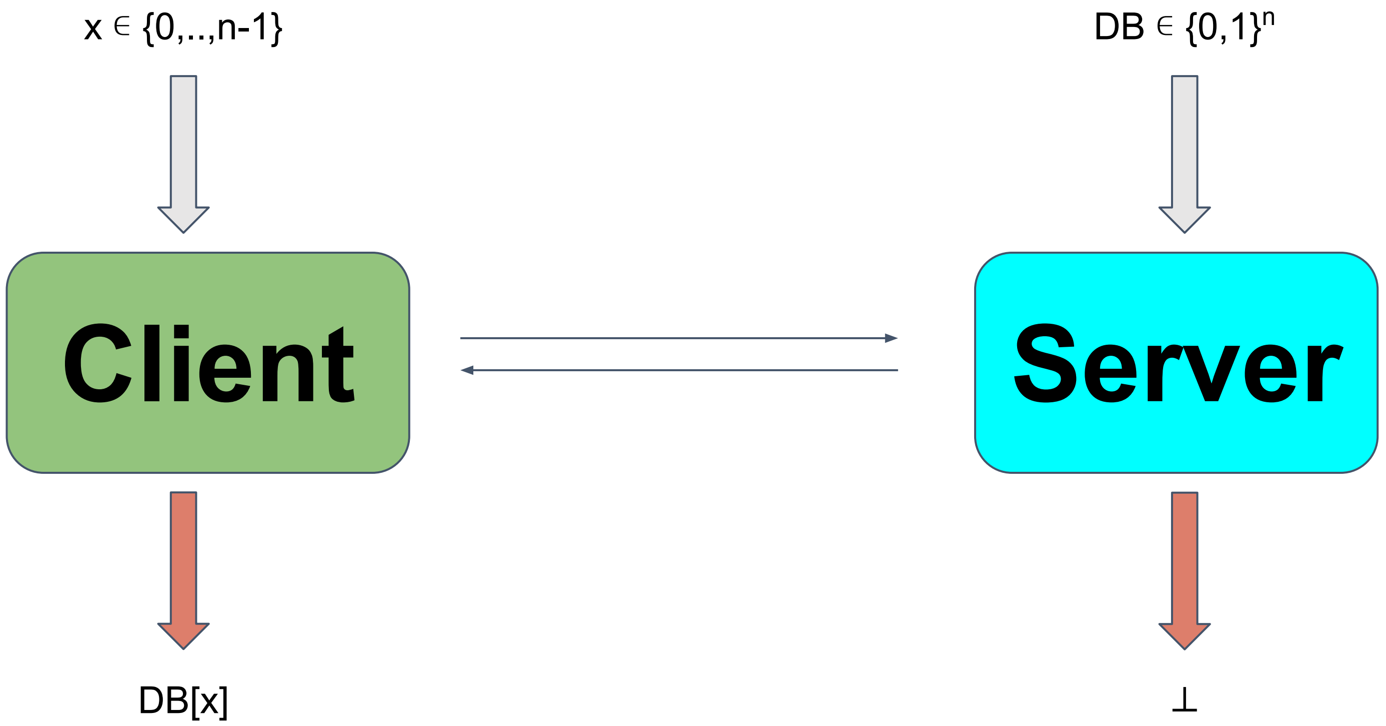
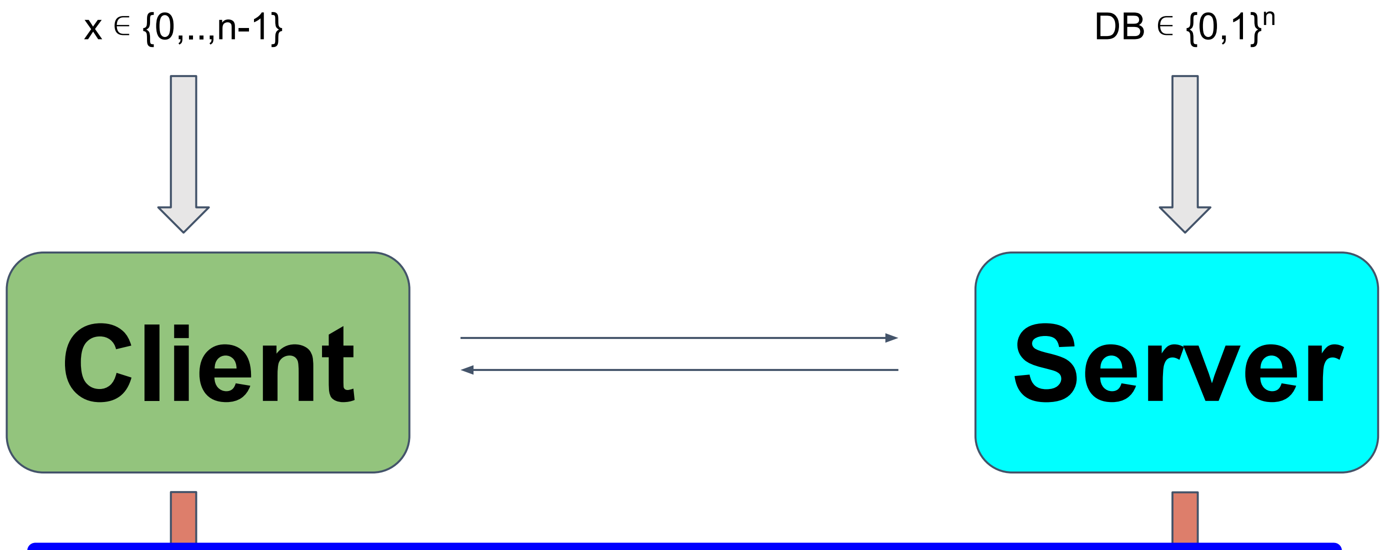
Near-Optimal Private Information Retrieval with Preprocessing

Arthur Lazzaretti and Charalampos Papamanthou

Private Information Retrieval [CGKM ‘95, KO ‘97,....]



Private Information Retrieval [CGKM ‘95, KO ‘97,....]



