Securely Sampling Biased Coins with Applications to Differential Privacy

Jeffrey Champion, abhi shelat, Jonathan Ullman

Northeastern University

This talk

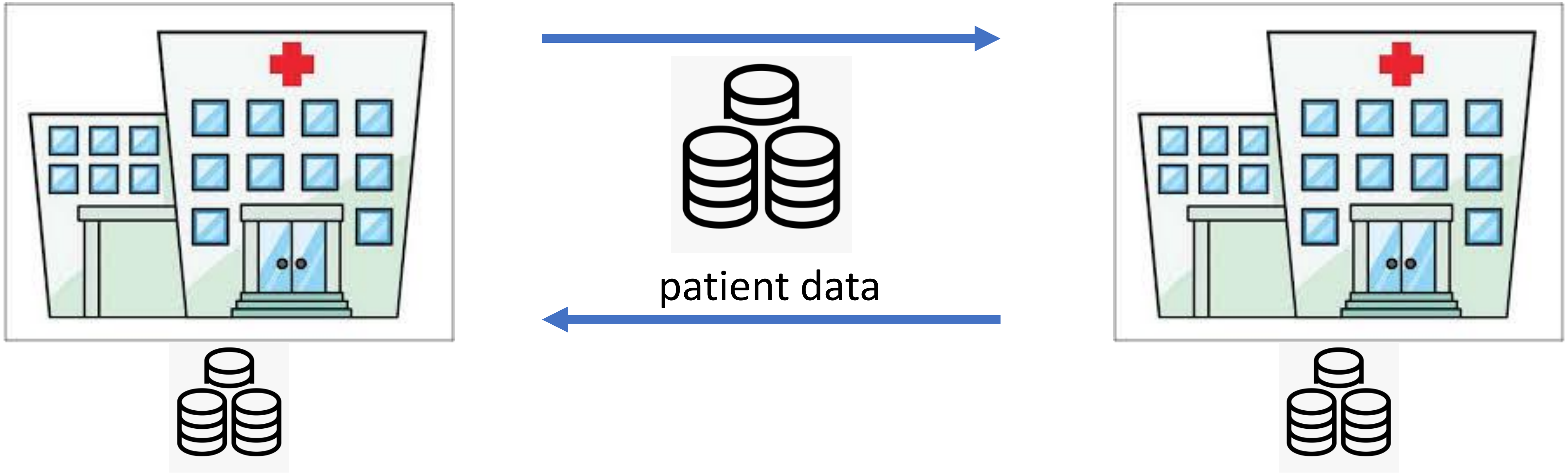
Asymptotically and
concretely more efficient

Improved protocols for securely generating noise from common distributions

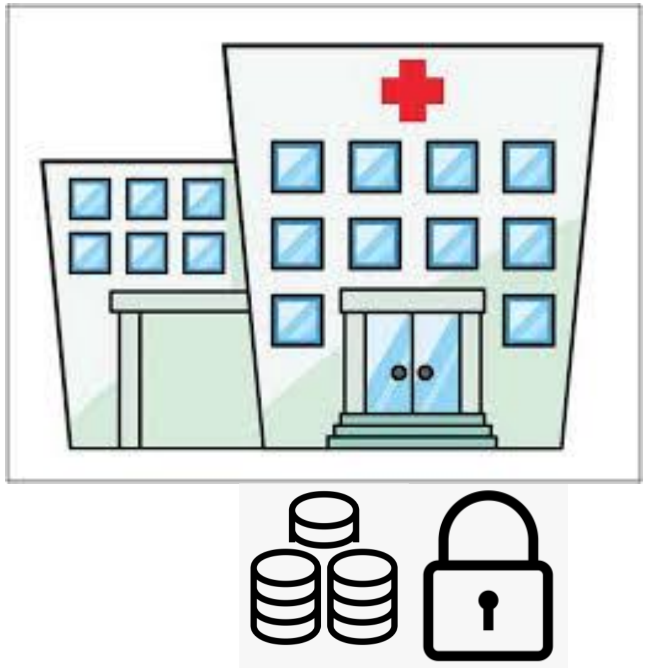
arising in differential
privacy [DMNS’06]

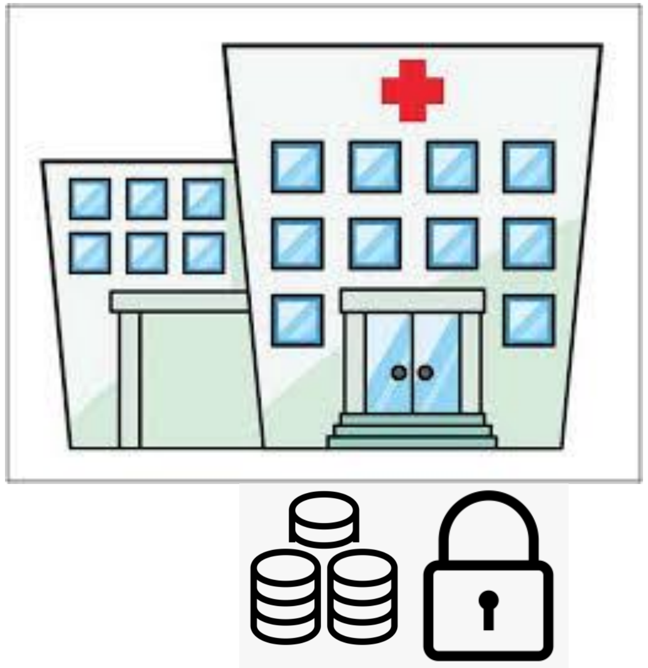
eg.
geometric,
binomial,
poisson

Suppose two or more hospitals want to jointly compute statistics of their patients data

