Part I: Constraining PRFs Privately

David Wu
Stanford University

Joint work with Dan Boneh and Kevin Lewi
Pseudorandom Functions (PRFs) [GGM84]

\[ F: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y} \]
Constrained PRFs [BW13, BGI13, KPTZ13]

Constrained PRF: PRF with additional “constrain” functionality

\[ F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y} \]

can be used to evaluate at all points \( x \in \mathcal{X} \) where \( C(x) = 1 \)
Constrained PRFs [BW13, BGI13, KPTZ13]

**Correctness:** constrained evaluation at \( x \in \mathcal{X} \) where \( C(x) = 1 \) yields PRF value at \( x \)

**Security:** PRF value at points \( x \in \mathcal{X} \) where \( C(x) = 0 \) are indistinguishable from random
Constrained PRFs [BW13, BGI13, KPTZ13]

Many applications:
• Identity-Based Key Exchange, Optimal Broadcast Encryption [BW13]
• Punctured Programming Paradigm [SW14]
• Multiparty Key Exchange, Traitor Tracing [BZ14]
Puncturable PRFs from GGM

- Puncturable PRF: constrained keys allow evaluation at all but a single point
- Easily constructed from GGM:

\[ G(s) = s_0 \parallel s_1 \]

\[ G(s_0) = s_{00} \parallel s_{01} \]

\[ G(s_1) = s_{10} \parallel s_{11} \]
Puncturable PRFs from GGM

\[ G(s_0) = s_{00} \parallel s_{01} \]

\[ G(s_1) = s_{10} \parallel s_{11} \]

\[ S = s_0 \parallel s_1 \]

puncture at \( x = 01 \)
Puncturable PRFs from GGM

\[ G(s) = s_0 \parallel s_1 \]

these two values suffice to evaluate at all other points
Puncturable PRFs from GGM

\[ G(s_0) = s_{00} \parallel s_{01} \]

\[ G(s_1) = s_{10} \parallel s_{11} \]

given \( s_1 \) and \( s_{00} \), easy to tell that 01 is the punctured point
Can we build a constrained PRF where the constrained key for a circuit $C$ hides $C$?
Constraining PRFs Privately

\[ \text{msk} \leftarrow \text{Setup}(1^\lambda) \]

World 0

Constrain(msk, C_0)

\[ C_0, C_1 \]

World 1

Constrain(msk, C_1)

\[ \approx_c \]

Single-key privacy

Definitions generalize to multi-key privacy. See paper for details.
Private Puncturing

- **Correctness:** constrained evaluation at $x \neq z$ yields $F(k, x)$
- **Security:** $F(k, z)$ is indistinguishable from random
- **Privacy:** constrained key hides $z$
Implications of Privacy

Consider value of ConstrainEval($sk_z, z$):

• **Security**: Independent of $\text{Eval}(msk, z)$
• **Privacy**: Unguessable by the adversary
Using Privacy: Restricted Keyword Search

PRF$_k$(Honeycomb) $\rightarrow \{5,8,13\}$
PRF$_k$(KitKat) $\rightarrow \{18,21\}$
PRF$_k$(Lollipop) $\rightarrow \{3,10,11\}$
PRF$_k$(Marshmallow) $\rightarrow \{1,9,22\}$

The server with encrypted index: $\text{ConstrainEval}(\text{sk}, \text{Honeycomb})$ $\rightarrow \{5,8,13\}$
Using Privacy: Restricted Keyword Search

\[ \text{PRF}_k(\text{Honeycomb}) \rightarrow \{5, 8, 13\} \]
\[ \text{PRF}_k(\text{KitKat}) \rightarrow \{18, 21\} \]
\[ \text{PRF}_k(\text{Lollipop}) \rightarrow \{3, 10, 11\} \]
\[ \text{PRF}_k(\text{Marshmallow}) \rightarrow \{1, 9, 22\} \]

search for non-existent keyword

server with encrypted index

ConstrainEval(\text{sk, Jelly Bean})

No results
Using Privacy: Restricted Keyword Search

PRF$_k$(Honeycomb) → {5, 8, 13}
PRF$_k$(KitKat) → {18, 21}
PRF$_k$(Lollipop) → {3, 10, 11}
PRF$_k$(Marshmallow) → {1, 9, 22}

search for “restricted” keyword

server with encrypted index

ConstrainEval(sk, Marshmallow)