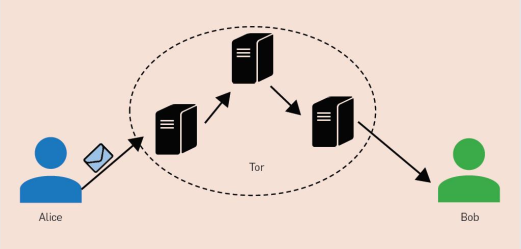
Privacy:
There exists a PPT simulator A* with no knowledge of x that can act as Client and Server cannot tell the difference

Applications

- Building block for different applications:

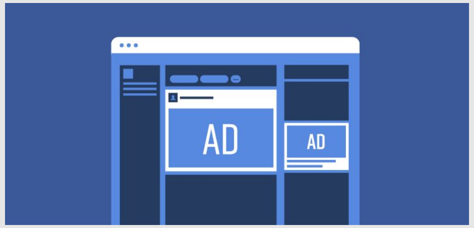
Applications

- Building block for different applications:



Meta-data hiding messaging
[Angel et al., OSDI ‘16]

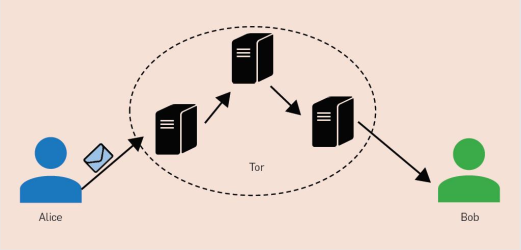
Private movie streaming
[Gupta et al., USENIX ‘16]



Private ad serving
[Zhong et al., USENIX ‘21]

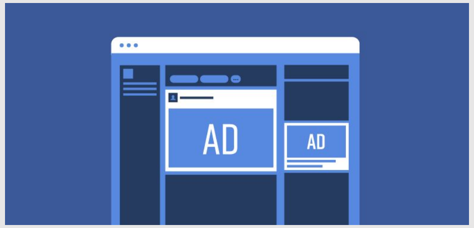
Applications

- Building block for different applications:



Meta-data hiding messaging
[Angel et al., OSDI ‘16]

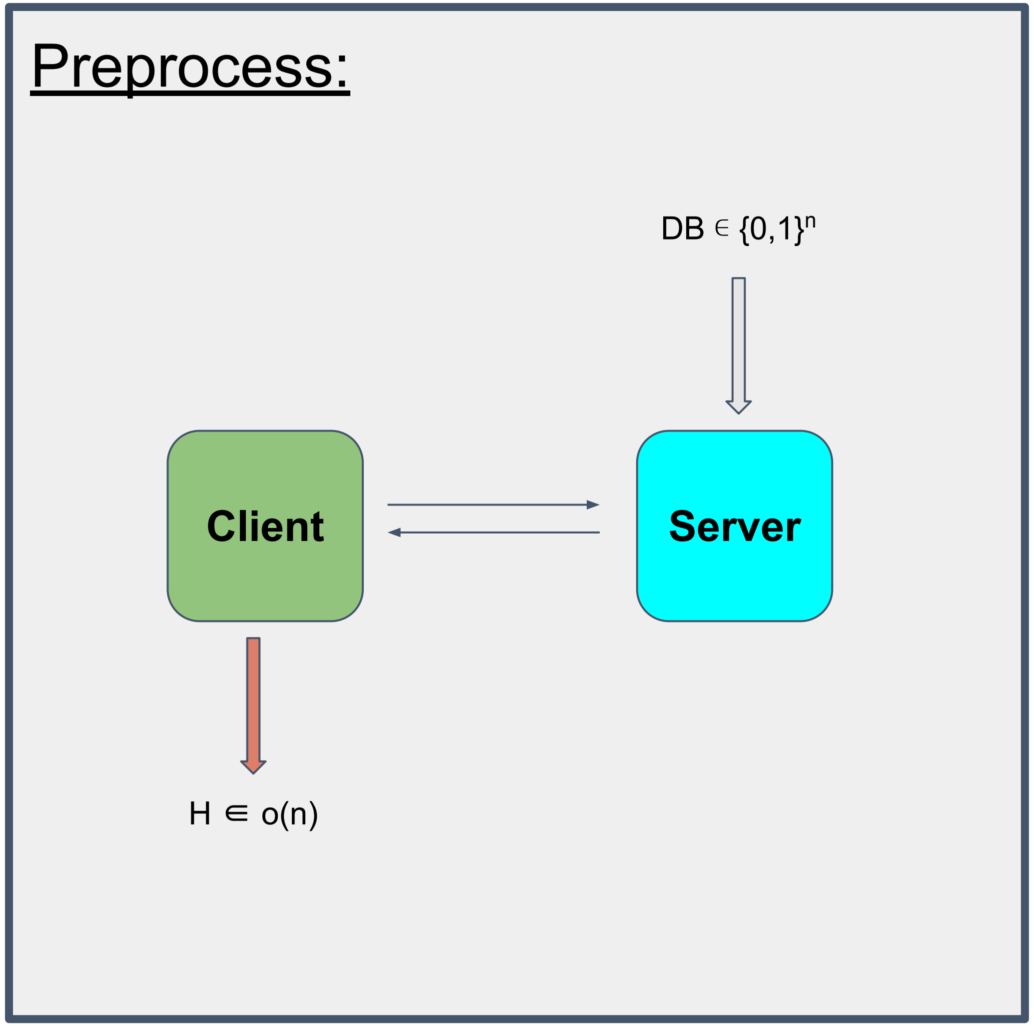
Private movie streaming
[Gupta et al., USENIX ‘16]



