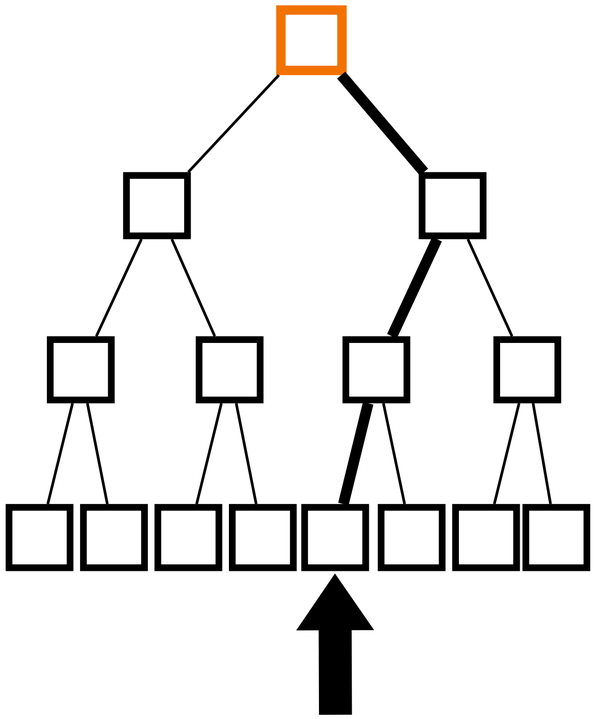
High level: Client will ensure servers have matching PRG seeds “just off” the path to the target leaf