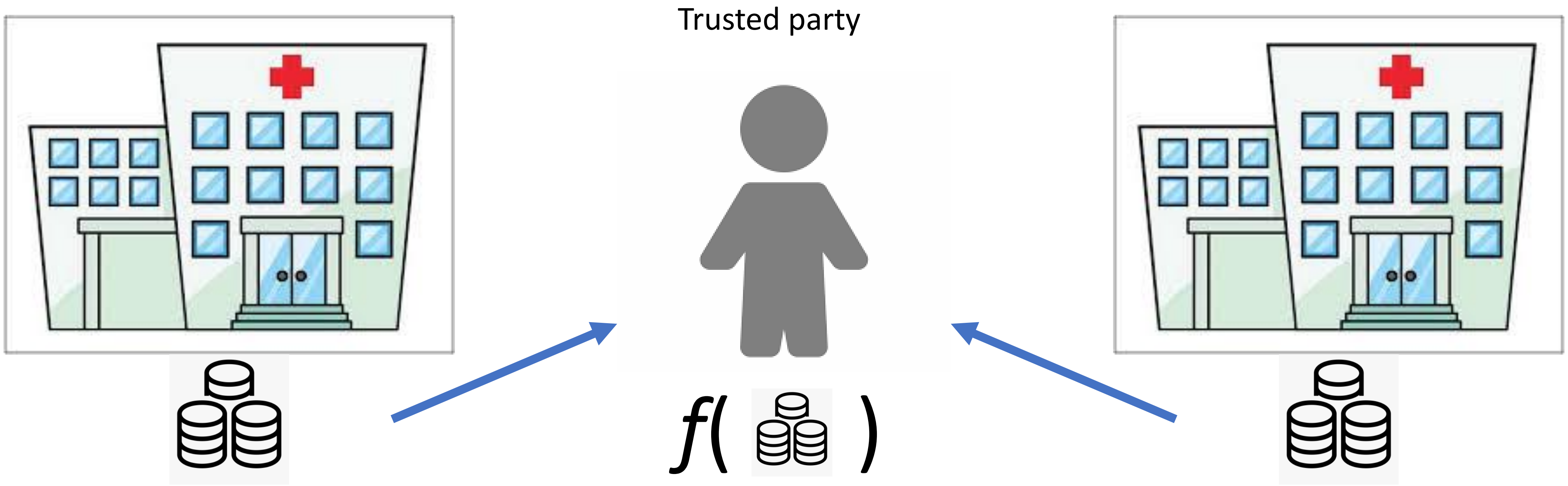


However, sharing data may be prohibited





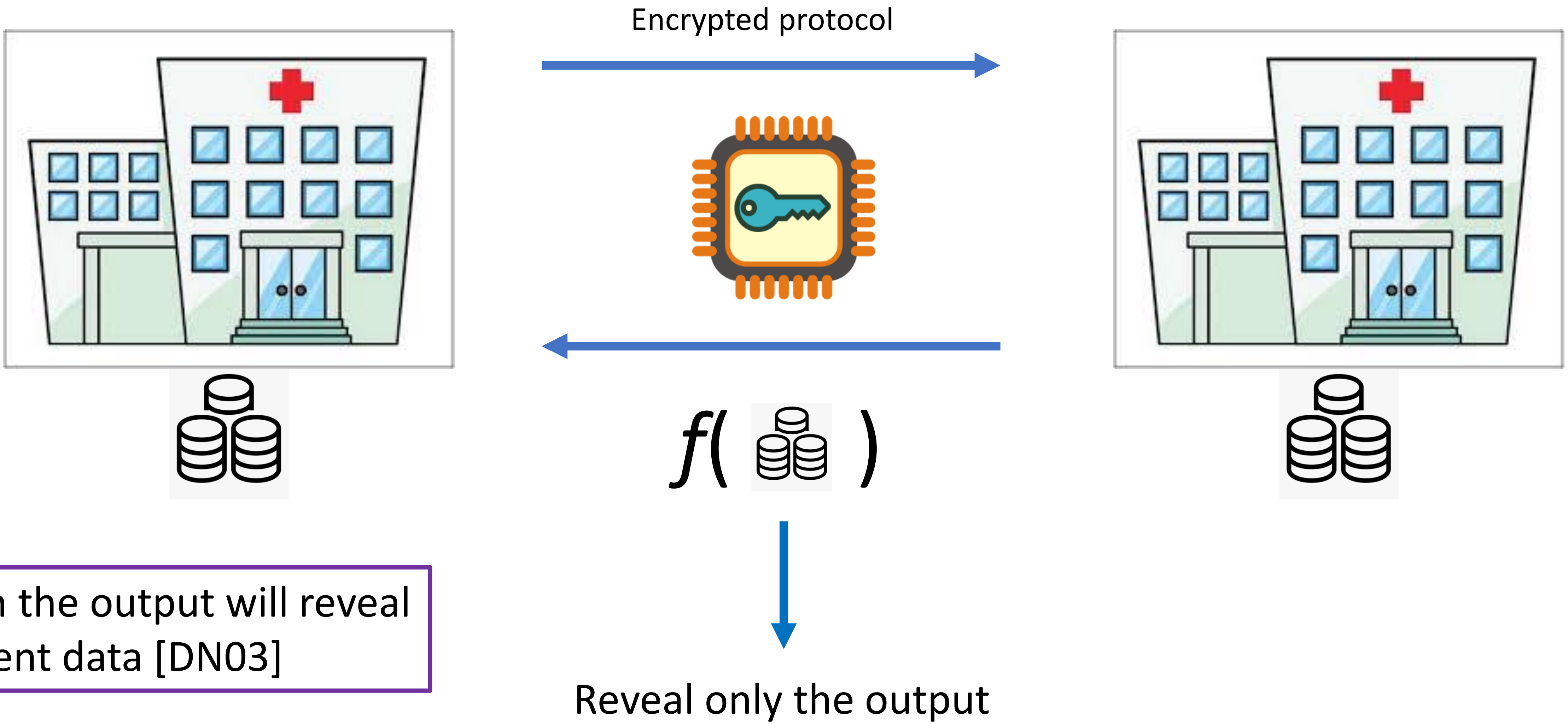
An ideal solution: find a trusted third party



A trusted party is usually not available

Release only the output of the study

Secure Computation



Even the output will reveal patient data [DN03]

Differential Privacy [DMNS06]

Strong guarantee of privacy for the patients.
Widely deployed in practice (e.g. Google, Apple, Uber, US Census Bureau)

Whole dataset

Randomness

$f($  ,  $)$

Inject carefully calibrated random noise during the computation of f

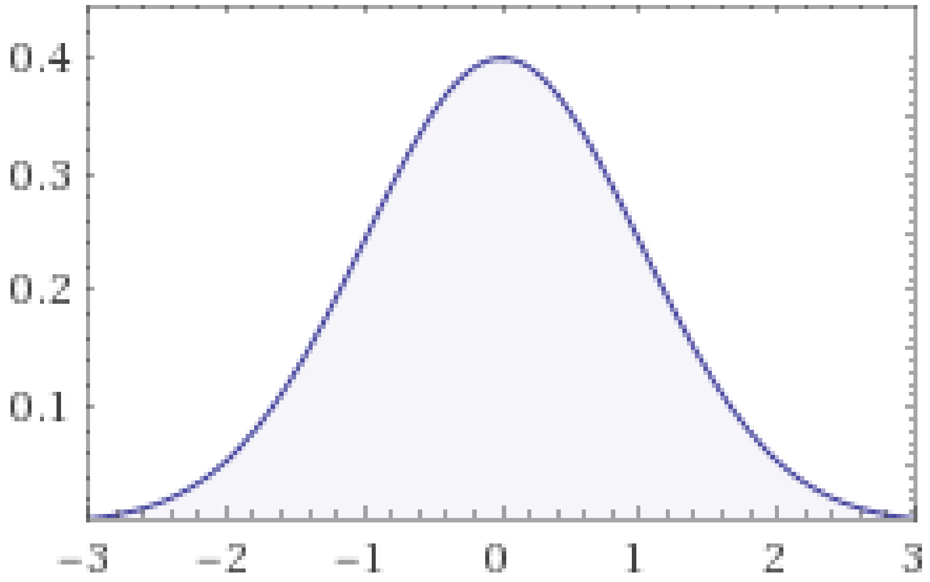
Private output

Have to design secure protocols for the modified algorithm

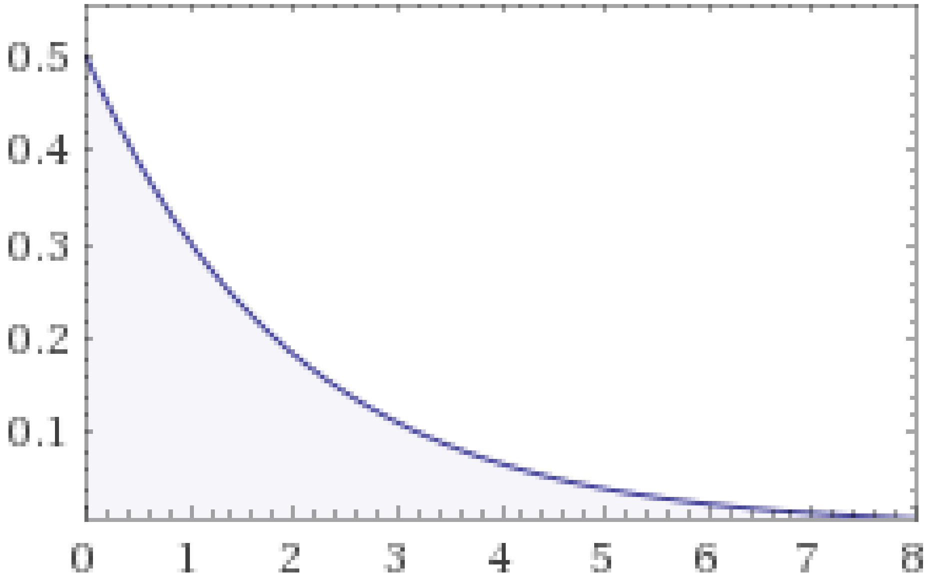
Bottleneck: generating random noise

Typical DP algorithms use noise from Gaussian, Laplace, or Exponential distributions:

Gaussian Noise



Exponential Noise

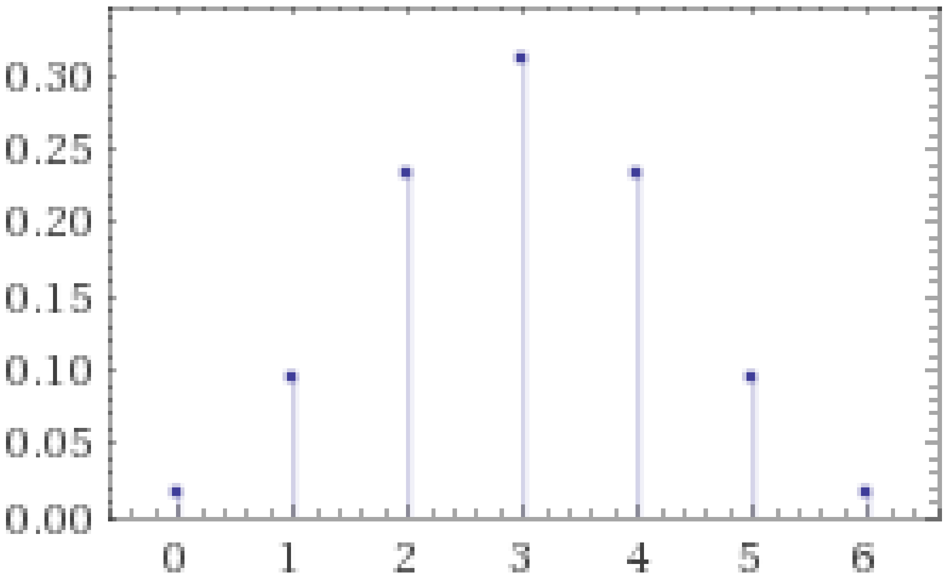


Approximating using floating points can destroy privacy [Mironov'12]

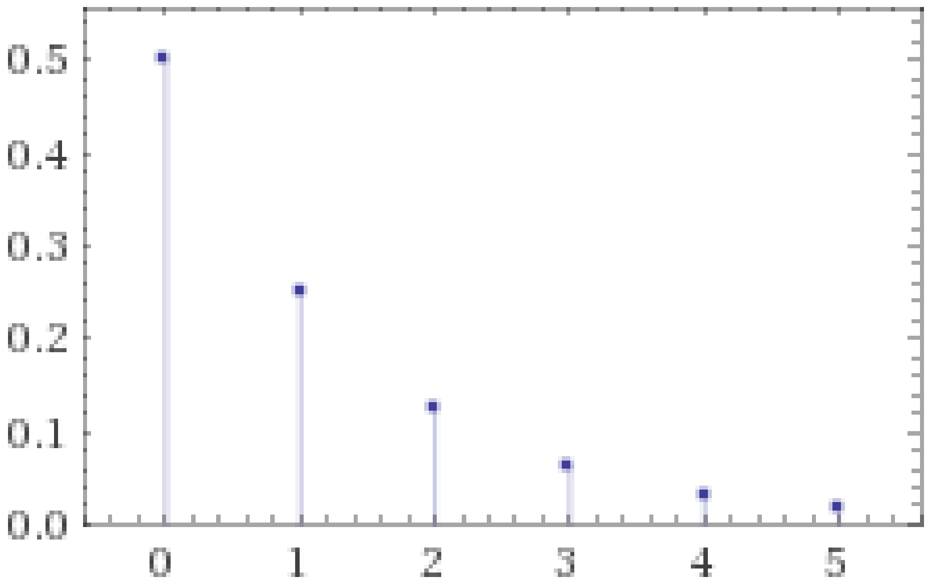
Bottleneck: generating random noise

Discrete distributions are more amenable to secure computation:

Binomial Noise



Geometric Noise



Still need to sample with high precision to ensure privacy

Prior work

- Inspired by [DKMMN'06]
  - Proposed combining differential privacy and secure computation
  - Identified the problem of noise generation
  - Gave protocols for sampling noise with various tradeoffs between resources
- [EKMPP'14] implemented floating point arithmetic in secure computation in order to compute Laplace noise
- [AC'15] implemented Laplace noise sampling with two parties using a cut-and-choose protocol (polynomial security)

Our Work – Theory

Reduce the complexity for sampling common noise distributions securely

- Improved amortized complexity for sampling a biased coin from $O(\lambda)$ to $O(\log \lambda)$
- Novel use of oblivious stacks [ZE'13]

Application to widely used differentially private algorithms (e.g. report-noisy-max/exponential mechanism [MT'07])

Our Work – Empirical

Full open source implementation in Obliv-C [ZE'15]

- Includes both our protocol and [DKMMN'06]

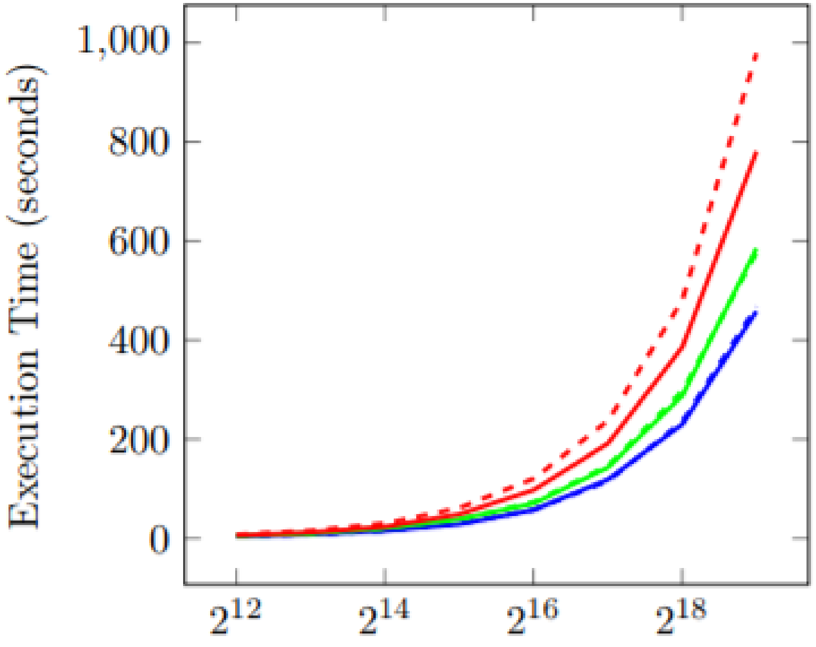
Experimental evaluation

- Consider a practical variant of our protocol (slightly worse asymptotic complexity)
- Improved cost, runtime, and communication for generating noise in specific differential privacy applications