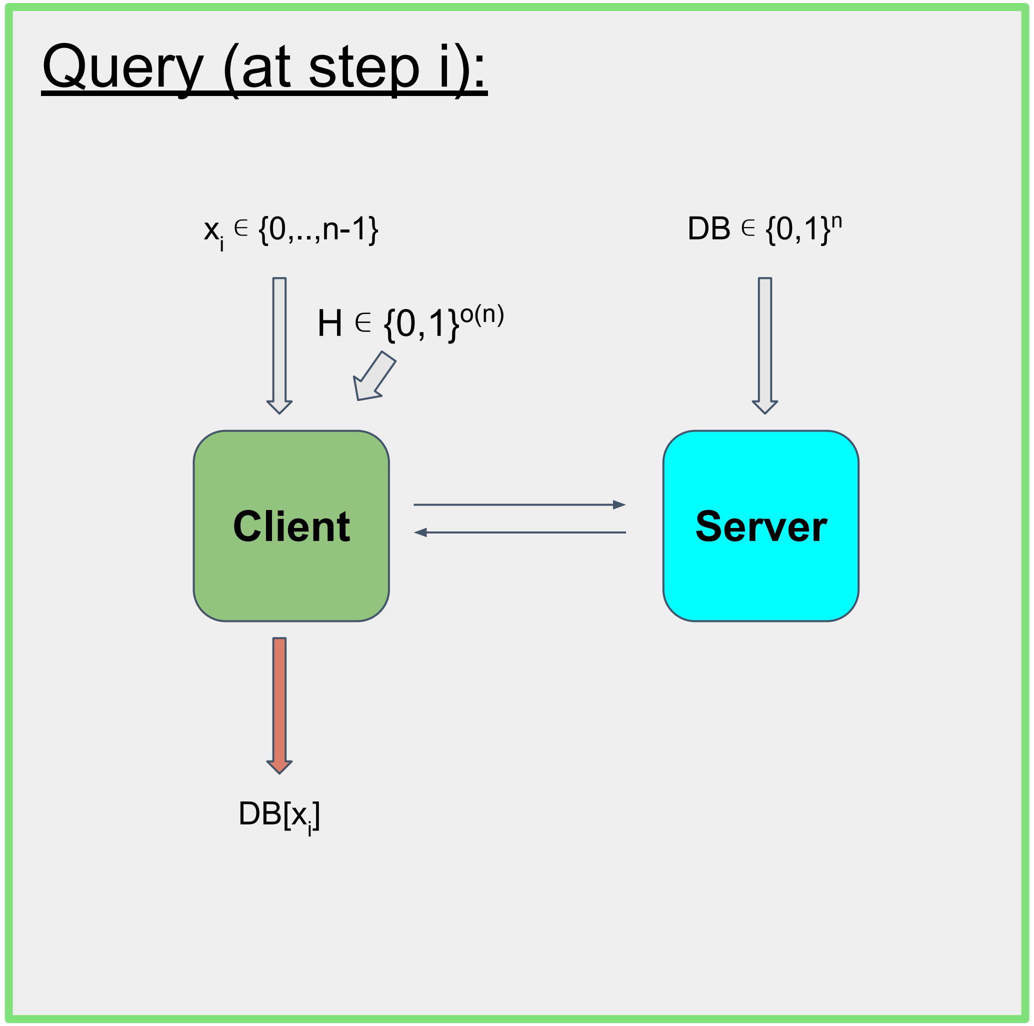
Private ad serving
[Zhong et al., USENIX ‘21]

Bottleneck: Server computation

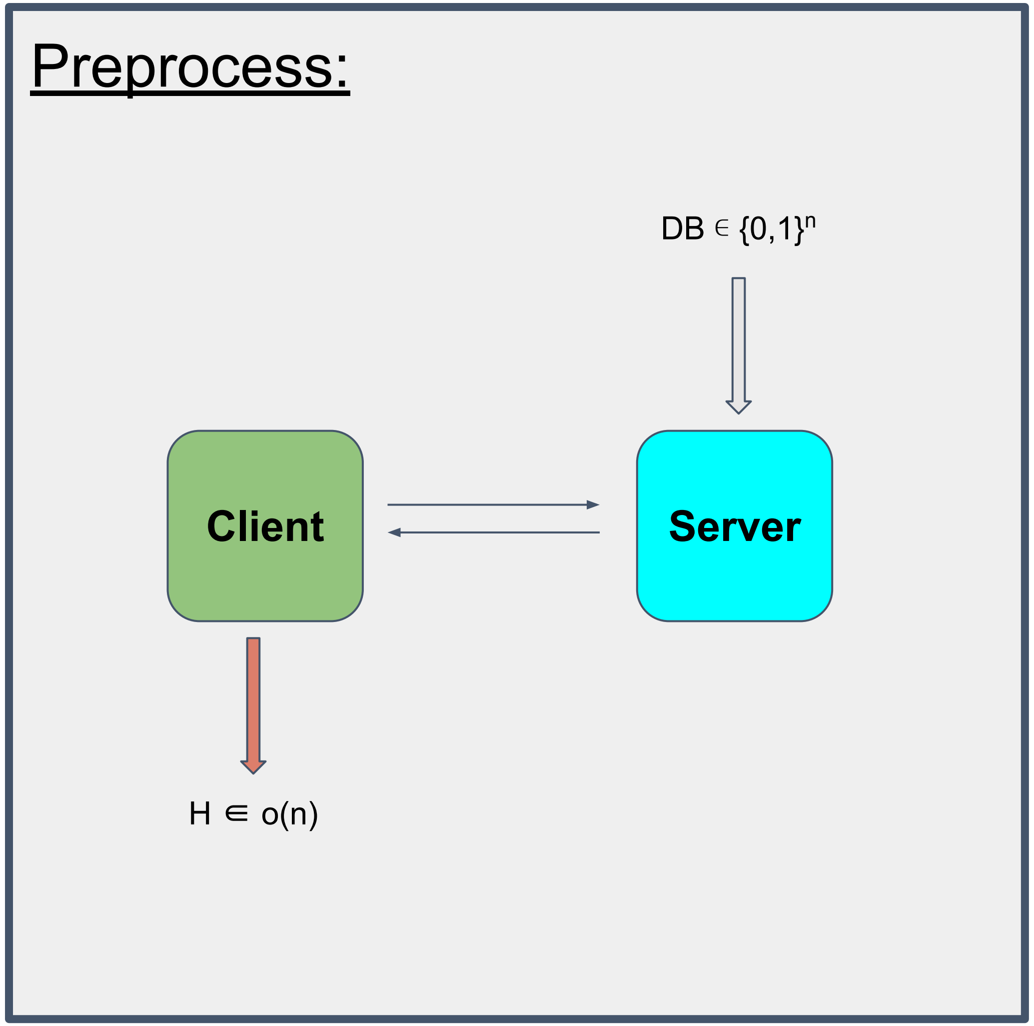
PIR with Client Preprocessing [BIM ‘04, …, CK ‘20, ...]

