Encryptor Combiners

Fermi Ma
Joint work with Mark Zhandry
Defining Obfuscation

**Goal:** Take a computer program and make it “unintelligible”.
Defining Obfuscation

**Goal:** Take a computer program and make it “unintelligible”.

**Why?**

- Prevent tampering
- Deter reverse engineering
- Hide cryptographic secrets
Defining Obfuscation

**Example:** What can we do with the python hello world program?

```python
print "Hello World!"
```
Defining Obfuscation

Example: What can we do with the python hello world program?

print "Hello World!"

Source: http://benkurtovic.com/2014/06/01/obfuscating-hello-world.html
Defining Obfuscation

As complicated as this looks, a determined adversary might be able to recover the original program.

Source: http://benkurtovic.com/2014/06/01/obfuscating-hello-world.html
Commercial vs. Mathematical Obfuscation

- Commercial obfuscators may rearrange parts of the program, rename variables, add useless code, etc.
- Code is very messy, but doesn’t *guarantee* any security.
Commercial vs. Mathematical Obfuscation

- Commercial obfuscators may rearrange parts of the program, rename variables, add useless code, etc.
- Code is very messy, but doesn’t guarantee any security.

Theoretical solution: define a mathematical obfuscator and formally prove security!
Mathematical Obfuscator

• Input: code for a program P
• Output: obfuscated code Obf(P)
• Correctness: Obf(P) and P must produce the same output for all possible inputs.
• Security?
Virtual Black Box (VBB) Obfuscation [BGI⁺01]

Observation: Given the obfuscation of a program P, we will can *always* learn its input/output behavior.
Virtual Black Box (VBB) Obfuscation [BGI+01]

Observation: Given the obfuscation of a program P, we will can always learn its input/output behavior.

Thus, the goal of VBB security is to hide everything beyond what we can learn from running the program.
Virtual Black Box (VBB) Obfuscation [BGI01]

\[ \Pr[ \text{outputs 1} ] - \Pr[ \text{outputs 1} ] < \text{negl} \]
General VBB is Impossible

Barak et al. show that VBB is unattainable. [BGI+01]
General VBB is Impossible

Barak et al. show that VBB is unattainable. [BGI+01]

**Intuition:** What can an adversary do given the code of Obf(P) that can’t be done with just black box access to P?
General VBB is Impossible

Barak et al. show that VBB is unattainable. [BGI⁺01]

**Intuition:** What can an adversary do given the code of Obf(P) that can’t be done with just black box access to P?

Adversary can run Obf(P) on itself!

Unclear how to do this with just a black box.
Indistinguishability Obfuscation

**Security**: Given two equivalent programs of the same length, their obfuscations are indistinguishable to any PPT adversary. [BGI+01]
Indistinguishability Obfuscation

**Security**: Given two equivalent programs of the same length, their obfuscations are indistinguishable to any PPT adversary. [BGI+01]

\[ P_0(a, b): \]
- \( \text{asq} = a^2 \)
- \( \text{bsq} = b^2 \)
- Output \( \text{asq-bsq} \)

\[ P_1(a, b): \]
- \( \text{dif} = a-b \)
- \( \text{sum} = a+b \)
- Output \( \text{dif*sum} \)
**Indistinguishability Obfuscation**

**Security:** Given two equivalent programs of the same length, their obfuscations are indistinguishable to any PPT adversary. [BGI⁺01]

\[ P_0(a,b): \]
\[ \text{asq} = a^2 \]
\[ \text{bsq} = b^2 \]
\[ \text{Output } \text{asq} - \text{bsq} \]

\[ P_1(a,b): \]
\[ \text{dif} = a - b \]
\[ \text{sum} = a + b \]
\[ \text{Output } \text{dif} \times \text{sum} \]

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

\[ \text{Obf}(P_b) \]

\[ b' \]
**Indistinguishability Obfuscation**

**Security:** Given two equivalent programs of the same length, their obfuscations are indistinguishable to any PPT adversary. [BGI⁺01]

P₀(a,b):
- asq = a^2
- bsq = b^2
- Output asq-bsq

P₁(a,b):
- dif = a-b
- sum = a+b
- Output dif*sum

For any PPT hacker, Pr[b' = b] ≤ ½ + negl
Public Key Encryption
Digital Signatures
Broadcast Encryption
Identity-Based Encryption