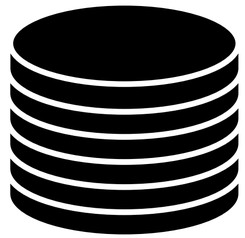


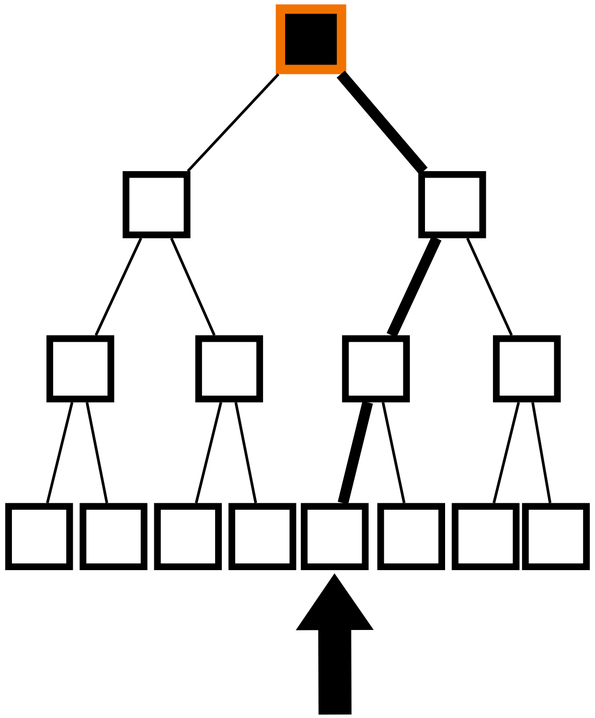


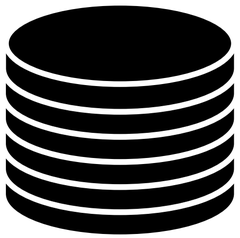


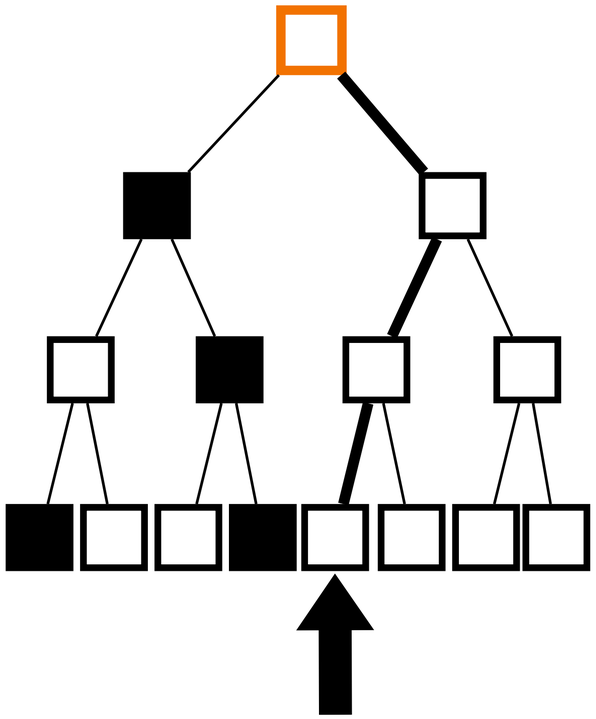


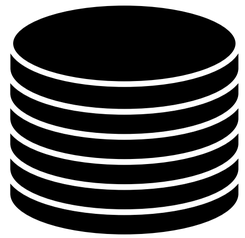


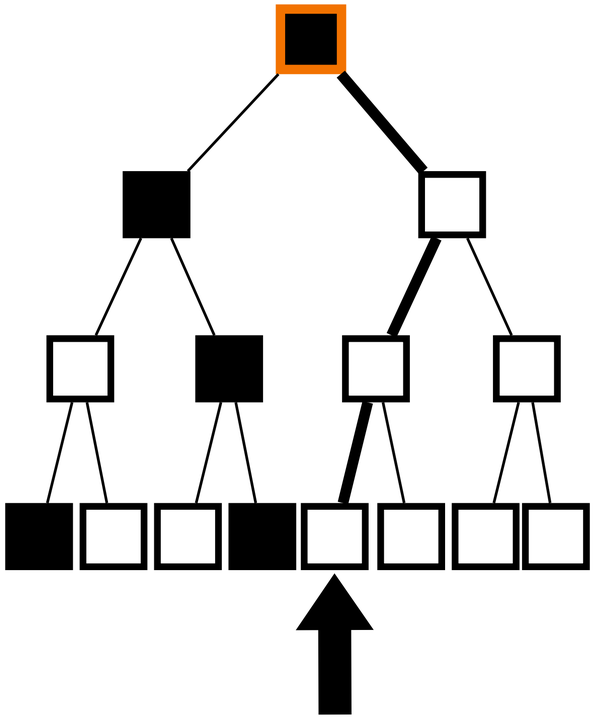


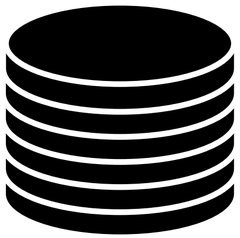


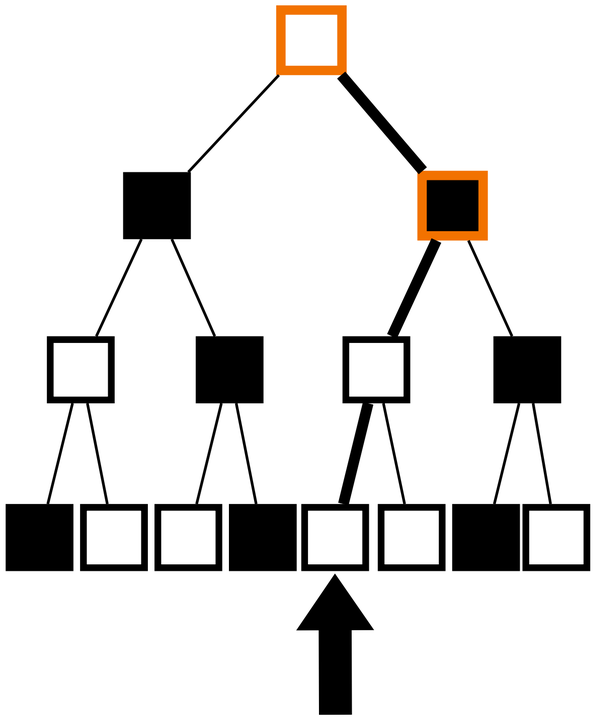


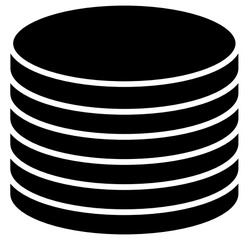


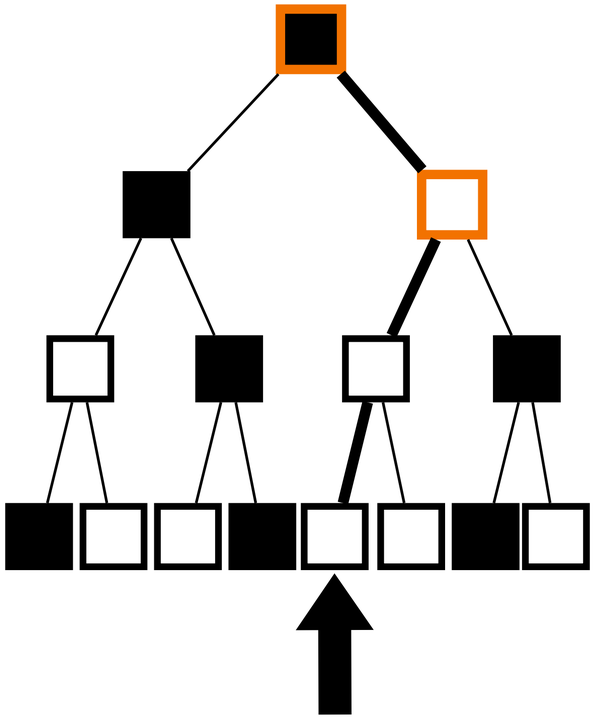


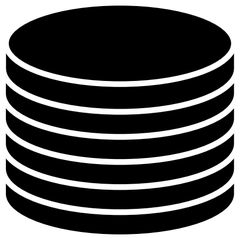


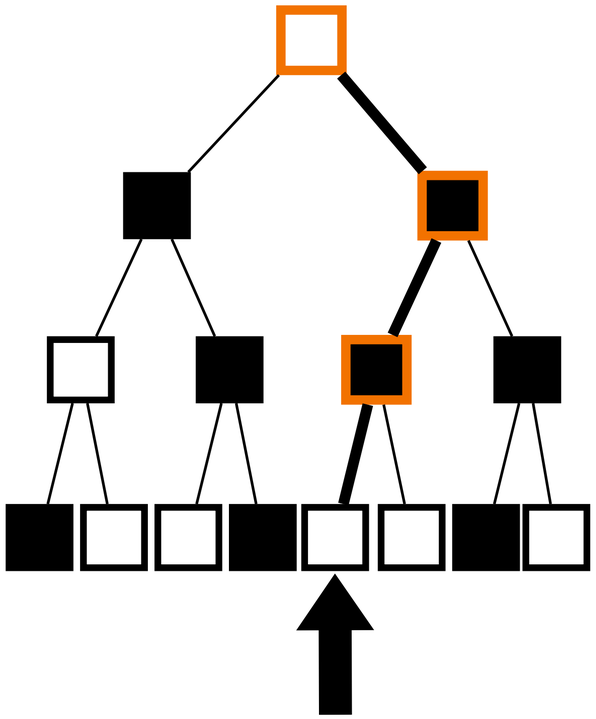


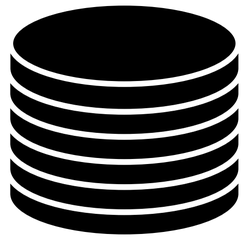