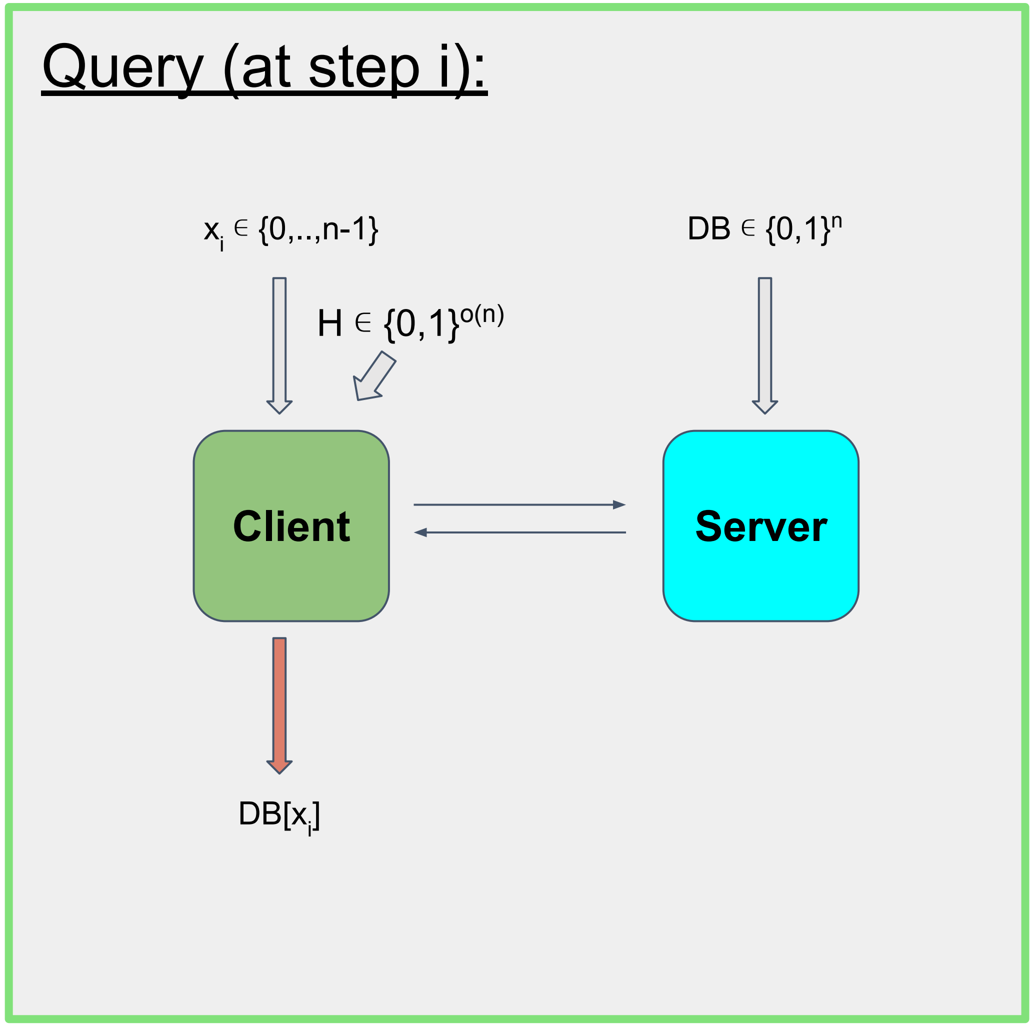


PIR with Client Preprocessing [BIM '04, …, CK '20, ...]



PIR with Client Preprocessing [BIM '04, …, CK '20, ...]





Our results

Scheme	Amortized Server Computation	Amortized Bandwidth	Client space	Number of servers
Ours	$\tilde{O}(\sqrt{n})$	$\tilde{O}(1)$	$\tilde{O}(\sqrt{n})$	1
[Shi et al., CRYPTO ‘21]	$\tilde{O}(\sqrt{n})$	$\tilde{O}(1)$	$\tilde{O}(\sqrt{n})$	2
[Corrigan Gibbs et al., EUROCRYPT ‘22]	$\tilde{O}(\sqrt{n})$	$\tilde{O}(\sqrt{n})$	$\tilde{O}(\sqrt{n})$	1

Our results

Concurrent work [Zhou et al., EUROCRYPT ‘23] achieves same asymptotics from different techniques

Scheme	Amortized Server Computation	Amortized Bandwidth	Client sp	
Ours	Õ(√n)	Õ(1)	Õ(√n)	1
[Shi et al., CRYPTO ‘21]	Õ(√n)	Õ(1)	Õ(√n)	2
[Corrigan Gibbs et al., EUROCRYPT ‘22]	Õ(√n)	Õ(√n)	Õ(√n)	1

Our results

1. Construct pseudorandom sets with the following properties:
   a. Concise representation
   b. Fast membership testing
   c. Fast enumeration
   d. Adaptability: Supports adding and removing a constant number of elements while maintaining concise representation

Our results

1. Construct pseudorandom sets with the following properties:
    a. Concise representation
    b. Fast membership testing
    c. Fast enumeration
    d. *Adaptability:* Supports adding and removing a constant number of elements while maintaining concise representation

Resulting key must hide which elements were operated on.

Our results

1. Construct pseudorandom sets with the following properties:
   a. Concise representation
   b. Fast membership testing
   c. Fast enumeration
   d. Adaptability: Supports adding and removing a constant number of elements while maintaining concise representation

2. Show that we can use such pseudorandom sets to construct a PIR scheme with the complexities aforementioned.

Our results

1. Construct pseudorandom sets with the following properties:
   a. Concise representation
   b. Fast membership testing
   c. Fast enumeration
   d. Adaptability: Supports adding and removing a constant number of elements while maintaining concise representation

2. Show that we can use such pseudorandom sets to construct a PIR scheme with the complexities aforementioned.

Our results

1. Construct pseudorandom sets with the following properties:
   a. Concise representation
   b. Fast membership testing
   c. Fast enumeration
   d. Adaptability: Supports adding and removing a constant number of elements while maintaining concise representation