No results
Using Privacy: Restricted Keyword Search

- **Security**: ConstrainEval\((sk, \text{Marshmallow})\) ≠ Eval\((msk, \text{Marshmallow})\)
- **Privacy**: Does not learn that no results were returned because no matches for keyword or if the keyword was restricted

\[
\begin{align*}
\text{PRF}_k(\text{Honeycomb}) &\rightarrow \{5,8,13\} \\
\text{PRF}_k(\text{KitKat}) &\rightarrow \{18,21\} \\
\text{PRF}_k(\text{Lollipop}) &\rightarrow \{3,10,11\} \\
\text{PRF}_k(\text{Marshmallow}) &\rightarrow \{1,9,22\}
\end{align*}
\]

server with encrypted index

ConstrainEval\((sk, \text{Marshmallow})\)

No results

User

Key
The Many Applications of Privacy

• Private constrained MACs
  • Parties can only sign messages satisfying certain policy (e.g., enforce a spending limit), but policies are hidden

• Symmetric Deniable Encryption [CDNO97]
  • Two parties can communicate using a symmetric encryption scheme
  • If an adversary has intercepted a sequence of messages and coerces one of the parties to produce a decryption key for the messages, they can produce a “fake” key that decrypts all but a subset of the messages

• Constructing a family of watermarkable PRFs
  • Can be used to embed a secret message within a PRF that is “unremovable” – useful for authentication [CHNVW15]

See paper for details!
Summary of our Constructions

• From indistinguishability obfuscation (iO):
  • Private puncturable PRFs from iO + one-way functions
  • Private circuit constrained PRFs from sub-exponentially hard iO + one-way functions

• From **concrete** assumptions on multilinear maps:
  • Private puncturable PRFs from subgroup hiding assumptions

• Private bit-fixing PRF from multilinear Diffie-Hellman assumption
Constructing Private Constrained PRFs

Tool: indistinguishability obfuscation [BGI$^+$01, GGH$^+$13]

Program $P_1$

Program $P_2$

\[ \forall x : P_1(x) = P_2(x) \]

\[ \text{iO}(P_1) \approx_c \text{iO}(P_2) \]
Private Puncturing from iO

• Starting point: puncturable PRFs (e.g. GGM)

• Need a way to hide the point that is punctured
  • Intuition: obfuscate the puncturable PRF

• Question: what value to output at the punctured point?
Private Puncturing from iO

Use iO to hide the punctured point and output uniformly random value at punctured point.

\[ P_z(x) : \]
- If \( x = z \), output \( r \)
- Else, output \( \text{PRF}(k, x) \)

Program for punctured PRF (punctured at \( z \))

Random value (hard coded)

Real value of the PRF
Private Puncturing from iO

Suppose PRF is puncturable (e.g., GGM)
- Master secret key: PRF key $k$
- PRF output at $x \in \mathcal{X}$: $\text{PRF}(k, x)$

$P_z(x)$:
- If $x = z$, output $r$
- Else, output $\text{PRF}(k, x)$

Punctured key for a point $z$ is an obfuscated program
Constrained evaluation corresponds to evaluating obfuscated program
Private Puncturing from iO: Privacy

Recall privacy notion:

\[ \approx_c \]

\[ msk \leftarrow \text{Setup}(1^\lambda) \]

\[ x_0, x_1 \in X \]

Puncture\((k, x_0)\)

\[ \approx_c \]

\[ msk \leftarrow \text{Setup}(1^\lambda) \]

\[ x_0, x_1 \in X \]

Puncture\((k, x_1)\)
Private Puncturing from iO: Privacy

Proof is simple exercise in punctured programming

\[ P_z(x) : \]
- If \( x = z \), output \( r \)
- Else, output \( \text{PRF}(k, x) \)

Program for punctured PRF (punctured at \( z \))

random value (hard coded)

real value of the PRF
Private Puncturing from iO: Privacy

Hybrid 0: Real game

\[ P_{x_0}(x): \]
- If \( x = x_0 \), output \( r \)
- Else, output \( \text{PRF}(k, x) \)

Hybrid 1: Challenger responds to puncture query with iO of this program

\[ P'_{x_0}(x): \]
- If \( x = x_0 \), output \( r \)
- Else, output \( \text{PRF}(k_{x_0}, x) \)
Private Puncturing from iO: Privacy