Practical improvement

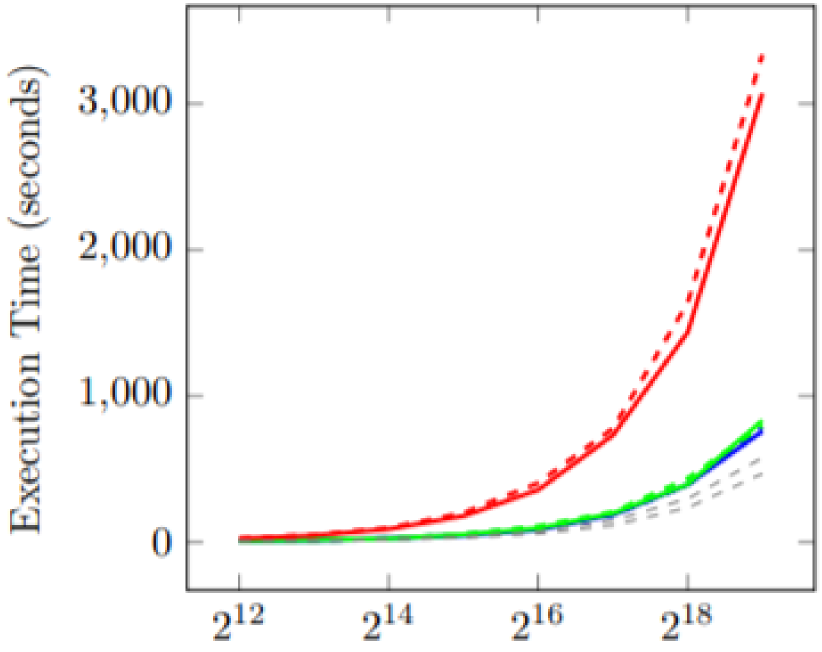
Experiment: generating *d* samples of geometric noise in 2PC with our method and the trivial method

east-east



Number of samples

east-west



Number of samples

Key
[DKMMN'06]
our protocol

[EKMPP'14] generate one Laplace sample in 15s
[AC'15] generate one Laplace sample in 9s

Generating Noise Insecurely

Mean $1/p$

Steps to sample geometric noise with parameter $0 < p < 1$

1. Sample a uniform real number: $0 < u < 1$
2. Compute the inverse CDF:

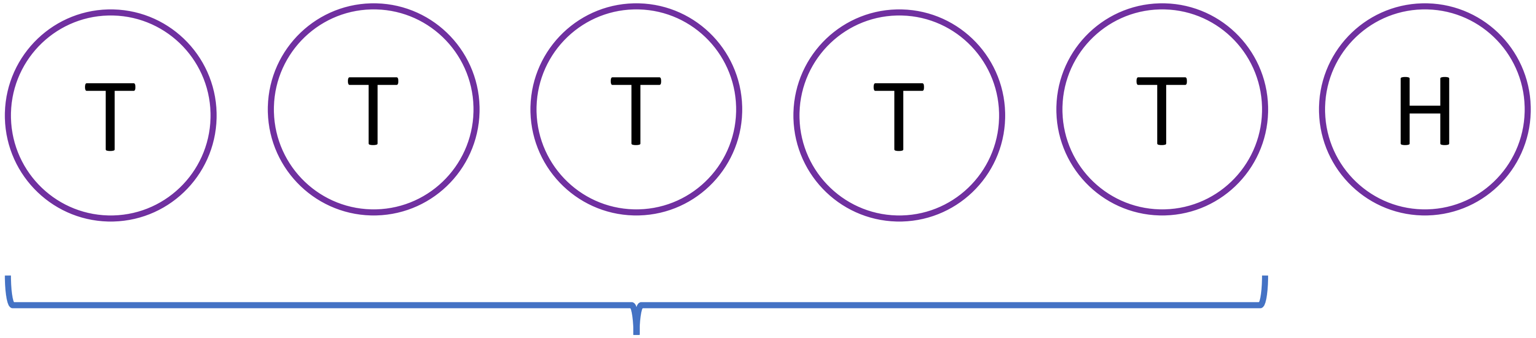
$$F^{-1}(u) = \left\lfloor \frac{\ln(1-u)}{\ln(1-p)} \right\rfloor$$

Computing logarithms is costly in MPC. Using finite arithmetic has hard-to-understand effects.

Generating Noise Insecurely: Biased Coins

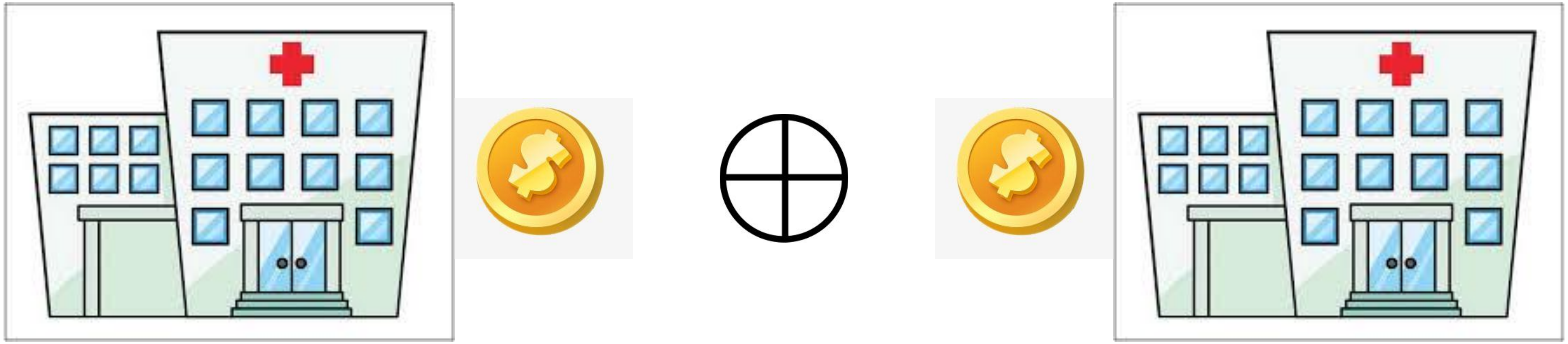
<u>Steps to sample geometric noise with parameter $0 < p < 1$</u>

1. Find a coin with bias p ($P[\text{heads}] = p$)
2. Flip the coin until it comes up heads
3. Count the number of tails before the first heads



Only simple, discrete operations. Using finite precision has predictable effects.

Securely sampling fair coins



Sampling fair coins in a secure computation is easy

How can we convert fair coins to biased coins in a secure computation?

Insecure Biased Coins: Lazy Comparison

Stream of random bits

r_1	r_2	r_3	r_4	r_5	r_6	…

$\overset{?}{<}$

Binary expansion of bias 0<p<1

b_1	b_2	b_3	b_4	b_5	b_6	…

Insecure Biased Coins: Lazy Comparison

Stream of random bits

0	1	1	1	0	0	…

rand $\oplus$ bias

Binary expansion of bias (1/3)

0	1	0	1	0	1	…

Insecure Biased Coins: Lazy Comparison

If (rand ≠ bias):
output bias

Stream of random bits

0	1	1	1	0	0	...

rand
0

$\oplus$

bias
0

Binary expansion of bias (1/3)

0	1	0	1	0	1	...