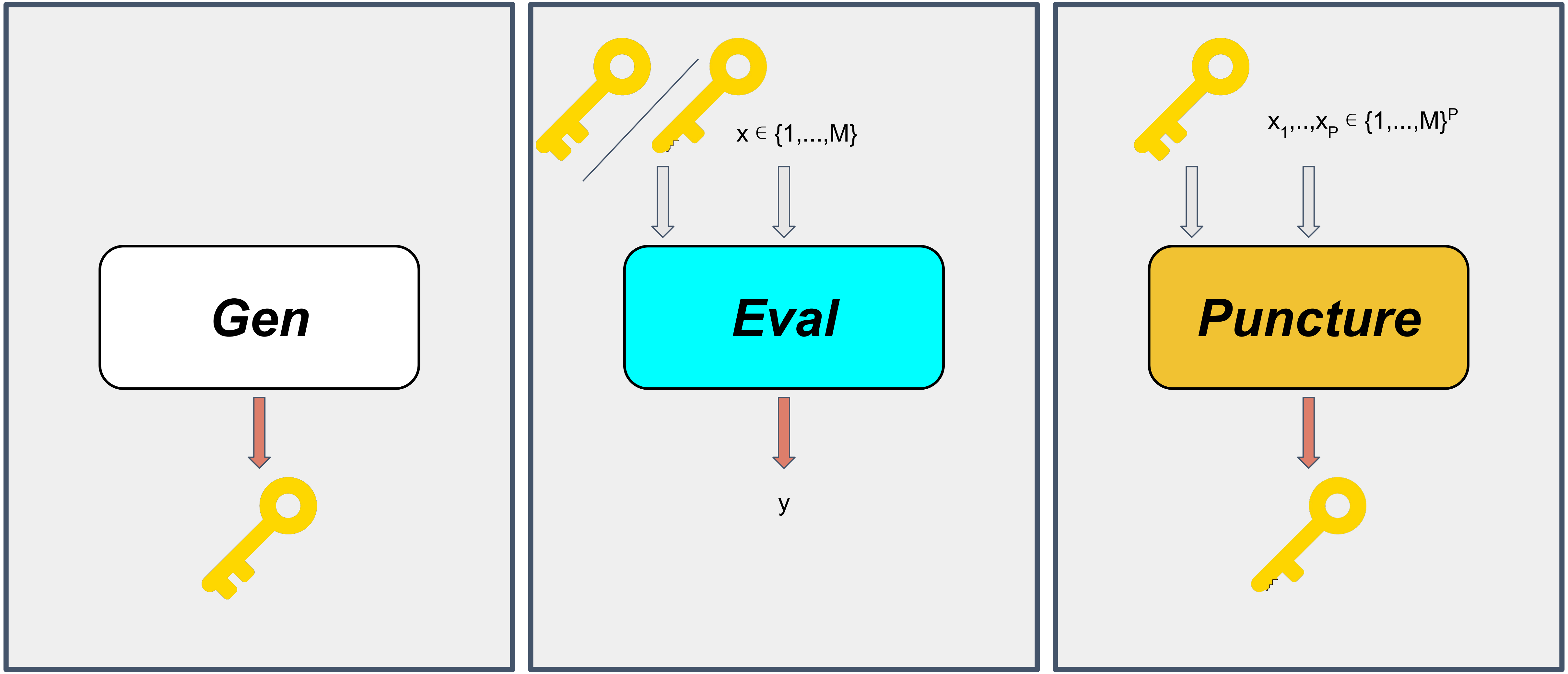
2. Show that we can use such pseudorandom sets to construct a PIR scheme with the complexities aforementioned.

[Shi et al., CRYPTO ‘21] previously achieved sets supporting (a)(b)(c) while allowing for a single removal, our construction allows for any constant number of additions and removals.

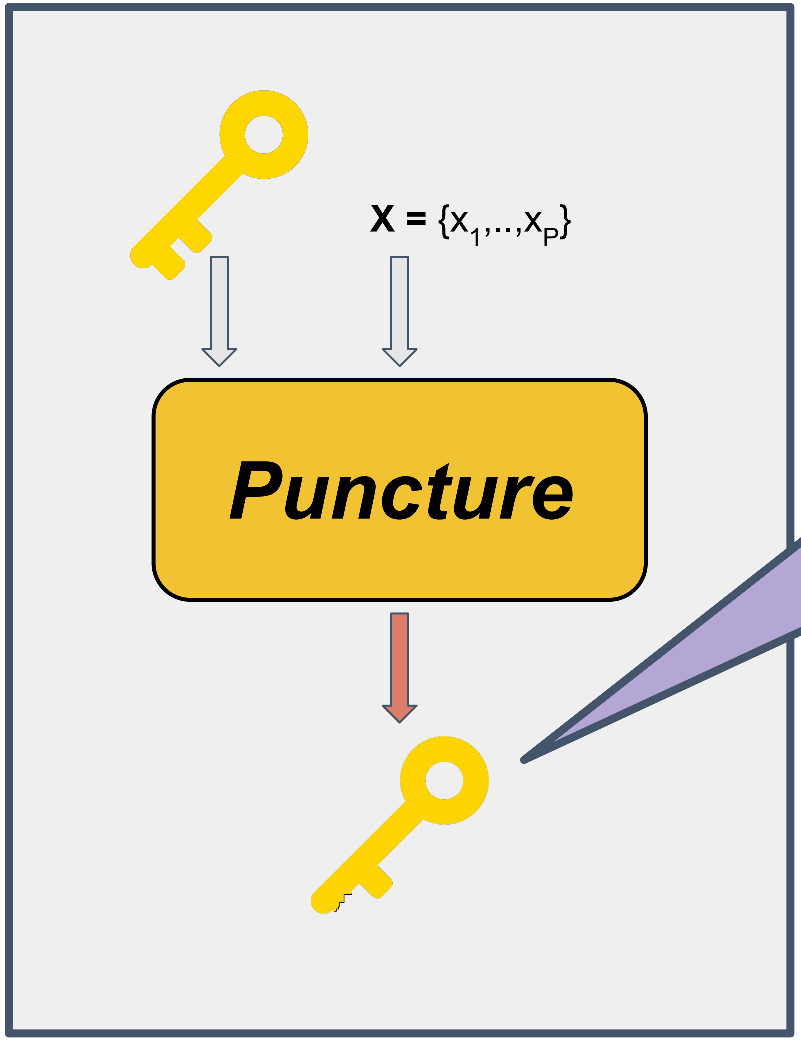
Starting point

- Our starting point is the privately puncturable PRF primitive.

Tool: Privately Puncturable PRF [BLW ‘15, BKM ‘17, CC ‘17, ...]



Puncture on Privately Puncturable PRF [BLW ‘15, BKM ‘17, CC ‘17, ...]



For this presentation: Puncture returns a key k’ where:

1. Outputs for points $x_1,\ldots,x_P$ are ‘re-randomized’.
2. Given k’ adversary cannot figure out which points were punctured.

Set definition [SACM '21]

- Given: Privately puncturable PRF F: $\{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}$.
- Want: set containing approximately $\sqrt{n}$ elements in $\{0,...,n-1\}$ picked from some sampling distribution over $\{0,...,n-1\}$.

Set definition [SACM '21]

- Given: Privately puncturable PRF F: $\{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}$.
- Want: set containing approximately $\sqrt{n}$ elements in $\{0,...,n-1\}$ picked from some sampling distribution over $\{0,...,n-1\}$.

- For each x in $\{0,...,n-1\}$:
  - $x \in S_k$ **iff** $F_k(x[i:]) = 1 \ \forall i \in \{0,...,\log(n)/2\}$.