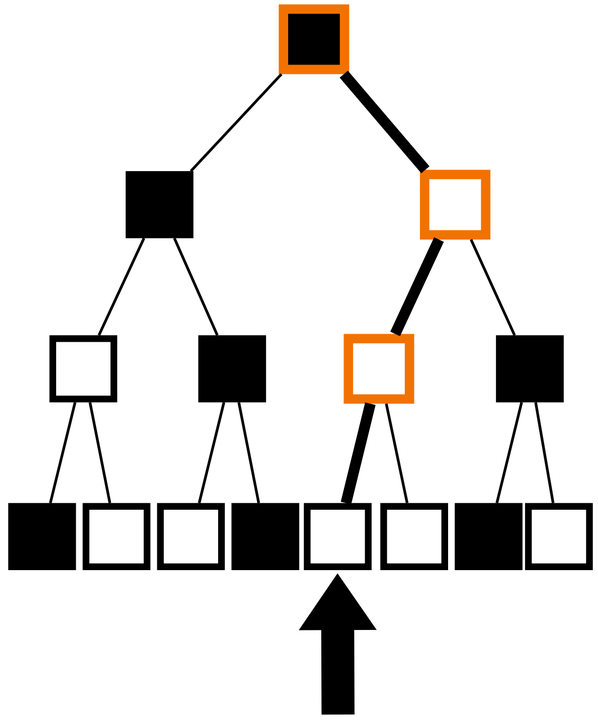


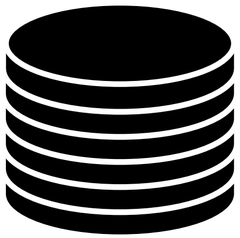


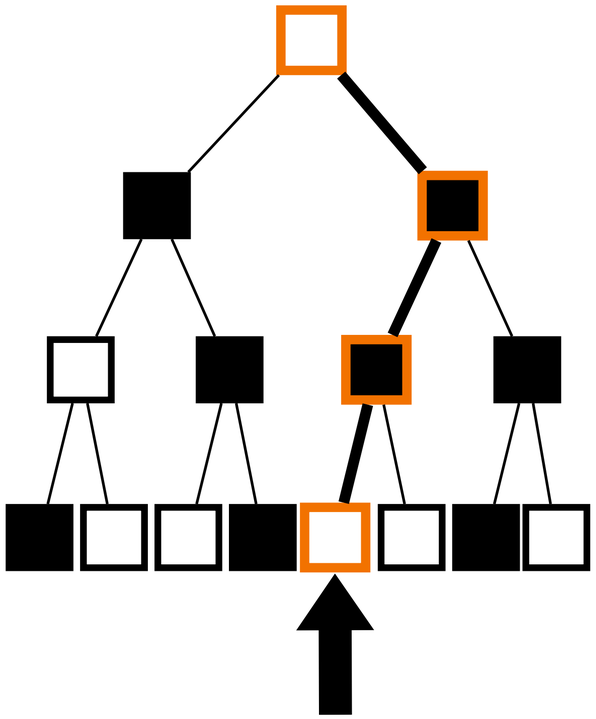


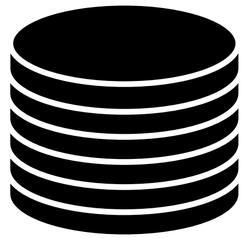


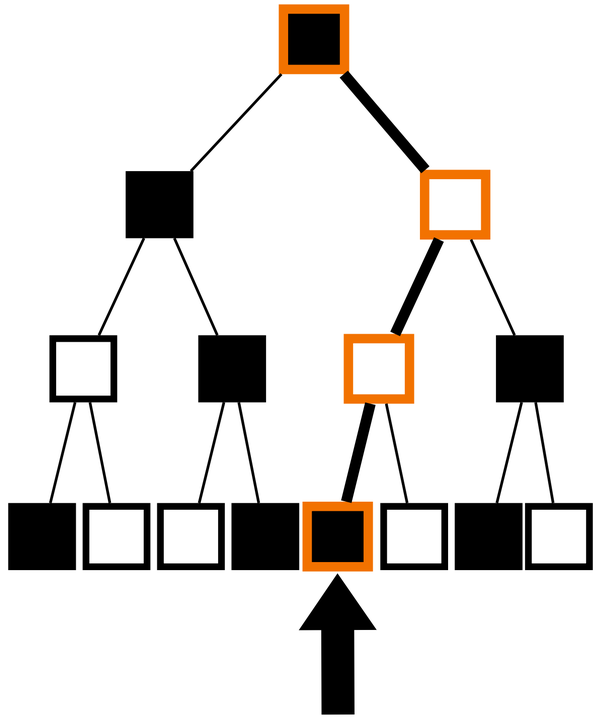






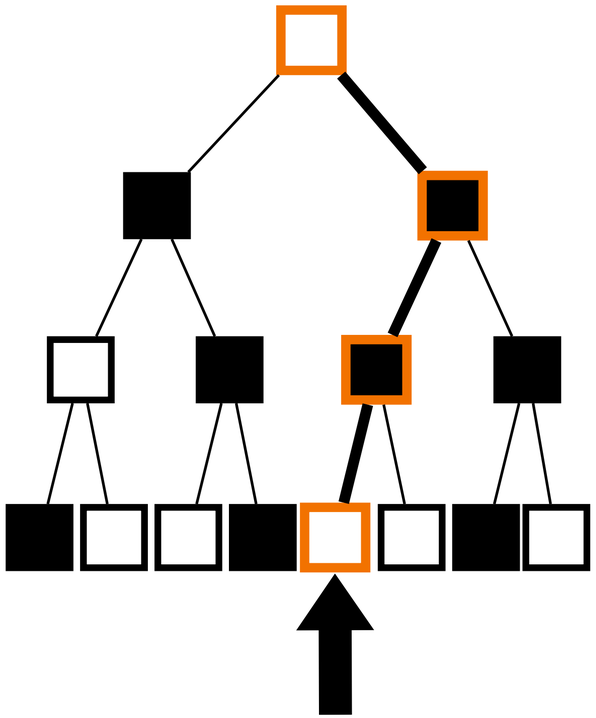


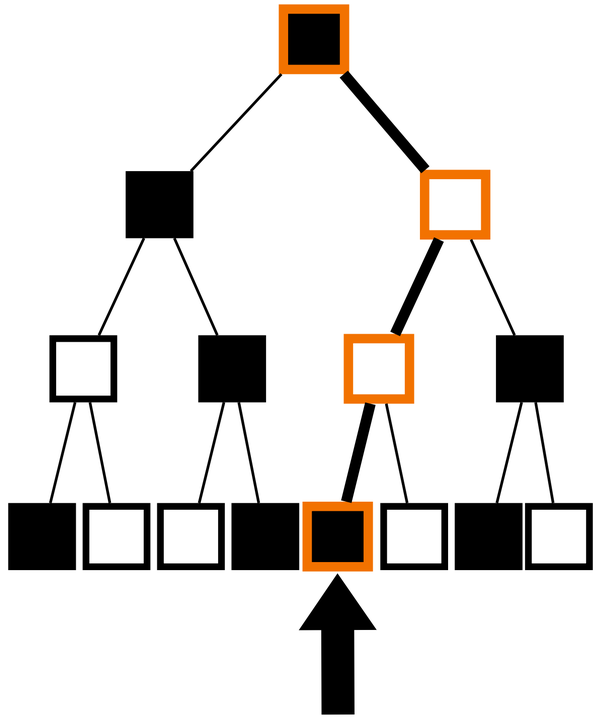




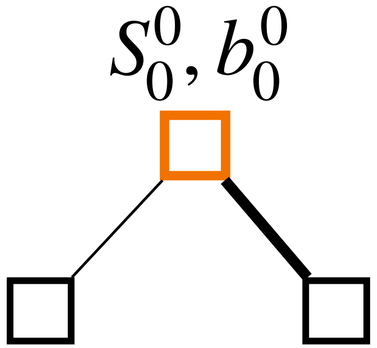
Tree is generated by PRG

Force equality off the path by sending extra information

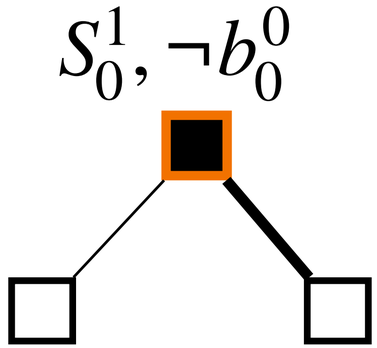




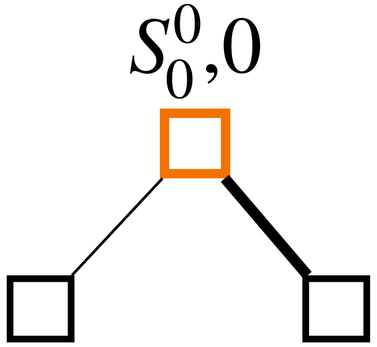
The tricky part is thinking about nodes in the tree exactly on the path