No output

Insecure Biased Coins: Lazy Comparison

If (rand ≠ bias):
output bias

Stream of random bits

0	1	1	1	0	0	...

rand

1

$\oplus$

bias

1

Binary expansion of bias (1/3)

0	1	0	1	0	1	...

No output

Insecure Biased Coins: Lazy Comparison

If (rand ≠ bias):
output bias

Stream of random bits

0	1	1	1	0	0	...

rand

1

$\oplus$

bias

0

Binary expansion of bias (1/3)

0	1	0	1	0	1	...

Output: 0

E[# of comparisons] = 2 per coin
O(1) time per coin

Securely Generating Biased Coins?

If (rand ≠ bias):
output bias

Stream of random bits

0	1	1	1	0	0	…

rand

$\oplus$

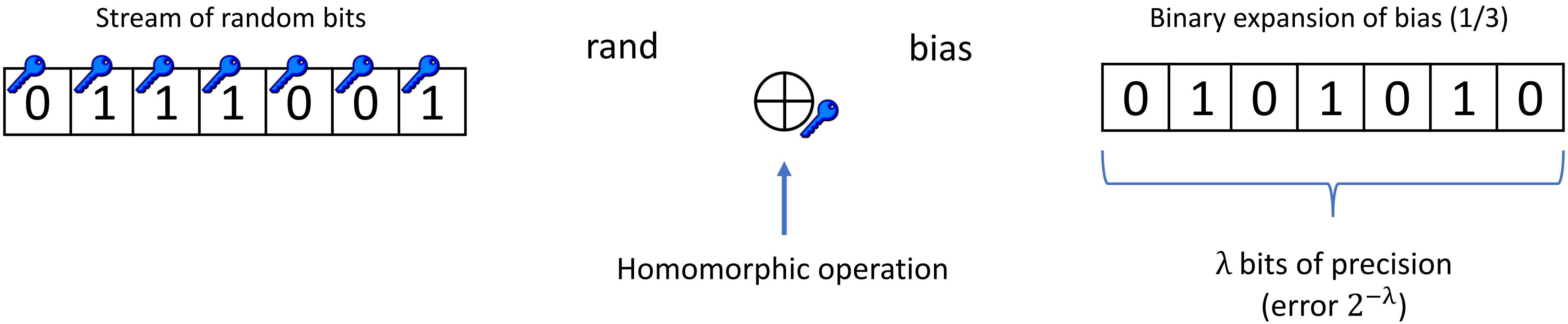
bias

Binary expansion of bias (1/3)

0	1	0	1	0	1	…

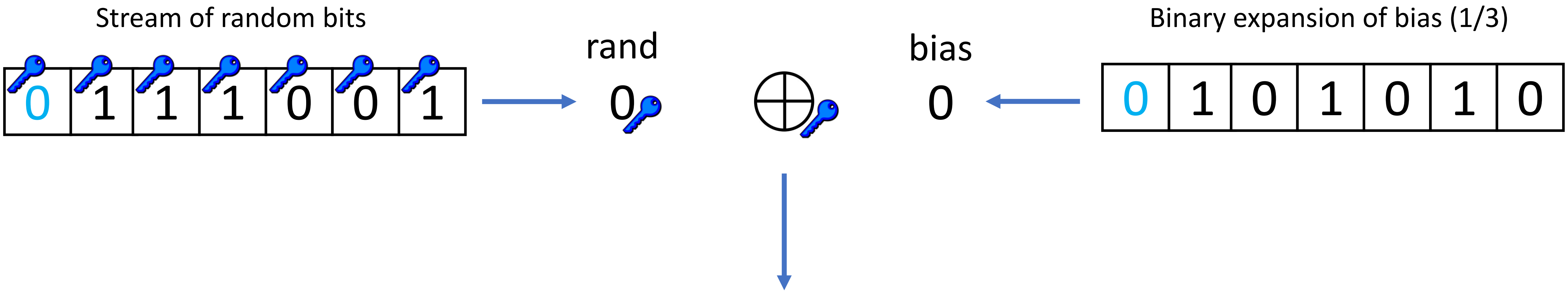
Securely Generating Biased Coins?

If (rand ≠ bias):
output bias



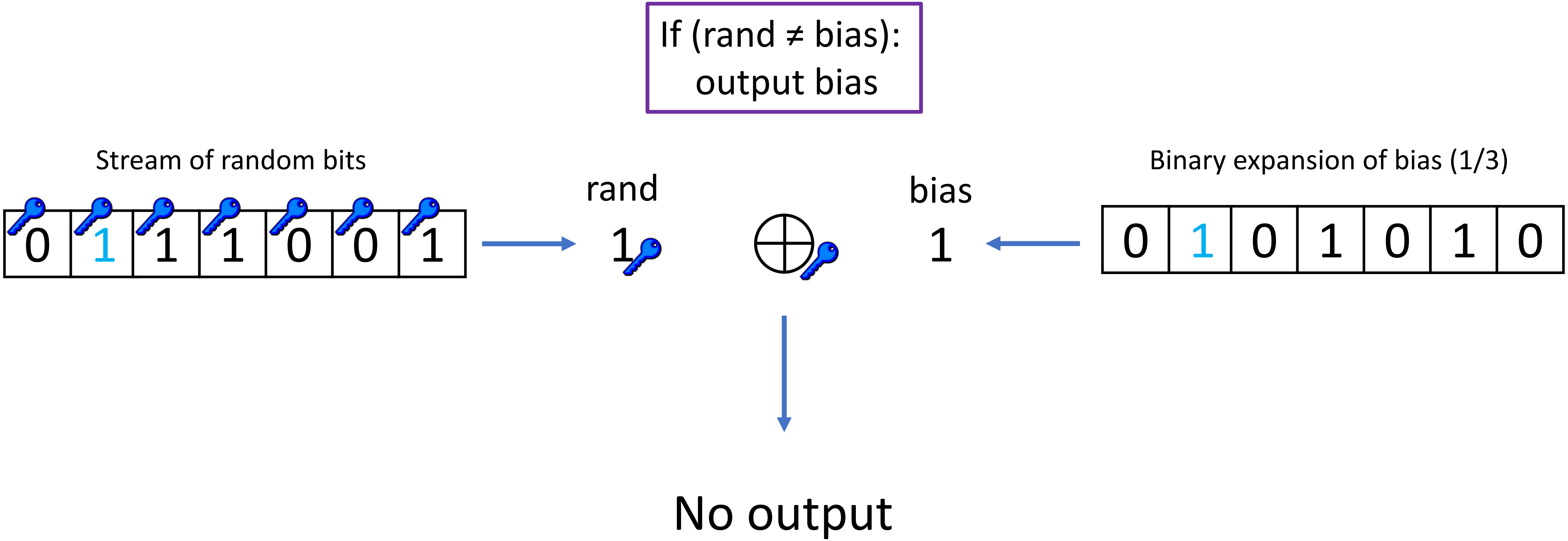
Securely Generating Biased Coins?