Set definition (by example)

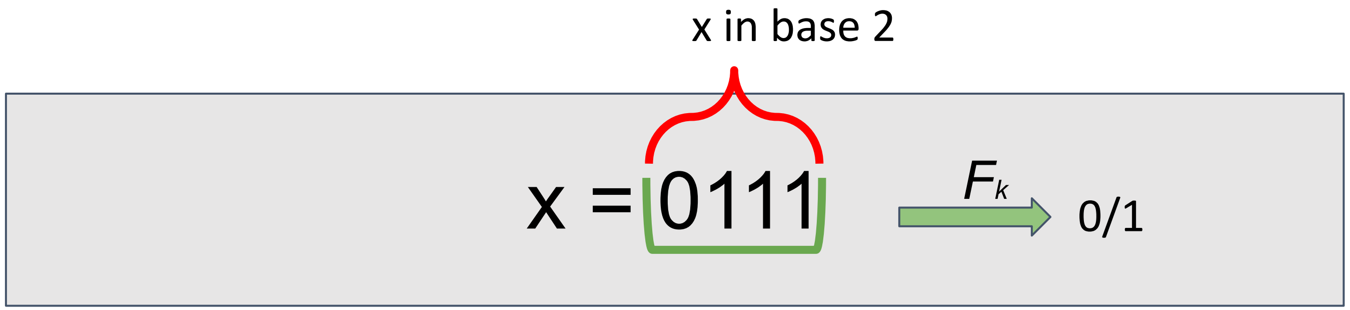
- Let n = 16, $k \leftarrow$ppPRF.Gen() represent our set, and let x = 7 = 0111.

x in base 2

x = 0111

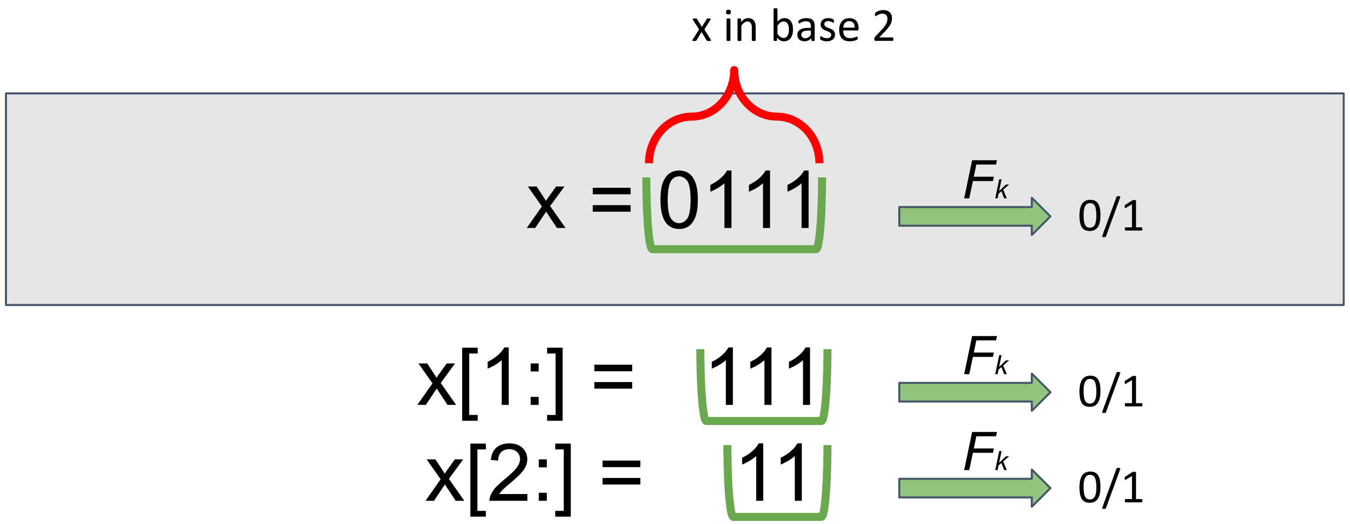
Set definition (by example)

- Let n = 16, $k \leftarrow$ppPRF.Gen() represent our set, and let x = 7 = 0111.



Set definition (by example)

- Let n = 16, $k \leftarrow$ppPRF.Gen() represent our set, and let x = 7 = 0111.



Set definition (by example)

- Let n = 16, $k \leftarrow$ppPRF.Gen() represent our set, and let x = 7 = 0111.

x in base 2

$x = 0111 \xrightarrow{F_k} 0/1$

$x[1:] = 111 \xrightarrow{F_k} 0/1$

$x[2:] = 11 \xrightarrow{F_k} 0/1$

$S_k = \{x : \forall i \in \{0,..,\log(n)/2\}, F_k(x[i:]) = 1 \wedge x \in \{0,...,n-1\}\}$

Set definition (by example)

- Let $n = 16$, $k \leftarrow$ppPRF.Gen() represent our set, and let $x = 7 = 0111$.

Test membership in Õ(1) time.

x in base 2

$x = 0111 \xrightarrow{F_k} 0/1$

$x[1:] = 111 \xrightarrow{F_k} 0/1$

$x[2:] = 11 \xrightarrow{F_k} 0/1$

$S_k = \{x : \forall i \in \{0,..,\log(n)/2\}, F_k(x[i:]) = 1 \wedge x \in \{0,...,n-1\}\}$

Set definition (by example)

- Let n = 16, $k \leftarrow$ppPRF.Gen() represent our set, and let x = 7 = 0111.

Test membership in Õ(1) time.

Enumerate in Õ(√n) time by starting from all strings of size log(n)/2 and appending 0/1 only to those that evaluate to 1.

x in base 2

$x = 0111 \xrightarrow{F_k} 0/1$

$x[1:] = 111 \xrightarrow{F_k} 0/1$

$x[2:] = 11 \xrightarrow{F_k} 0/1$

$S_k = \{x : \forall i \in \{0,..,\log(n)/2\}, F_k(x[i:]) = 1 \wedge x \in \{0,...,n-1\}\}$

Set definition (by example)

- Let n = 16, $k \leftarrow$ppPRF.Gen() represent our set, and let x = 7 = 0111.

Concise representation

Fast membership testing

Fast enumeration

Adaptability

Set definition (by example)

- Let n = 16, $k \leftarrow$ppPRF.Gen() represent our set, and let x = 7 = 0111.

Concise representation ✓

Fast membership testing ✓

Fast enumeration ✓

Adaptability ?

Adding and removing elements

Remove (k, x):
- Let **Z** be set of points that define x’s membership.
- Output new set key k’ = ppPRF.Puncture(k, **Z**).

Adding and removing elements

Remove (k, x):
- Let **Z** be set of points that define x’s membership.
- Output new set key k’ = ppPRF.Puncture(k, **Z**).

Very likely removes element from set.

Adding and removing elements

Remove (k, x):
- Let **Z** be set of points that define x's membership.
- Output new set key k' = ppPRF.Puncture(k, **Z**).

Very likely removes element from set.

Punctured key indistinguishable from freshly sampled key from distribution.

Adding and removing elements

Remove (k, x):

- Let **Z** be set of points that define x's membership.
- Output new set key k' = ppPRF.Puncture(k, **Z**).

Very likely removes element from set.

Punctured key indistinguishable from freshly sampled key from distribution.