Fully Homomorphic Encryption
Order Revealing Encryption
Multiparty Key Agreement

(and much more)
Public Key Encryption
Fully Homomorphic Encryption
Digital Signatures
Order Revealing Encryption
Broadcast Encryption
Multiparty Key Agreement
Identity-Based Encryption
(and much more)
iO → Multiparty Key Agreement (NIKE) [BZ14]
iO → Multiparty Key Agreement (NIKE) [BZ14]
iO → Multiparty Key Agreement (NIKE) [BZ14]

\[ x_1 = \text{PRG}(s_1) \]
\[ x_2 = \text{PRG}(s_2) \]
\[ x_3 = \text{PRG}(s_3) \]
\[ x_4 = \text{PRG}(s_4) \]
iO → Multiparty Key Agreement (NIKE) [BZ14]

\[ x_1 = \text{PRG}(s_1) \]
\[ x_2 = \text{PRG}(s_2) \]
\[ x_3 = \text{PRG}(s_3) \]
\[ x_4 = \text{PRG}(s_4) \]

\[ \text{iO}(P) \]
iO → Multiparty Key Agreement (NIKE) [BZ14]

\[
P(i,s_i,x_1,x_2,x_3,x_4):
\begin{align*}
\text{if } x_i &\neq \text{PRG}(s_i) : \\
&\quad \text{output } \bot \\
\text{else: } \\
&\quad \text{output } \text{PRF}(x_1,x_2,x_3,x_4)
\end{align*}
\]
iO → Multiparty Key Agreement (NIKE) [BZ14]

\[ \text{iO}(P) \]

\[ x_1 = \text{PRG}(s_1) \]
\[ x_2 = \text{PRG}(s_2) \]
\[ x_3 = \text{PRG}(s_3) \]
\[ x_4 = \text{PRG}(s_4) \]

\[ P(i, s_i, x_1, x_2, x_3, x_4) : \]
\[ \text{if } x_i \neq \text{PRG}(s_i) : \]
\[ \quad \text{output } \bot \]
\[ \text{else:} \]
\[ \quad \text{output } \text{PRF}(x_1, x_2, x_3, x_4) \]

Person i computes \( \text{iO}(P)(i, s_i, x_1, x_2, x_3, x_4) \) to get the key.
Why not use iO everywhere?

- Very cryptographically useful ✓
- We know how to build it (multilinear maps) ✓
Why not use iO everywhere?

- Very cryptographically useful ✓
- We know how to build it (multilinear maps) ✓

...and

- completely impractical X
- security is not fully understood X
Building iO

A multilinear map $e: G_1, G_2, \ldots, G_k \rightarrow G_t$ on $k+1$ cyclic groups satisfies

$$e(g_1, \ldots, g_i^\alpha, \ldots, g_k) = e(g_1, \ldots, g_i, \ldots, g_k)^\alpha$$

- Encode plaintexts as group elements
- **High level idea**: Transform input circuit into a matrix branching program. Evaluate the program using the multilinear map and group operations.
- Recovering program requires solving discrete logs.
Brief History

• 2000: Multilinear maps $\rightarrow$ multiparty Diffie-Hellman [BS00]
• 2013: First constructions of multilinear maps [GGH13a, CLT13]
• 2013: First construction of iO from multilinear maps [GGH+13b]
• 2014: CLT13 maps completely broken in some settings by zeroizing attacks [CHL+14]
• 2016: Annihilation attacks break many iO schemes based on GGH13 [MSZ16]
New Tool: Encryptor Combiners

- Weaker Objects
- Multiparty NIKE
- Identity Based Encryption
- Broadcast Encryption
- (and more)
Outline

1. Define Encryptor Combiners
2. Identity Based Encryption from Encryptor Combiners
3. Encryptor Combiner Construction from Universal Samplers
4. Multiparty NIKE from Encryptor Combiners
5. Encryptor Combiners for Dual Regev Encryption
Outline

1. Define Encryptor Combiners
2. Identity Based Encryption from Encryptor Combiners
3. Encryptor Combiner Construction from Universal Samplers
4. Multiparty NIKE from Encryptor Combiners
5. Encryptor Combiners for Dual Regev Encryption
Terminology

• An encryptor is a randomized algorithm $E(m) \rightarrow c$

• A decryptor is a deterministic algorithm $D(c) \rightarrow m$

$D$ is **valid** for $E$ if for all $m$,

$$\Pr[D(E(m)) = m] > 1 - \text{negl}$$
Terminology

• An encryptor is a randomized algorithm $E(m) \rightarrow c$
• A decryptor is a deterministic algorithm $D(c) \rightarrow m$

$D$ is **valid** for $E$ if for all $m$,
\[
\Pr[D(E(m)) = m] > 1 - \text{negl}
\]

**Example:** In public key encryption, $\text{Gen}() \rightarrow (pk,sk)$
\[
E(m) = \text{Enc}(pk,m), \ D(c) = \text{Dec}(sk,c)
\]
Encryptor Combiner Definition

An encryptor combiner consists of:

• Setup(): Output \textit{params}
• Combine(\textit{params}, E_1, \ldots, E_n): Output encryptor \textit{E}\textsuperscript{*}
• Extract(\textit{params}, E_1, \ldots, E_n, i, D_i): Output decryptor \textit{D}\textsuperscript{*}
Encryptor Combiner Definition