Invoke puncturing security

\[ P'_{x_0}(x): \]
- If \( x = x_0 \), output \( r \)
- Else, output \( \text{PRF}(k_{x_0}, x) \)

\[ P''_{x_0}(x): \]
- If \( x = x_0 \), output \( \text{PRF}(k, x_0) \)
- Else, output \( \text{PRF}(k_{x_0}, x) \)
Private Puncturing from iO: Privacy

Invoke iO security

\[ P''_x(x) : \]
- If \( x = x_0 \), output \( \text{PRF}(k, x_0) \)
- Else, output \( \text{PRF}(k_{x_0}, x) \)

\approx_c iO

Hybrid 2

\[ P'''_x(x) : \]
- Output \( \text{PRF}(k, x) \)

Hybrid 3

The program in Hybrid 3 is independent of \( x_0 \). Similar argument holds starting from \( P_{x_1}(x) \).
Private Puncturing from iO: Summary

Use iO to hide the punctured point and output uniformly random value at punctured point

$P_z(x)$:
- If $x = z$, output $r$
- Else, output $\text{PRF}(k, x)$
Private Circuit Constrained PRF from iO

Construction generalizes to circuit constraints, except random values now derived from another PRF

\[ P_C(x): \]
- If \( C(x) = 0 \), output \( \text{PRF}(k', x) \)
- If \( C(x) = 1 \), output \( \text{PRF}(k, x) \)

\( k' \) is independently sampled PRF key

“real” PRF value
Recall intuitive requirements for private constrained PRF:

- **Security**: Values at constrained points independent of actual PRF value at those points
- **Privacy**: Values at constrained points are unguessable by the adversary

\[ P_C(x) : \]
- If \( C(x) = 0 \), output PRF\((k', x)\)
- If \( C(x) = 1 \), output PRF\((k, x)\)
Private Circuit Constrained PRF from iO

Security proof similar to that for private puncturable PRF

Number of hybrids equal to number of points that differ across the two circuits, so sub-exponential hardness needed in general

$P_C(x)$:
- If $C(x) = 0$, output PRF($k', x$)
- If $C(x) = 1$, output PRF($k, x$)
Private Puncturing from Multilinear Maps

• Composite-order (ideal) multilinear maps* [BS04]
  • Fix composite modulus $N = pq$

• Base group $\mathbb{G}_1$ and target group $\mathbb{G}_n$ (of order $N$) with canonical generators $g_1$ and $g_n$, respectively

• Multilinear map operation:
  $$e(g_1^{\alpha_1}, g_1^{\alpha_2}, \ldots, g_1^{\alpha_n}) = g_n^{\alpha_1 \alpha_2 \cdots \alpha_n}$$

*For simplicity, we describe our construction using ideal multilinear maps. It is straightforward to translate our construction to use composite-order graded multilinear encodings [CLT13]
Private Puncturing from Multilinear Maps

• Composite-order (ideal) multilinear maps [BS04]
  • Let $\mathbb{G}_1, \mathbb{p}$ be subgroup of order $p$ of $\mathbb{G}_1$

• Subgroup decision assumption [BGN05]: hard to distinguish random elements of the full group $\mathbb{G}_1$ from random elements of the subgroup $\mathbb{G}_{1,p}$
Private Puncturing from Multilinear Maps

Starting point: multilinear analog of Naor-Reingold [NR97, BW13]

<table>
<thead>
<tr>
<th>$g_1^{\alpha_{1,0}}$</th>
<th>$g_1^{\alpha_{2,0}}$</th>
<th>...</th>
<th>$g_1^{\alpha_{n,0}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g_1^{\alpha_{1,1}}$</td>
<td>$g_1^{\alpha_{2,1}}$</td>
<td>...</td>
<td>$g_1^{\alpha_{n,1}}$</td>
</tr>
</tbody>
</table>

master secret key:

collection of 2n random group elements from $\mathbb{G}_1$
Private Puncturing from Multilinear Maps

PRF evaluation via multilinear map

<table>
<thead>
<tr>
<th>$g_1^{\alpha_{1,0}}$</th>
<th>$g_1^{\alpha_{2,0}}$</th>
<th>$g_1^{\alpha_{3,0}}$</th>
<th>$g_1^{\alpha_{4,0}}$</th>
<th>$g_1^{\alpha_{5,0}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g_1^{\alpha_{1,1}}$</td>
<td>$g_1^{\alpha_{2,1}}$</td>
<td>$g_1^{\alpha_{3,1}}$</td>
<td>$g_1^{\alpha_{4,1}}$</td>
<td>$g_1^{\alpha_{5,1}}$</td>
</tr>
</tbody>
</table>
Private Puncturing from Multilinear Maps