With some probability, does not remove or removes other elements.

Adding and removing elements

<u>Remove (k, x):</u>

- Let **Z** be set of points that define x's membership.
- Output new set key k' = ppPRF.Puncture(k, **Z**).

<u>Add (k, x):</u>

- Let **Z** be set of points that define x's membership.
- Run k' = ppPRF.Puncture(k, **Z**) *until* $F(k', z) = 1\ \forall z \in \mathbf{Z}$. Output k'.

Adding and removing elements

Uses privately puncturable PRF where puncture operation is randomized [Canetti and Chen '17].

Remove (k, x):

- Let **Z** be set of points that define x's membership.
- Output new set key k' = ppPRF.Puncture(k, **Z**).

Add (k, x):

- Let **Z** be set of points that define x's membership.
- Run k' = ppPRF.Puncture(k, **Z**) *until* $F(k', z) = 1 \ \forall z \in \mathbf{Z}$. Output k'.

Adding and removing elements

Remove (k, x):
- Let **Z** be set of points that define x's membership.
- Output new set key k' = ppPRF.Puncture(k, **Z**).

Add (k, x):
- Let **Z** be set of points that define x's membership.
- Run k' = ppPRF.Puncture(k, **Z**) *until* *F*(k',z) = 1 $\forall$ z $\in$ **Z.** Output k'.

Always adds element to set.

Adding and removing elements

<u>Remove (k, x):</u>

- Let **Z** be set of points that define x's membership.
- Output new set key k' = ppPRF.Puncture(k, **Z**).

<u>Add (k, x):</u>

- Let **Z** be set of points that define x's membership.
- Run k' = ppPRF.Puncture(k, **Z**) *until* *F*(k',z) = 1 $\forall$ z $\in$ **Z.** Output k'.

Always adds element to set.

Punctured key is indistinguishable from sampling distribution until finding a set with x.

Adding and removing elements

<u>Remove (k, x):</u>

- Let **Z** be set of points that define x's membership.
- Output new set key k' = ppPRF.Puncture(k, **Z**).

<u>Add (k, x):</u>

- Let **Z** be set of points that define x's membership.
- Run k' = ppPRF.Puncture(k, **Z**) *until* $F(k',z) = 1\ \forall z \in \mathbf{Z}$. Output k'.

Always adds element to set.

Punctured key is indistinguishable from sampling distribution until finding a set with x.

With some probability, adds other elements.

Adding and removing elements

Remove (k, x

- Let **Z** be s … define x's x's memb
- Output ne … (k, **Z**) *until* ppPRF.Pu … ut k'.

How to make both Add and Remove work together?
Both access the same PRF operation.

Adding and removing elements

<u>Remove (k, x):</u>

- Let **Z** be set of points that define x's membership.
- Output new set key k' = ppPRF.Puncture(k, **Z**).

<u>Add (k, x):</u>

- Let **Z** be set of points that define x's membership.
- Run k' = ppPRF.Puncture(k, **Z**) *until* $F(k',z) = 1 \ \forall z \in \mathbf{Z}$. Output k'.

- Use C privately puncturable PRF keys to define set, where C is the total number of additions + removals we would like to support:

Adding and removing elements

Remove (k, x):
- Let **Z** be set of points that define x's membership.
- Output new set key k' = ppPRF.Puncture(k, **Z**).

Add (k, x):
- Let **Z** be set of points that define x's membership.
- Run k' = ppPRF.Puncture(k, **Z**) *until* $F(k',z) = 1 \; \forall z \in \mathbf{Z}$. Output k'.

- Use C privately puncturable PRF keys to define set, where C is the total number of additions + removals we would like to support:
  - Membership is defined as the XOR of the evaluation of each key on the points seen before. Let set key $\mathbf{k} = \{k_1,...,k_C\}$ where k_i are ppPRF keys. Then:

Adding and removing elements

<u>Remove (k, x):</u>

- Let **Z** be set of points that define x's membership.
- Output new set key k' = ppPRF.Puncture(k, **Z**).

<u>Add (k, x):</u>

- Let **Z** be set of points that define x's membership.
- Run k' = ppPRF.Puncture(k, **Z**) *until* $F(k',z) = 1 \ \forall z \in \mathbf{Z}$. Output k'.

- Use C privately puncturable PRF keys to define set, where C is the total number of additions + removals we would like to support:
  - Membership is defined as the XOR of the evaluation of each key on the points seen before. Let set key **k** = $\{k_1,\ldots,k_C\}$ where k_i are ppPRF keys. Then:
    - $x \in S_{\mathbf{k}}$ **iff** $\forall i \in \{0,..,\log(n)/2\}$, $F_{k_1}(x[i:]) \oplus \ldots \oplus F_{k_C}(x[i:]) = 1$.

Adding and removing elements

<u>Remove (k, x):</u>
- Let **Z** be set of points that define x's membership.
- Output new set key k' = ppPRF.Puncture(k, **Z**).

<u>Add (k, x):</u>
- Let **Z** be set of points that define x's membership.
- Run k' = ppPRF.Puncture(k, **Z**) *until* $F(k',z) = 1 \;\forall z \in \mathbf{Z}$. Output k'.

- Use C privately puncturable PRF keys to define set, where C is the total number of additions + removals we would like to support:
  - Membership is defined as the XOR of the evaluation of each key on the points seen before. Let set key $\mathbf{k} = \{k_1,\ldots,k_C\}$ where k_i are ppPRF keys. Then:
    - $x \in S_{\mathbf{k}}$ **iff** $\forall i \in \{0,..,\log(n)/2\}$, $F_{k_1}(x[i:]) \oplus \ldots \oplus F_{k_C}(x[i:]) = 1$.
  - We show additions and removals can be done sequentially on each key $k_1,\ldots,k_C$ and satisfy appropriate notions of privacy.

Our results

1. Construct pseudorandom sets with the following properties:
   a. Concise representation
   b. Fast membership testing
   c. Fast enumeration
   d. Adaptability: Supports adding and removing a constant number of elements while maintaining concise representation

2. Show that we can use such pseudorandom sets to construct a PIR scheme with the complexities aforementioned.

Thank you!