If (rand ≠ bias):
output bias



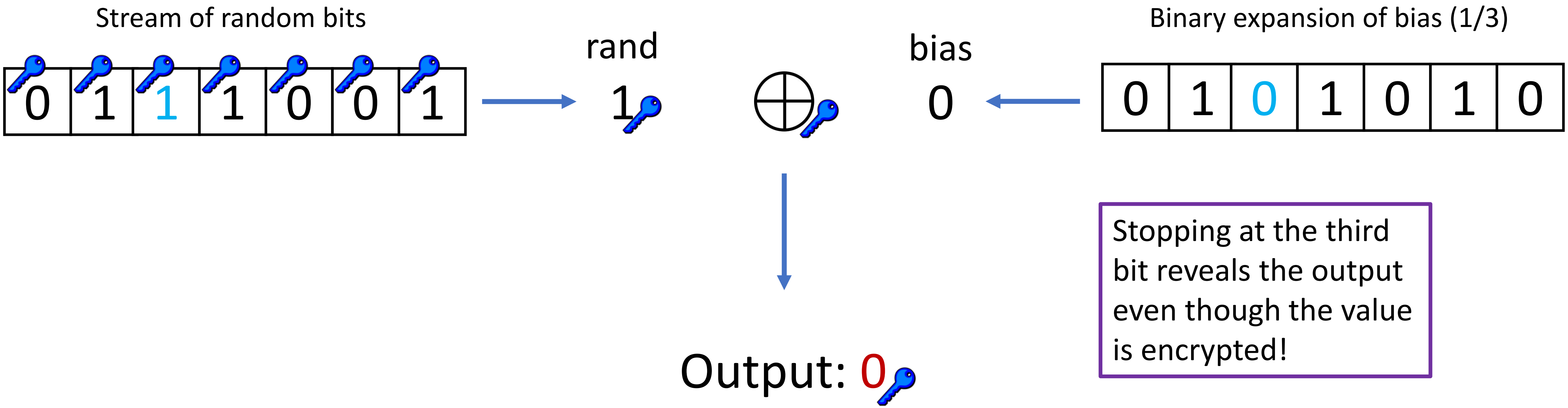
No output

Securely Generating Biased Coins?



Securely Generating Biased Coins?

If (rand ≠ bias):
output bias



Securely Generating Biased Coins

Output rand $\overset{?}{<}$ bias

Stream of random bits

0 1 1 1 0 0 1

$\oplus$

Binary expansion of bias (1/3)

0 1 0 1 0 1 0

Output: 0

Naïve solution: do the full comparison
O(λ) time per coin!

Stopping at the third bit reveals the output even though the value is encrypted!

Our Work – Theory

Reduce the complexity for sampling common noise distributions securely

- Improved amortized complexity for sampling a biased coin from $O(\lambda)$ to $O(\log \lambda)$
- Novel use of oblivious stacks [ZE'13]

Application to widely used differentially private algorithms (e.g. report-noisy-max/exponential mechanism [MT'07])

Our Approach – secure lazy sampling

Stream of random bits

1	1	1	1	0	0	1

rand $\oplus$ bias

Binary expansion of bias

0	1	0	1	0	1	0

Our Approach – secure lazy sampling

Stream of random bits

1	1	1	1	0	0	1

rand

1

$\oplus$

bias

0

Binary expansion of bias

0	1	0	1	0	1	0

Need to produce correct bit of bias and hide when we restart bias

1. Get current bit of bias and compare to rand
2. If bits are not equal:
   a. Make coin
   b. Reset bias

Output: 0

Want to hide whether a biased coin was output

Oblivious data structures

- Best known example: ORAM [GO’97]
  - Major recent progress, but not suited for single bit data blocks
- Oblivious stacks [ZE13] are a more efficient alternative
  - We modify the construction from [ZE13] to suit our application

Role:
Conditionally make coin

Push-only stack

Current stack:

		f

Conditional push

Given input element e and condition c:

If $c = 1$:

	e	f

If $c = 0$:

		f

Role:
Get current bit of bias
Conditionally reset bias

Pop-only stack

Current stack: $\boxed{}\boxed{b}\boxed{c}$

Initial stack: $\boxed{a}\boxed{b}\boxed{c}$

Pop

Given reset bit r:

If $r = 1$: $a \leftarrow \boxed{}\boxed{b}\boxed{c}$ (reset to initial and pop)

If $r = 0$: $b \leftarrow \boxed{}\boxed{}\boxed{c}$ (pop from current stack)

Pop-only stack

Role:
Get current bit of bias
Conditionally reset bias

Conditional reset

Given reset bit r and condition c (stack is untouched):

If $c = 1$: set $r = 1$

If $c = 0$: nothing

Oblivious stack complexity

Conditional push, pop, and conditional reset can all be implemented such that the amortized complexity per operation is O(log n) for total capacity n

Oblivious Stacks in context

Stream of random bits

0	1	1	1	0	0	1

rand $\oplus$ bias

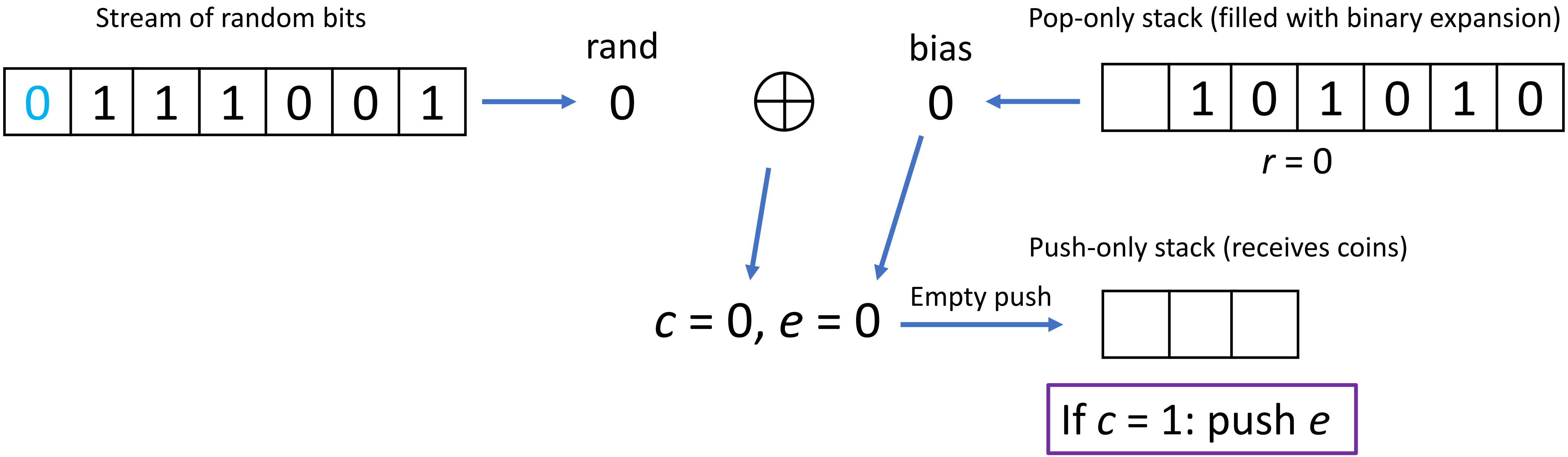
Pop-only stack (filled with binary expansion)