PRF evaluation via multilinear map

$$F_k(01101) = e(g_1^{\alpha_{1,0}}, g_1^{\alpha_{2,1}}, g_1^{\alpha_{3,1}}, g_1^{\alpha_{4,0}}, g_1^{\alpha_{5,1}})$$
Private Puncturing from Multilinear Maps

Puncture PRF by exploiting orthogonality

master secret key:

\[ g_1^{\alpha_{1,0}}, g_1^{\alpha_{2,0}}, g_1^{\alpha_{3,0}}, g_1^{\alpha_{4,0}}, g_1^{\alpha_{5,0}} \]

\[ g_1^{\alpha_{1,1}}, g_1^{\alpha_{2,1}}, g_1^{\alpha_{3,1}}, g_1^{\alpha_{4,1}}, g_1^{\alpha_{5,1}} \]

all elements in subgroup

puncture at 01101:

\[ g_1^{\alpha_{1,0}}, g_1^{\alpha_{2,0}}, g_1^{\alpha_{3,0}}, g_1^{\alpha_{4,0}}, g_1^{\alpha_{5,0}} \]

\[ g_1^{\alpha_{1,1}}, g_1^{\alpha_{2,1}}, g_1^{\alpha_{3,1}}, g_1^{\alpha_{4,1}}, g_1^{\alpha_{5,1}} \]

punctured components in full group
Private Puncturing from Multilinear Maps

Correctness

puncture at \( x^* = 01101 \):

<table>
<thead>
<tr>
<th>( g_1^{\alpha_{1,0}} )</th>
<th>( g_1^{\alpha_{2,0}} )</th>
<th>( g_1^{\alpha_{3,0}} )</th>
<th>( g_1^{\alpha_{4,0}} )</th>
<th>( g_1^{\alpha_{5,0}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_1^{\alpha_{1,1}} )</td>
<td>( g_1^{\alpha_{2,1}} )</td>
<td>( g_1^{\alpha_{3,1}} )</td>
<td>( g_1^{\alpha_{4,1}} )</td>
<td>( g_1^{\alpha_{5,1}} )</td>
</tr>
</tbody>
</table>

Correctness by multilinearity (and CRT):

\[
e \left( g_1^{\beta_1}, ..., g_1^{\beta_n} \right) = e(g_1^{\beta_1}, ..., g_1^{\beta_n})^{\beta_1 \cdots \beta_n \pmod{p}} e(g_1^{\beta_1}, ..., g_1^{\beta_n})^{\beta_1 \cdots \beta_n \pmod{q}}
\]

For all \( x \neq x^* \), there is some \( i \) where \( x_i \neq x_i^* \) so \( \beta_i, x_i^* = 0 \pmod{q} \)

where \( (g^{\beta_{i,0}}, g^{\beta_{i,1}}) \) is the \( i \)th component of the secret key
Private Puncturing from Multilinear Maps

Privacy

puncture at $x^* = 01101$:

\[
\begin{array}{c|c|c|c|c}
\alpha_{1,0} & \alpha_{2,0} & \alpha_{3,0} & \alpha_{4,0} & \alpha_{5,0} \\
\hline
\alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1} & \alpha_{5,1} \\
\hline
\end{array}
\]

Follows directly by subgroup decision: elements of $\mathbb{G}_1$ look indistinguishable from elements of $\mathbb{G}_{1,p}$.
Private Puncturing from Multilinear Maps

Puncturing Security

puncture at $x^* = 01101$:

Follows from a multilinear Diffie-Hellman subgroup decision assumption on composite-order multilinear maps

See paper for details!
Private Puncturing and Distributed Point Functions [GI14]

\[
\text{Gen}(1^\lambda, x)
\]

\[
\begin{align*}
\text{Eval}(k_0, x') \oplus \text{Eval}(k_1, x') &= \begin{cases} 
1, & x' = x \\
0, & x' \neq x
\end{cases}
\end{align*}
\]

**Privacy:** \(k_0\) and \(k_1\) individually must hide \(x\)

**Correctness:** \(k_0\) and \(k_1\) implement a point function
A private puncturable PRF can be used to build a distributed point function (DPF):

\[
\text{Gen}(1^\lambda, x)
\]

\[
k_0 \leftarrow \text{Setup}(1^\lambda)
\]

\[
k_1 \leftarrow \text{Puncture}(k_0, x)
\]