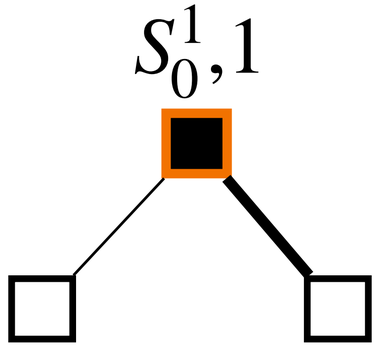
$[\$,1]$

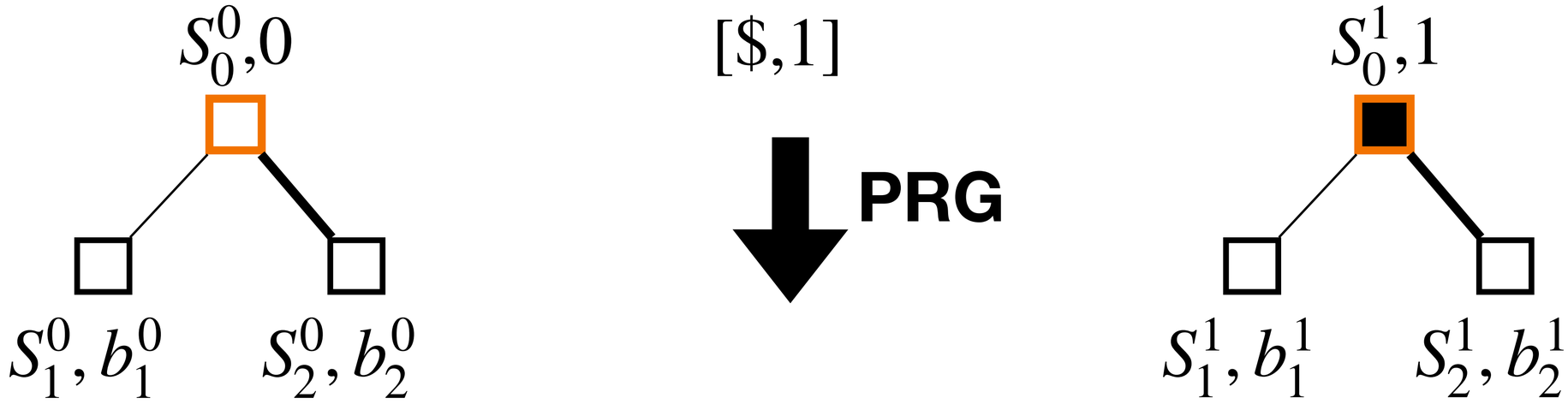


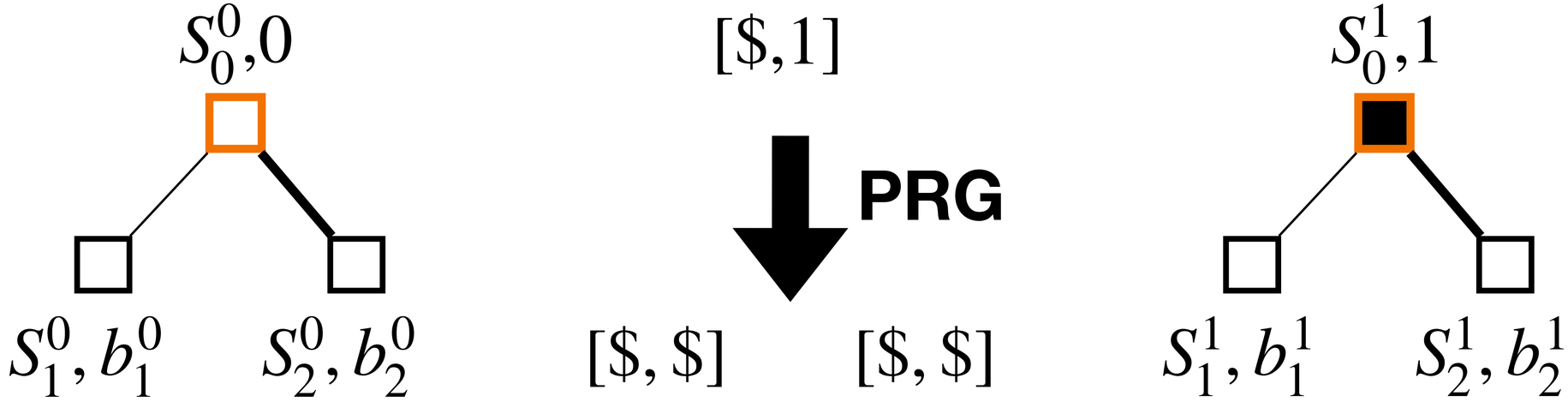
Servers disagree on the path, we must arrange that they agree just off the path

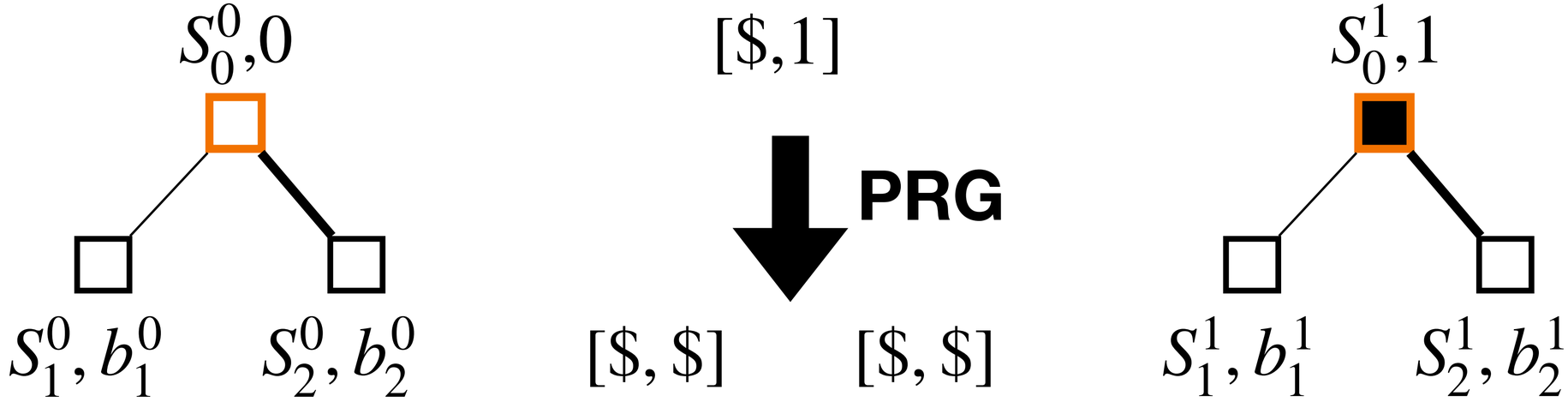


[$,1]




$S_0^0, 0$
S_1^0, b_1^0
S_2^0, b_2^0
[$,1]
PRG
$S_0^1, 1$
S_1^1, b_1^1
S_2^1, b_2^1


$S_0^0, 0$
S_1^0, b_1^0
S_2^0, b_2^0
[$, 1]
PRG
[$, $]
[$, $]
$S_0^1, 1$
S_1^1, b_1^1
S_2^1, b_2^1



Goal: [0,0] [$,1]

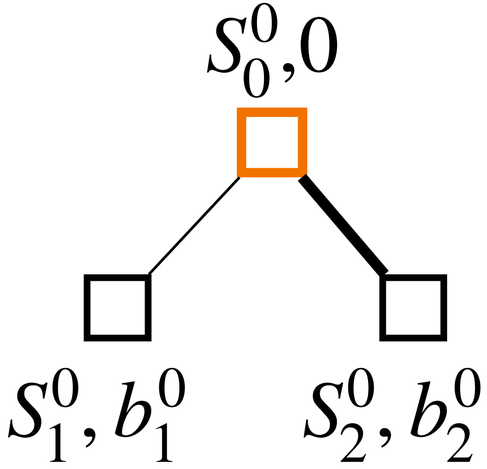
Key includes *correction words*

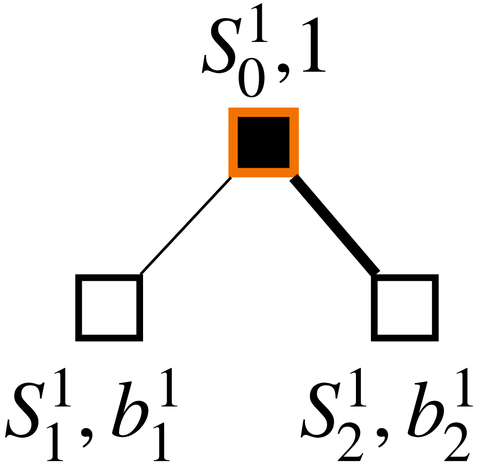
$S_1^0 \oplus S_1^1$

$b_1^0 \oplus b_1^1$

$S_2^0 \oplus S_2^1 \oplus \$$

$b_2^0 \oplus b_2^1 \oplus 1$





	[\$, \$]	[\$, \$]
Goal:	[0,0]	[\$,1]