An encryptor combiner consists of:

- **Setup():** Output *params*
- **Combine**(*params*, E₁,..., Eₙ): Output encryptor E*
- **Extract**(*params*, E₁,..., Eₙ, i, Dᵢ): Output decryptor D*

Key point: One decryptor is sufficient to compute D*
Combine

\[ E_1, E_2, E_3, \ldots, E_n \]

Extract

\[ E_1, E_2, E_3, \ldots, E_n \]

Setup

\[ params \]

\[ \text{i,}_D_i \]

\[ E^* \]

\[ D^* \]
Encryptor Combiner Definition

**Correctness:** If $D_i$ is valid for $E_i$, and

$$E^* \leftarrow \text{Combine}(\text{params},E_1,\ldots,E_n)$$

$$D^* \leftarrow \text{Extract}(\text{params},E_1,\ldots,E_n,i,D_i)$$

$D^*$ must be valid for $E^*$. 
Encryptor Combiner Definition

**Correctness**: If $D_i$ is valid for $E_i$, and

$$E^* \leftarrow \text{Combine}(\text{params}, E_1, \ldots, E_n)$$

$$D^* \leftarrow \text{Extract}(\text{params}, E_1, \ldots, E_n, i, D_i)$$

$D^*$ must be valid for $E^*$.

**Security**: If an adversary can decrypt $E^*$, it can decrypt $E_i$ for some $i$.  

41
Encryptor Combiner Properties

Perfect Independence: (deterministic) For all i, j,

\[ \text{Extract}(\text{params}, E_1, \ldots, E_n, i, D_i) = \text{Extract}(\text{params}, E_1, \ldots, E_n, j, D_j) \]
Encryptor Combiner Properties

**Perfect Independence:** (deterministic) For all i,j,

\[
\text{Extract}(\text{params}, E_1, \ldots, E_n, i, D_i) = \text{Extract}(\text{params}, E_1, \ldots, E_n, j, D_j)
\]

**Distributional Independence:** (randomized) For all i,j,

\[
\text{Dist}(D_i^*) \approx \text{Dist}(D_j^*)
\]

\[
D_i^* \leftarrow \text{Extract}(\text{params}, E_1, \ldots, E_n, i, D_i)
\]

\[
D_j^* \leftarrow \text{Extract}(\text{params}, E_1, \ldots, E_n, j, D_j)
\]
Encryptor Combiner Properties

Ciphertext Compactness: Ciphertexts produced by E* don’t grow with number of encryptors.
Encryptor Combiner Properties

**Example:** Consider a “trivial” encryptor combiner

\[ E^*(m) = E_1(m) \ || \ E_2(m) \ || \ ... \ || \ E_n(m) \]

\[ D_i^*(m): \text{Given } D_i, \text{ decrypt } E_i(m) \]
Encryptor Combiner Properties

**Example:** Consider a “trivial” encryptor combiner

\[ E^*(m) = E_1(m) \parallel E_2(m) \parallel \ldots \parallel E_n(m) \]

\[ D_i^*(m): \text{Given } D_i, \text{ decrypt } E_i(m) \]

- Not perfect / distributionally independent
- Ciphertext size grows with n
Outline

1. Define Encryptor Combiners ✓
2. Identity Based Encryption from Encryptor Combiners
3. Encryptor Combiner Construction from Universal Samplers
4. Multiparty NIKE from Encryptor Combiners
5. Encryptor Combiners for Dual Regev Encryption
Application: Identity-Based Encryption

Setup() → (mpk,msk)
Application: Identity-Based Encryption

\[ \text{Setup()} \rightarrow (\text{mpk}, \text{msk}) \]

\[ E_{\text{Bob}} \leftarrow \text{EncGen}(\text{mpk}, \text{“Bob@pton.edu”}) \]
Application: Identity-Based Encryption

Setup() → (mpk,msk)

\[ E_{\text{Bob}} \leftarrow \text{EncGen}(\text{mpk},"\text{Bob@pton.edu}")) \]
Application: Identity-Based Encryption

Setup() → (mpk, msk)

$E_{Bob} \leftarrow \text{EncGen}(mpk, \text{“Bob@pton.edu”})$

$D_{Bob} \leftarrow \text{Extract}(msk, \text{“Bob@pton.edu”})$

$E_{Bob}(m)$

Bob authenticates

$D_{Bob}$
Application: Identity-Based Encryption

Setup() → (mpk, msk)

\[ E_{Bob} \leftarrow \text{EncGen}(mpk, \text{“Bob@pton.edu”}) \]

\[ D_{Bob} \leftarrow \text{Extract}(msk, \text{“Bob@pton.edu”}) \]

Bob authenticates

\[ E_{Bob}(m) \rightarrow \quad \text{Bob recovers } m = D_{Bob}(E_{Bob}(m)) \]
Application: Identity-Based Encryption

Security?
Application: Identity-Based Encryption

Security?