**Correctness:** \(\text{Eval}(k_0, \cdot)\) and \(\text{ConstrainEval}(k_1, \cdot)\) agree everywhere except \(x\)

**Privacy:** \(k_0\) is independent of \(x\) and \(k_1\) hides \(x\)
Private Puncturing and Distributed Point Functions [GI14]

However, distributed point functions do not give a private puncturable PRF

**Key difference:**

- In a DPF, the point $x$ is known at setup time: both $k_0$ and $k_1$ are generated together
- In a private puncturable PRF, the point $x$ is known after the master parameters (the key $k_0$) are generated

**Open question:** Can constructions of DPFs be adapted to obtain a private puncturable PRF?
Conclusions

• New notion of private constrained PRFs

• Simple definitions, but require powerful tools to construct: iO / multilinear maps

• Private constrained PRFs immediately provide natural solutions to many problems
Open Questions

• Puncturable PRFs (and DPFs) can be constructed from OWFs
  • Can we construct private puncturable PRFs from OWFs?
  • Does private puncturing necessitate strong assumptions like multilinear maps?
  • Can we construct private circuit-constrained PRFs without requiring sub-exponentially hard iO?

• Most of our candidate applications just require private puncturable PRFs
  • New applications for more expressive families of constraints?
Part II: Practical Order-Revealing Encryption with Limited Leakage

Joint work with Nathan Chenette, Kevin Lewi, and Stephen A. Weis
Order-Revealing Encryption [BLRSZZ15]

\[ \begin{align*}
    ct_1 &= \text{Enc}(sk, 123) \\
    ct_2 &= \text{Enc}(sk, 512) \\
    ct_3 &= \text{Enc}(sk, 273)
\end{align*} \]

secret-key encryption scheme
Order-Revealing Encryption [BLRSZZ15]

\[
\begin{align*}
ct_1 &= \text{Enc}(sk, 123) \\
ct_2 &= \text{Enc}(sk, 512) \\
ct_3 &= \text{Enc}(sk, 273)
\end{align*}
\]

Which is greater: the value encrypted by \(ct_1\) or the value encrypted by \(ct_2\)?

Application: range queries / binary search on encrypted data
Order-Revealing Encryption [BLRSZZ15]

Given any two ciphertexts

\[ \text{ct}_1 = \text{Enc}(sk, x) \]
\[ \text{ct}_2 = \text{Enc}(sk, y) \]

there is a publicly evaluable function that evaluates the comparison function

\[ x > y \]
Defining Security

Starting point: semantic security (IND-CPA) [GM84]

\[ b \in \{0, 1\} \]

\[ \text{Enc}(sk, m_b^{(i)}) \]

semantic security: adversary cannot guess \( b \) (except with probability negligibly close to \( 1/2 \))
Best-Possible Security [BCLO09]

\[ b \in \{0,1\} \]

\[ m_0^{(i)}, m_1^{(i)} \in \mathcal{M} \]

\[ \text{Enc}(sk, m_b^{(i)}) \]

must impose restriction on messages: otherwise trivial to break semantic security using comparison operator
Best-Possible Security [BCLO09]

\[ b \in \{0,1\} \]

\[ \text{sk} \]

\[ m_0^{(i)}, m_1^{(i)} \in \mathcal{M} \]

\[ \text{Enc}(\text{sk}, m_b^{(i)}) \]

\[ \forall i, j: m_0^{(i)} < m_0^{(j)} \iff m_1^{(i)} < m_1^{(j)} \]
Best-Possible Security [BCLO09]

\[ b \in \{0,1\} \]

\[ \text{sk} \]

\[ m_0^{(i)}, m_1^{(i)} \in M \]

\[ \text{Enc} \left( \text{sk}, m_b^{(i)} \right) \]

order of “left” set of messages same as order of “right” set of messages
Existing Approaches


- Powerful cryptographic primitive that fully subsumes ORE
- Achieves best-possible security
- Impractical (requires obfuscating a PRF)
Existing Approaches

Multilinear-map-based Solution [BLRSZZ15]

- Much more efficient than general purpose indistinguishability obfuscation
- Achieves best-possible security
- Security of multilinear maps not well-understood
- Still quite inefficient (e.g., ciphertexts on the order of GB)
Existing Approaches

Order-preserving encryption (OPE) [BCLO09, BCO11]:

- Comparison operation is **direct** comparison of ciphertexts:
  \[ x > y \iff \text{Enc}(sk, x) > \text{Enc}(sk, y) \]