Image sources:

https://sproutsocial.com/insights/facebook-ad-examples/

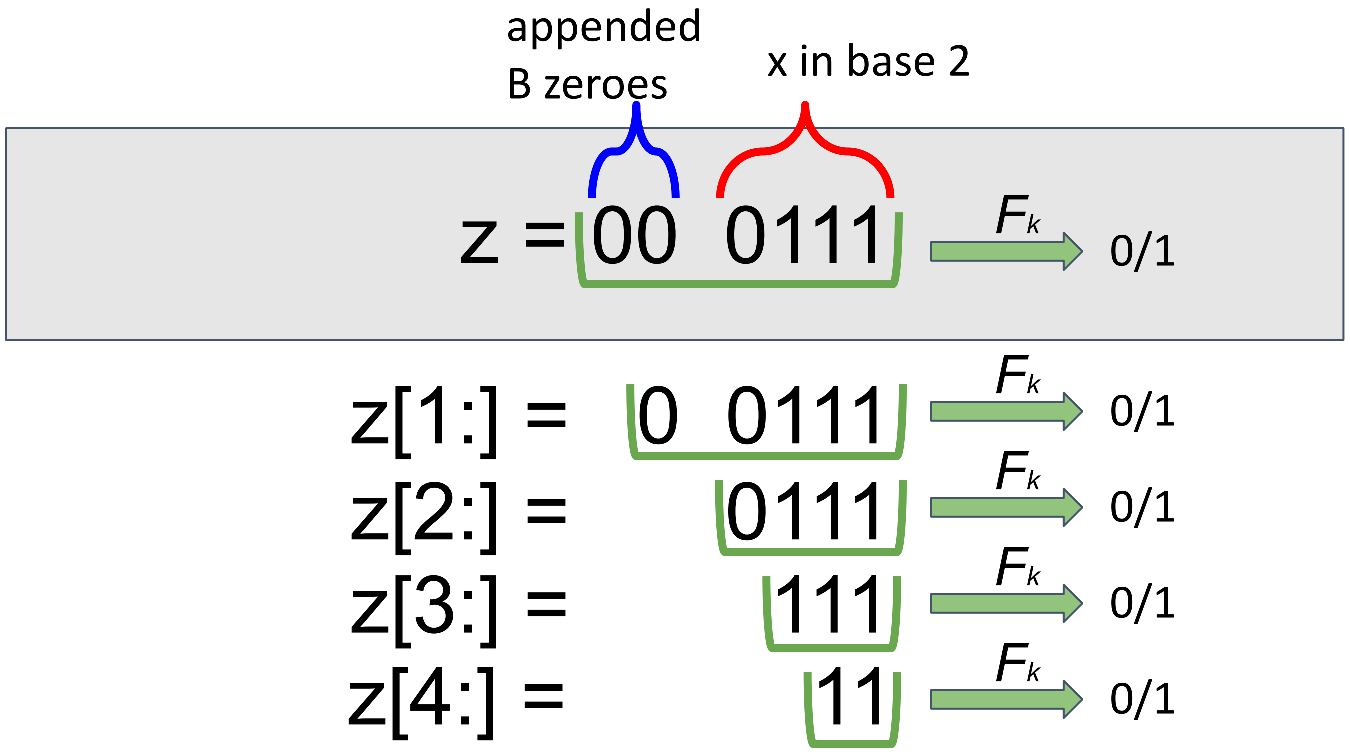
https://icon-library.com/icon/key-icon-png-7.html.html

https://www.freepnglogos.com/images/tick-33835.html

https://cacm.acm.org/magazines/2019/9/238971-metadata-private-communication-for-the-99/abstract

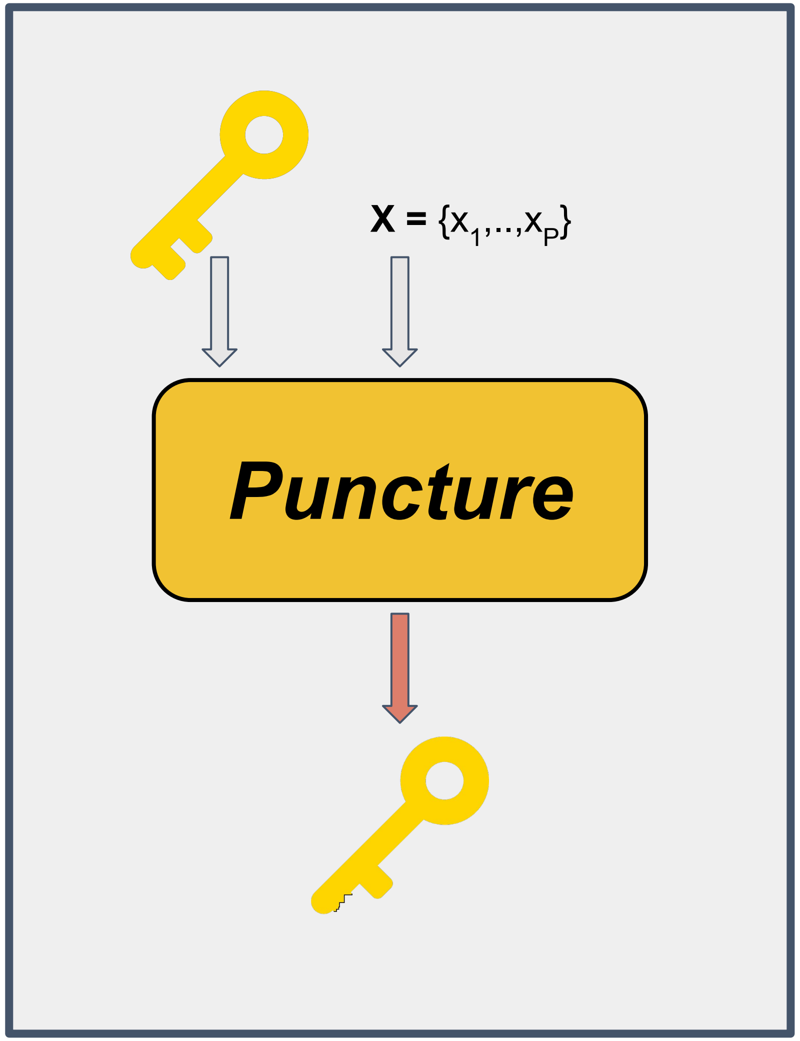
Set definition (by example)

- Let $n = 16$, $k \leftarrow$ppPRF.Gen() represent our set, and let $x = 7 = 0111$.
- Let $z = 0^B || x$, $B = \log(\log(n))$.



$S_k = \{x : \forall i \in \{0,..,\log(n)/2 + B\}, F_k(z[i:]) = 1 \text{ for } z = 0^B \,||\, x \wedge x \in \{0,...,n-1\}\}$

Puncture on Privately Puncturable PRF [BLW ‘15, BKM ‘17, CC ‘17, ...]



Correctness:

For any input $x' \notin \mathbf{X}$, punctured key evaluates to same output as original key.

Security:

New key contains no information about original evaluation at punctured points.

Privacy:

New key contains no information about what points were punctured.