Two servers **conditionally** use the correction word, depending on their bits b

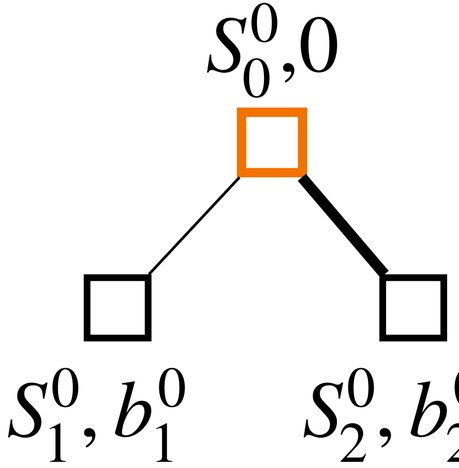
Key includes *correction words*

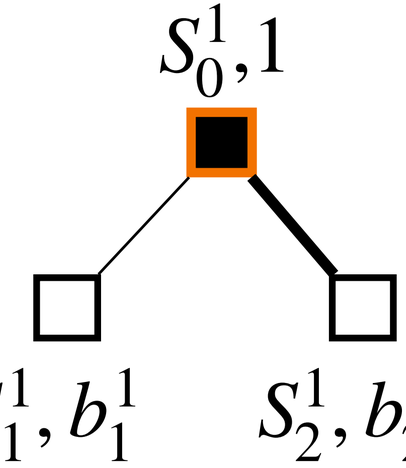
$S_1^0 \oplus S_1^1$

$b_1^0 \oplus b_1^1$

$S_2^0 \oplus S_2^1 \oplus \$$

$b_2^0 \oplus b_2^1 \oplus 1$





$[\$, \$]$ $[\$, \$]$

S_1^0, b_1^0 $S_2^0 \oplus \$, b_2^0 \oplus 1$

Goal: $[0,0]$ $[\$,1]$

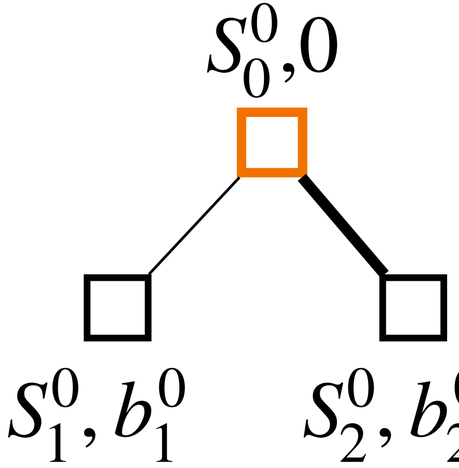
Key includes *correction words*

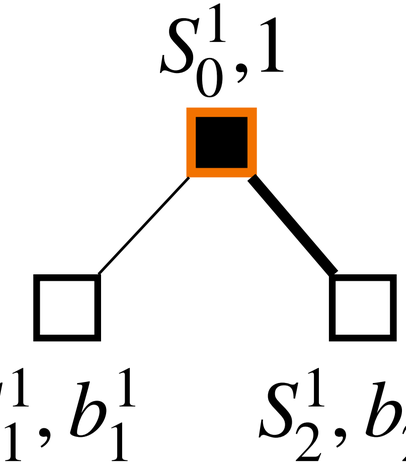
$$\Delta = S_1^0 \oplus S_1^1$$