- Lower bound: no OPE scheme can satisfy “best-possible” security unless the size of the ciphertext space is **exponential** in the size of the plaintext space.
Existing Approaches

Order-preserving encryption (OPE) [BCLO09, BCO11]:

- No “best-possible” security, so instead, compare with random order-preserving function (ROPF)

encryption function implements a random order-preserving function
Existing Approaches

Properties of a random order-preserving function [BCO’11]:

• Each ciphertext roughly leaks half of the most significant bits
• Each pair of ciphertexts roughly leaks half of the most significant bits of their difference

No semantic security for even a single message!
Existing Approaches

Security

Efficiency

OPE

Something in between?

Direct construction from multilinear maps

General-purpose MIFE from iO

Not drawn to scale
A New Security Notion

Two existing security notions:

- IND-OCPA: strong security, but hard to achieve efficiently
- ROPF-CCA: efficiently constructible, but lots of leakage, and difficult to precisely quantify the leakage
A New Security Notion: SIM-ORE

Idea: augment “best-possible” security with a leakage function $\mathcal{L}$
A New Security Notion: SIM-ORE

Similar to SSE definitions [CM05, CGKO06]

Leakage functions specifies exactly what is leaked

“Best-possible” simulation security:

\[ \mathcal{L}(m_1, \ldots, m_q) = \{ \mathbf{1}\{m_i < m_j\} \mid 1 \leq i < j \leq q \} \]
A New Security Notion: SIM-ORE

“Best-possible” simulation security:

\[ \mathcal{L}(m_1, \ldots, m_q) = \{1\{m_i < m_j\} \mid 1 \leq i < j \leq q\} \]

Anything that can be computed given the ciphertexts can be computed given the ordering on the messages.
Our Construction

Leak a little more than just the ordering:

\[ \mathcal{L}(m_1, \ldots, m_q) = \left\{ \left( \mathbf{1}_{m_i < m_j}, \text{ind}_{\text{diff}}(m_i, m_j) \right) \mid 1 \leq i < j \leq q \right\} \]

\[
\begin{array}{cccccc}
1 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 1
\end{array}
\]

\text{ind}_{\text{diff}}(m_1, m_2): \text{index of first bit that differs}
For each index $i$, apply a PRF to the first $i - 1$ bits, then add $b_i \pmod{n}$.
Our Construction

For each index $i$, apply a PRF to the first $i - 1$ bits, then add $b_i \mod n$.

$F: \mathcal{K} \times \{0,1\}^* \rightarrow \mathbb{Z}_3$
Our Construction

For each index $i$, apply a PRF to the first $i - 1$ bits, then add $b_i \pmod{n}$.

$F: K \times \{0,1\}^* \rightarrow \mathbb{Z}_3$
Our Construction

For each index $i$, apply a PRF to the first $i - 1$ bits, then add $b_i \pmod{n}$

$F: \mathcal{K} \times \{0,1\}^* \rightarrow \mathbb{Z}_3$
Our Construction

- \( F_k(\epsilon) + 1 \)
- \( F_k(1) + 0 \)
- \( F_k(10) + 0 \)
- \( F_k(100) + 1 \)
- \( F_k(1001) + 0 \)
- \( F_k(10010) + 1 \)

**same prefix = same ciphertext block**

- \( F_k(\epsilon) + 1 \)
- \( F_k(1) + 0 \)
- \( F_k(10) + 0 \)
- \( F_k(100) + 0 \)
- \( F_k(1000) + 1 \)
- \( F_k(10001) + 1 \)

**different prefix = value computationally hidden**

- \( F_k(\epsilon) + 1 \)
- \( F_k(1) + 0 \)
- \( F_k(10) + 0 \)
- \( F_k(100) + 0 \)
- \( F_k(1000) + 1 \)
- \( F_k(10001) + 1 \)

**compare values (mod n) to determine ordering**
Our Construction: Security

Security follows directly from security of the PRF

Proof sketch. Simulator responds to encryption queries using random strings. Maintains consistency using leakage information (first bit that differs).
The Landscape of OPE/ORE

- OPE
- Direct construction from multilinear maps
- General-purpose MIFE from iO
- Our construction

Not drawn to scale
Directions for Future Research

- Shorter ciphertexts?
- New leakage functions?
- Best-possible ORE from standard assumptions?

Our construction

- Direct construction from multilinear maps
- General-purpose MIFE from iO

Not drawn to scale
Questions?