0	1	0	1	0	1	0

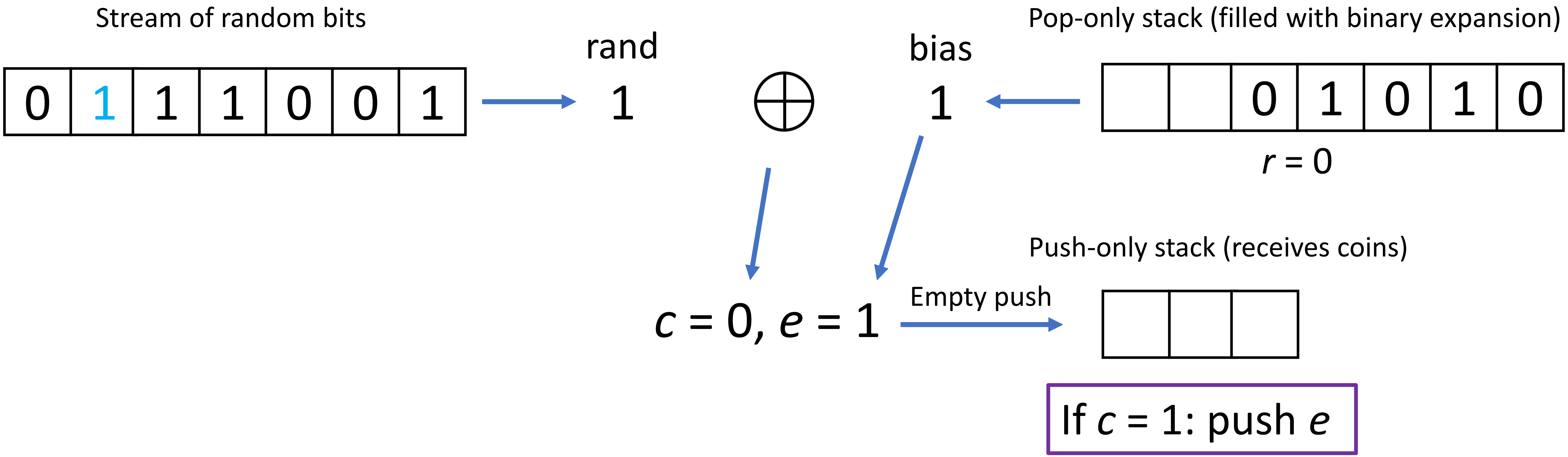
$r = 0$

Push-only stack (receives coins)