$$b_1^0 \oplus b_1^1$$

~~$S_2^0 \oplus S_2^1 \oplus \$$~~

$$b_2^0 \oplus b_2^1 \oplus 1$$





Goal: $[0,0]$ $[\$,1]$

$[\$, \$]$ $[\$, \$]$

S_1^0, b_1^0 $S_2^1 \oplus \Delta, b_2^0 \oplus 1$

Key includes *correction words*

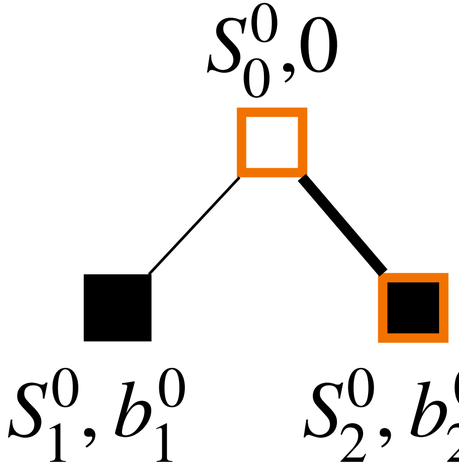
$\lambda + 2$ bits

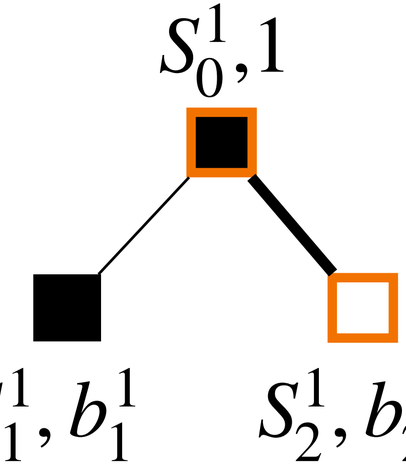
$\Delta = S_1^0 \oplus S_1^1$

$b_1^0 \oplus b_1^1$

~~$S_2^0 \oplus S_2^1 \oplus \$$~~

$b_2^0 \oplus b_2^1 \oplus 1$





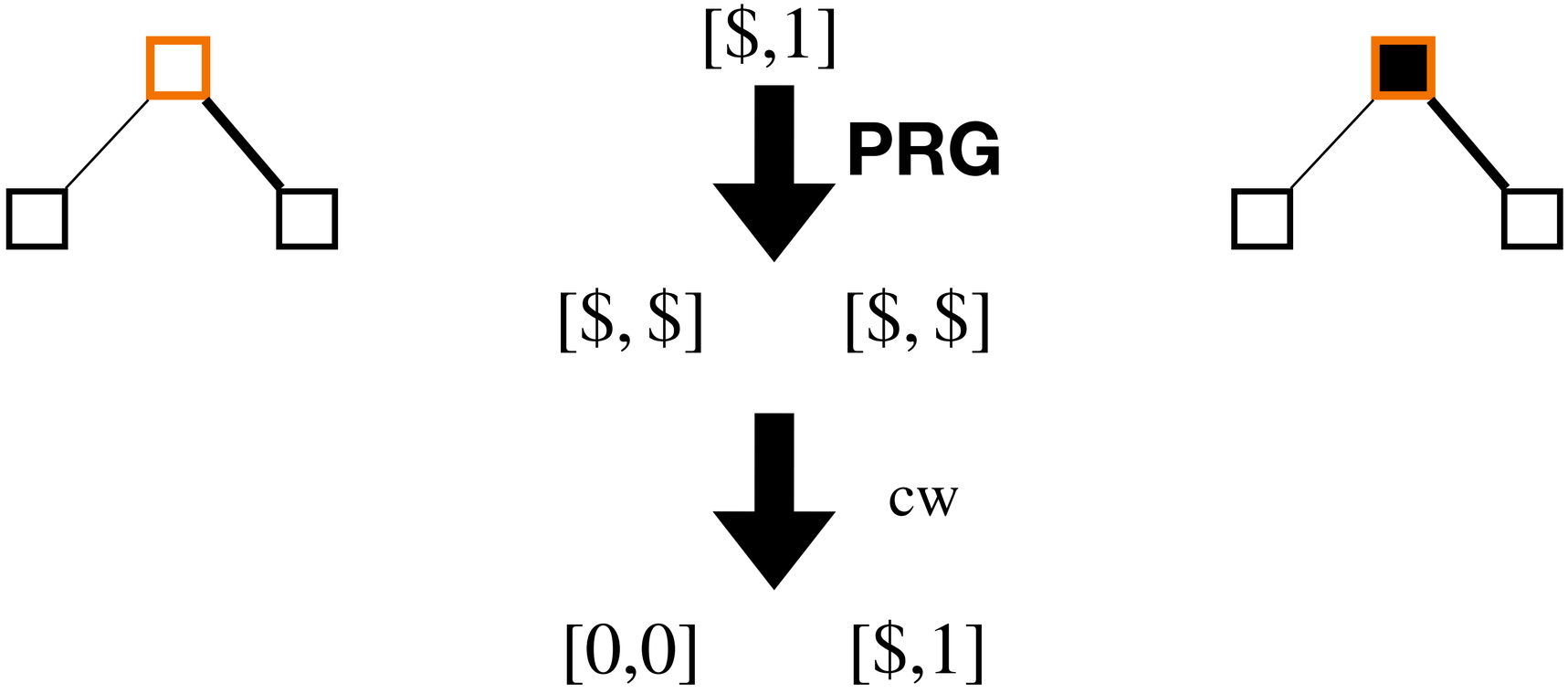
$[\$, \$]$ $[\$, \$]$

S_1^0, b_1^0 $S_2^1 \oplus \Delta, b_2^0 \oplus 1$

Goal: $[0,0]$ $[\$,1]$

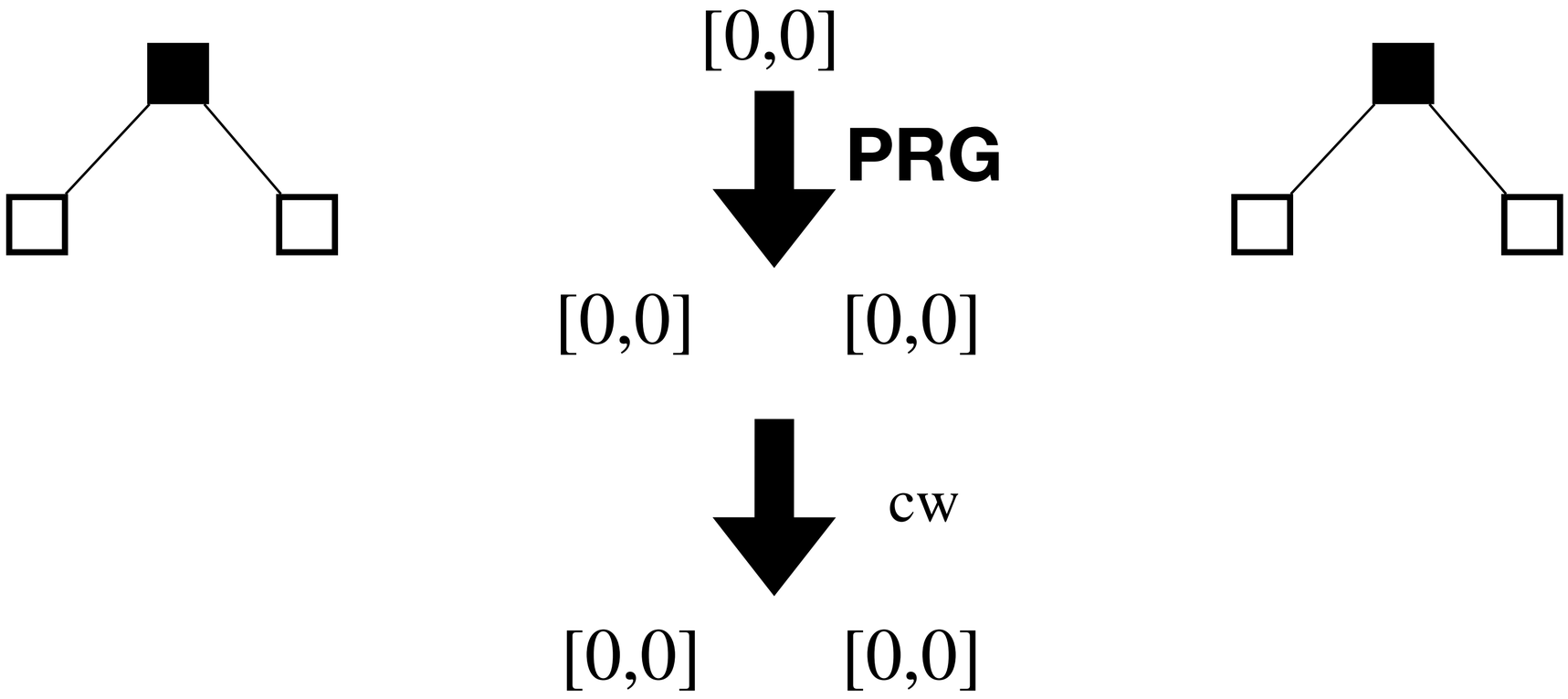
Key includes *correction words*

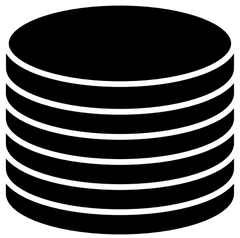
$\lambda + 2$ bits

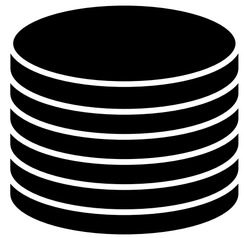


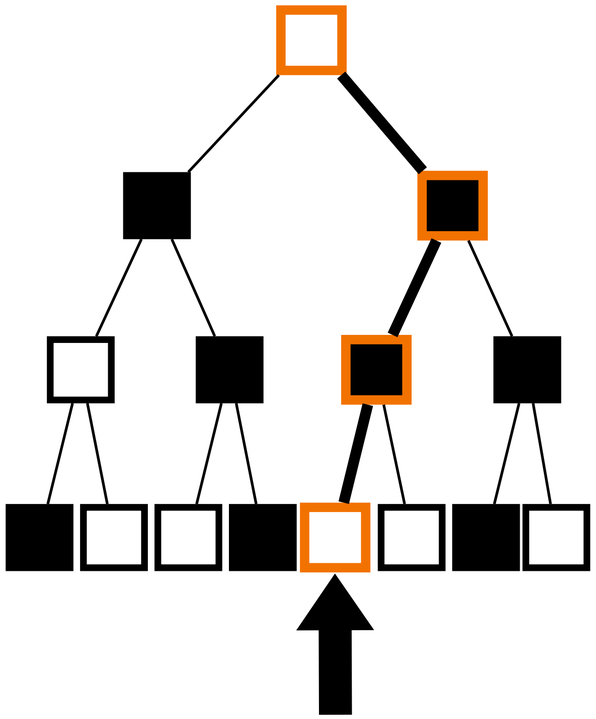
Key includes *correction words*

$\lambda + 2$ bits





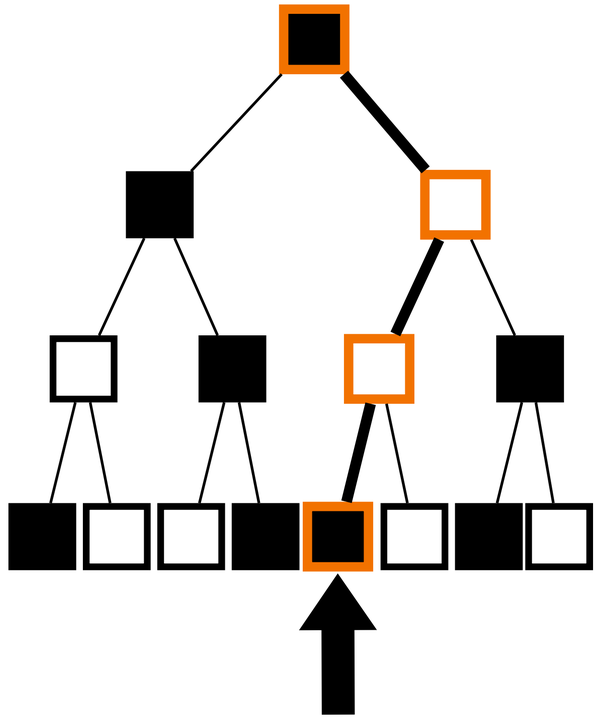


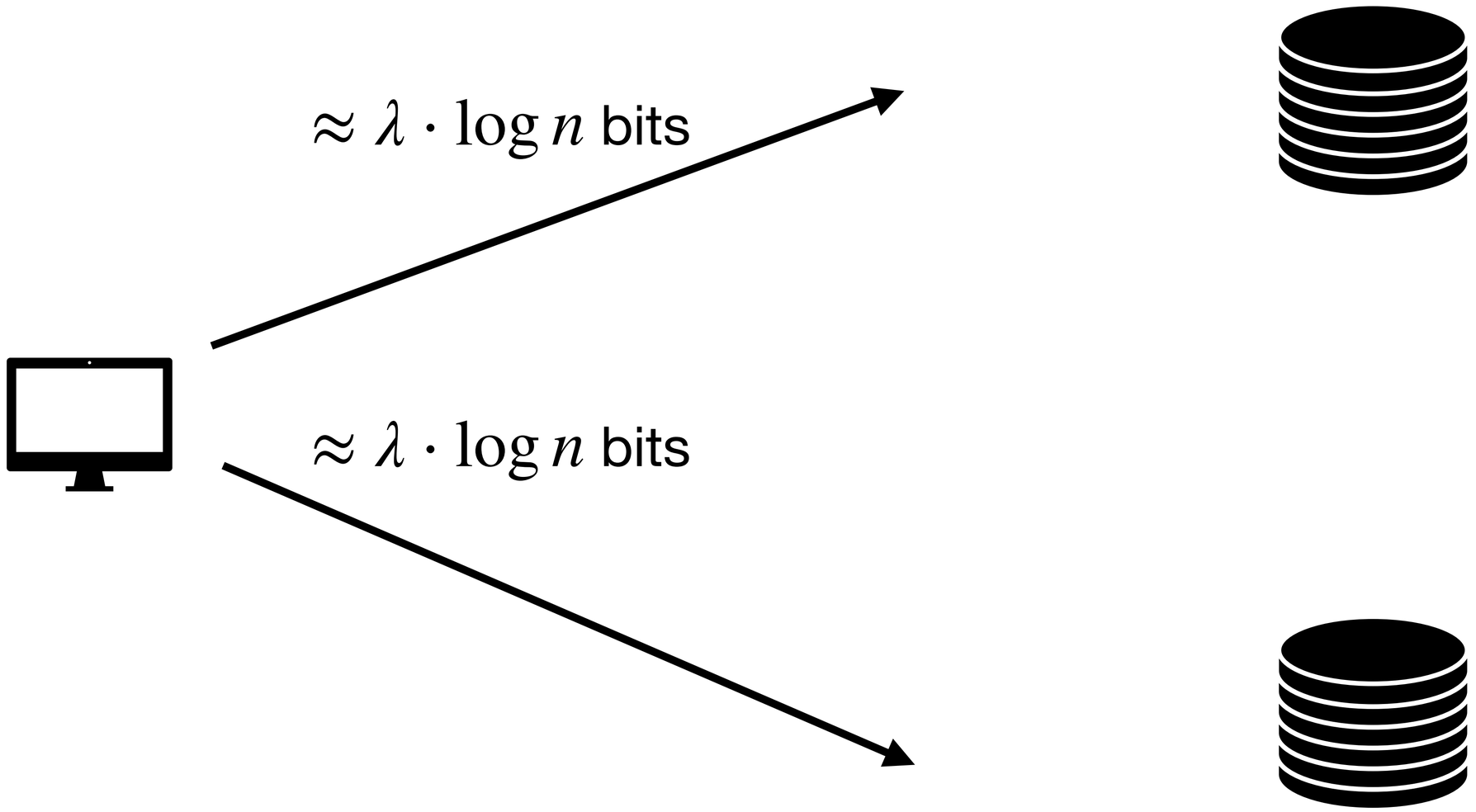


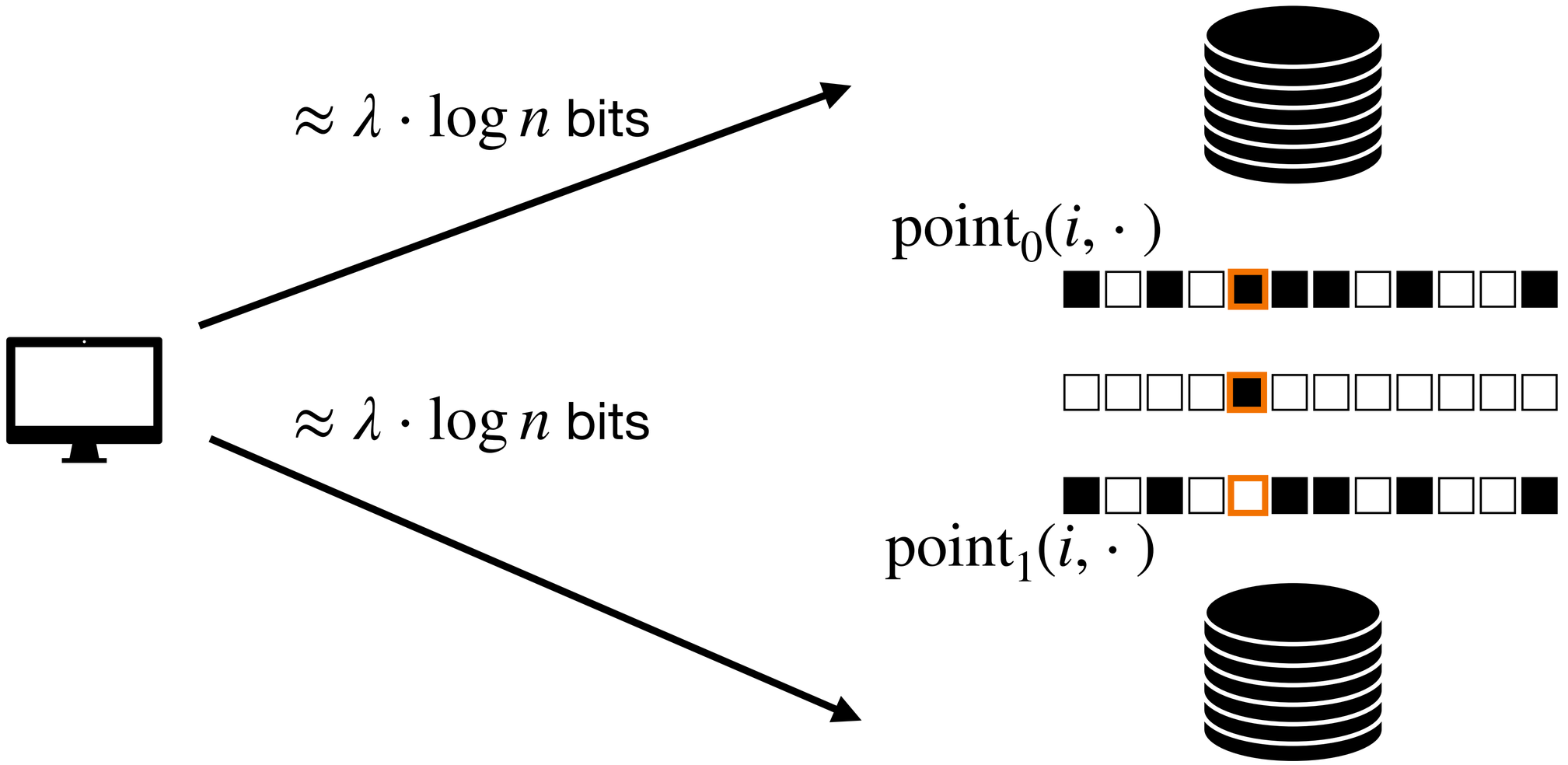
cw_0

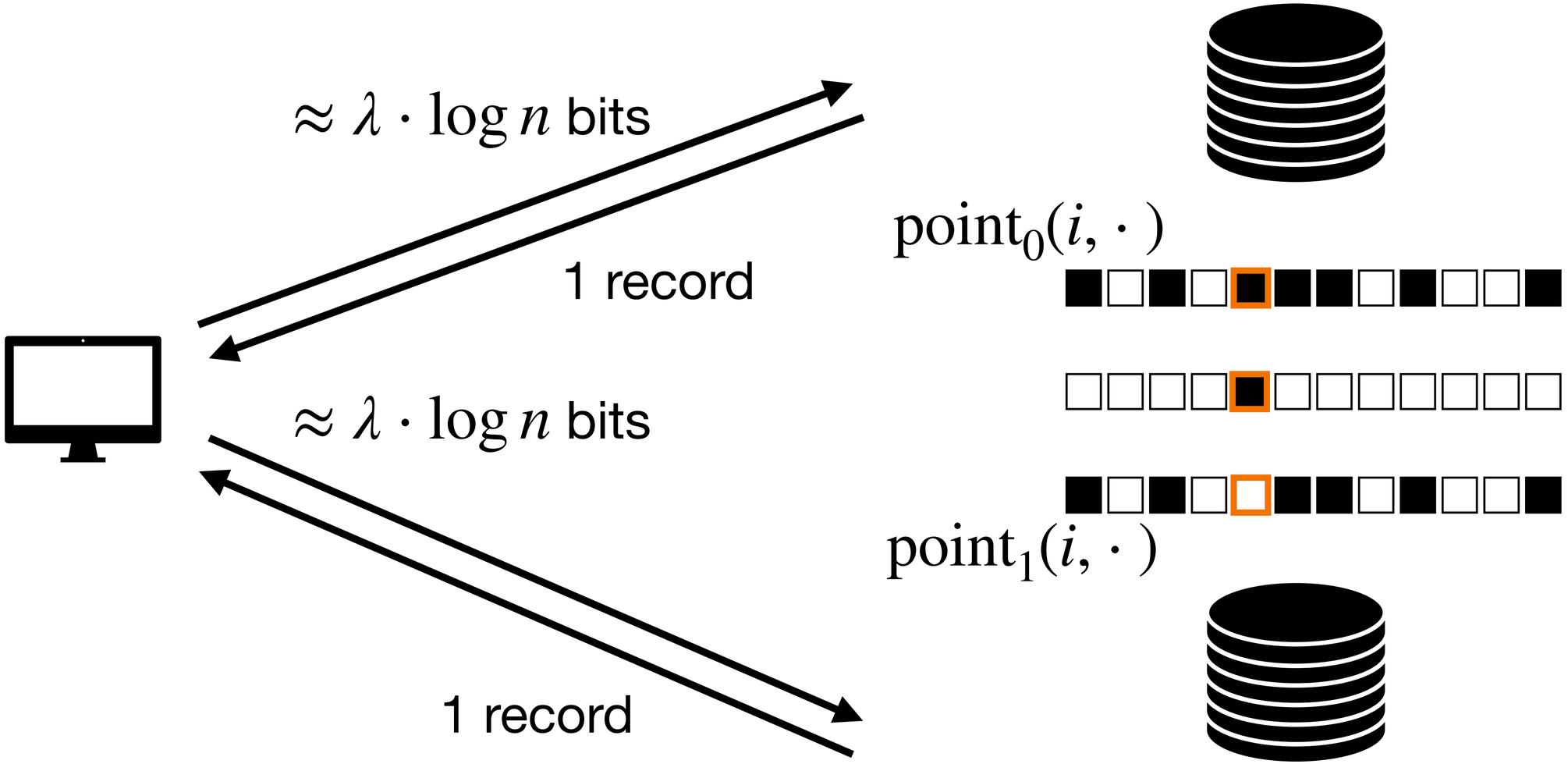
cw_1

cw_2




$\approx \lambda \cdot \log n$ bits
$\approx \lambda \cdot \log n$ bits


$\approx \lambda \cdot \log n$ bits
$\approx \lambda \cdot \log n$ bits
$\text{point}_0(i, \cdot)$
$\text{point}_1(i, \cdot)$


$\approx \lambda \cdot \log n$ bits
1 record
$\approx \lambda \cdot \log n$ bits
1 record
$\text{point}_0(i, \cdot)$
$\text{point}_1(i, \cdot)$

Today's objectives

Define 2-server private information retrieval

Introduce notion of function secret sharing

Construct distributed point function

Show improved 2-server PIR protocol