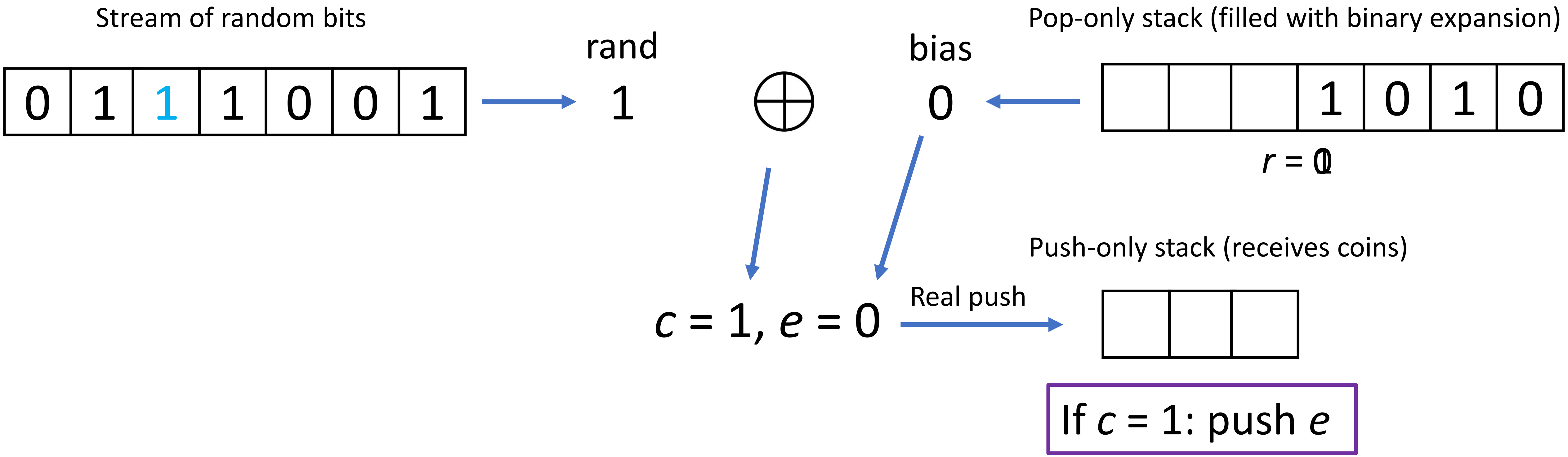
Oblivious Stacks in context



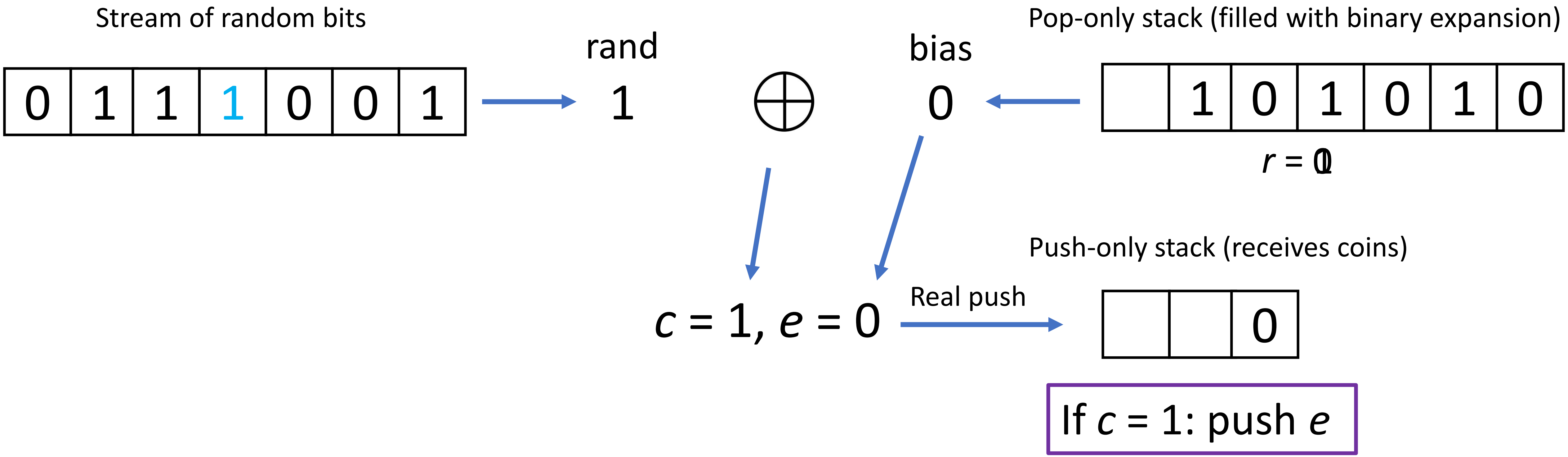
Oblivious Stacks in context



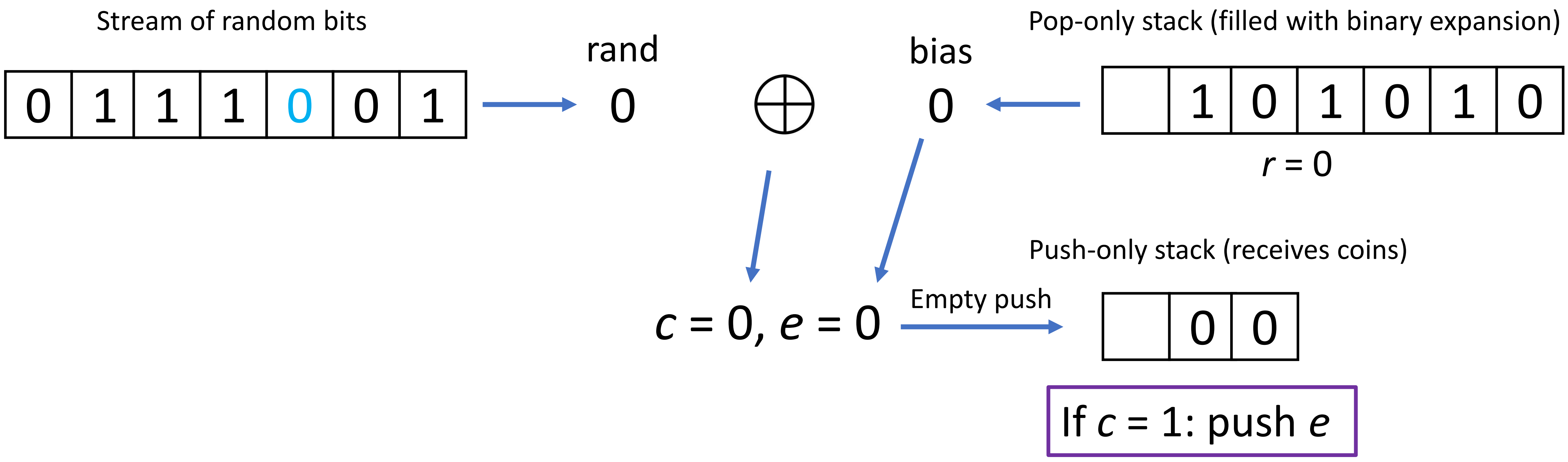
Oblivious Stacks in context



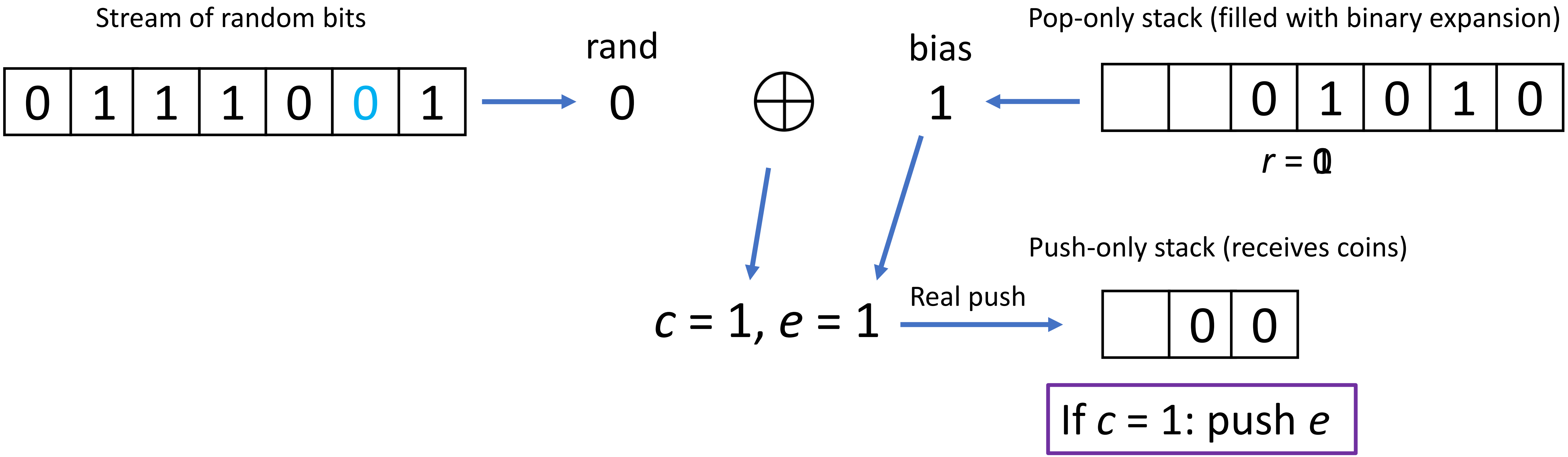
Oblivious Stacks in context



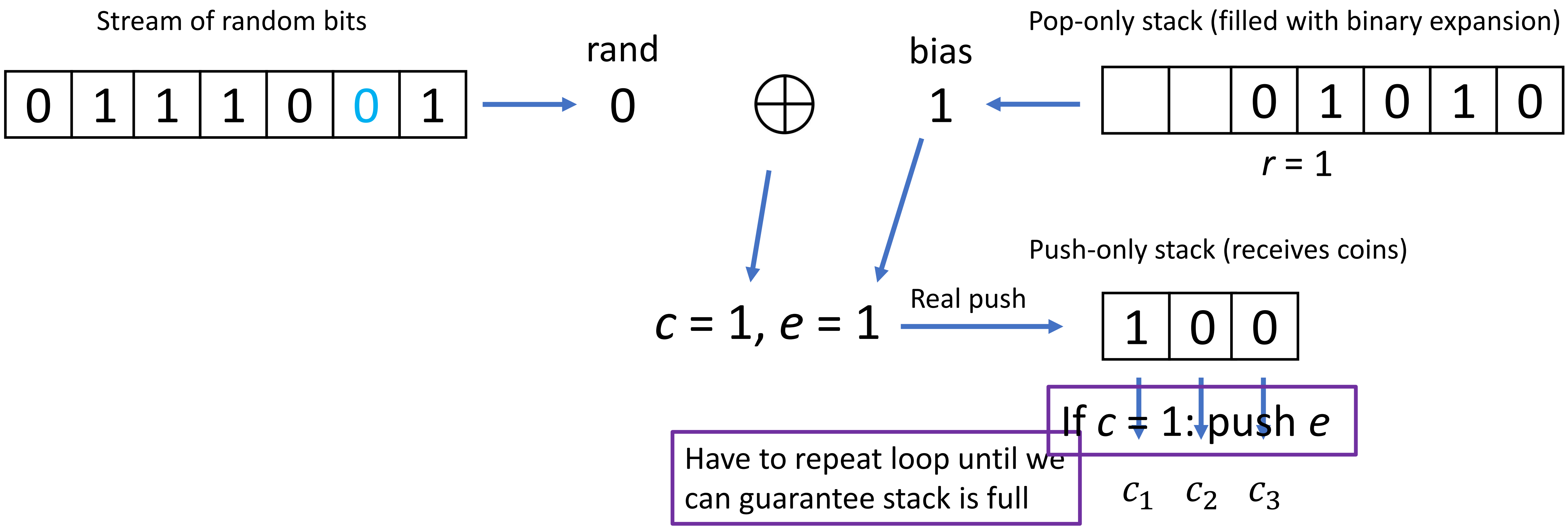
Oblivious Stacks in context



Oblivious Stacks in context



Oblivious Stacks in context



Summary

Our secure sampling protocol allows us to:

- Exponentially reduce the amortized cost of flipping a biased coin
- Sample hundreds of times faster than previous implementations
- Generate 500k samples from the geometric distribution in 7 min

We give the first complete, secure implementation of the exponential mechanism [MT07] for differential privacy

Thanks for listening!

Full paper: https://eprint.iacr.org/2019/823.pdf

Code: https://gitlab.com/neucrypt/securely_sampling