$E_{\text{Bob}} \leftarrow \text{EncGen}(\text{mpk,} \text{“Bob@pton.edu”})$
Application: Identity-Based Encryption

Security?

$E_{Bob} \leftarrow \text{EncGen}(\text{mpk}, \text{"Bob@pton.edu"})$

$E_{Bob}(m)$

Other users cannot decrypt $E_{Bob}$ even if they collude!
Theorem: Can build Identity-Based Encryption from Encryptor Combiners + Public Key Encryption + Collision-Resistant Hashing
Application: Identity-Based Encryption

ID space = \{0, 1\}^4  
mpk = \begin{array}{cccc}
E_{1,0} & E_{2,0} & E_{3,0} & E_{4,0} \\
E_{1,1} & E_{2,1} & E_{3,1} & E_{4,1}
\end{array}  
msk = \begin{array}{c}
D_{1,0} \\
D_{1,1}
\end{array}
Application: Identity-Based Encryption

ID space = \{0,1\}^4 \hspace{1cm} mpk = \begin{array}{cccc}
E_{1,0} & E_{2,0} & E_{3,0} & E_{4,0} \\
E_{1,1} & E_{2,1} & E_{3,1} & E_{4,1} \\
\end{array} \hspace{1cm} msk = \begin{array}{c}
D_{1,0} \\
D_{1,1} \\
\end{array}

\text{IBE.EncGen}(mpk, \text{id} = \text{"1101"}): E^* \leftarrow \text{EC.Combine}(E_{1,1}, E_{2,1}, E_{3,0}, E_{4,1})

\text{IBE.Extract}(msk, \text{id} = \text{"1101"}): D^* \leftarrow \text{EC.Extract}(E_{1,1}, E_{2,1}, E_{3,0}, E_{4,1}, 1, D_{1,1})
Application: Identity-Based Encryption

ID space = \{0,1\}^4 \quad mpk = \begin{array}{cccc}
E_{1,0} & E_{2,0} & E_{3,0} & E_{4,0} \\
E_{1,1} & E_{2,1} & E_{3,1} & E_{4,1}
\end{array} \quad msk = \begin{array}{c}
D_{1,0} \\
D_{1,1}
\end{array}

IBE.EncGen(mpk, id = “1101”): E^* \leftarrow EC.Combine(E_{1,1}, E_{2,1}, E_{3,0}, E_{4,1})

IBE.Extract(msk, id = “1101”): D^* \leftarrow EC.Extract(E_{1,1}, E_{2,1}, E_{3,0}, E_{4,1}, 1, D_{1,1})

Intuition: If A can break security of E^* for id = 1101, we can break security of encryptor combiner on E_{1,1}, E_{2,1}, E_{3,0}, E_{4,1}.
Outline

1. Define Encryptor Combiners ✓
2. Identity Based Encryption from Encryptor Combiners ✓
3. Encryptor Combiner Construction from Universal Samplers
4. Multiparty NIKE from Encryptor Combiners
5. Encryptor Combiners for Dual Regev Encryption
Universal Samplers

Randomness $r$

$C$

$C(r)$

Description of $C$

Universal Sampler

$C(d)$

(deterministic!)
Universal Samplers

Randomness $r$

$C(r)$

$C$ (Universal Sampler)

Description of $C$

$C(d)$

*Adversary does not get to make any queries to the sampler
Universal Samplers

Can be built from indistinguishability obfuscation and one way functions. [HJK+14]
Universal Samplers

Formally, a universal sampler consists of:

• Setup(): Output \textit{params}

• Samp(params,C): Deterministically output p_C, a valid output from C.
Circuit $C[E_i, E_j](r)$:
Generate $E_{ij}, D_{ij}$ from PKE with randomness $r$.
Output $E_{ij}, E_i(D_{ij}), E_j(D_{ij})$
Encryptor Combiners from Universal Samplers

**Circuit C[E_i,E_j](r):**
Generate $E_{ij}, D_{ij}$ from PKE with randomness $r$.
Output $E_{ij}, E_i(D_{ij}), E_j(D_{ij})$

Samp($C[E_i,E_j]$) outputs $E_{ij}, E_i(D_{ij}), E_j(D_{ij})$ for “fresh” $E_{ij}, D_{ij}$
Encryptor Combiners from Universal Samplers

**Circuit C[E_i,E_j](r):**
Generate $E_{ij},D_{ij}$ from PKE with randomness $r$.
Output $E_{ij},E_i(D_{ij}),E_j(D_{ij})$

Samp(C[E_i,E_j]) outputs $E_{ij},E_i(D_{ij}),E_j(D_{ij})$ for “fresh” $E_{ij},D_{ij}$
Encryptor Combiners from Universal Samplers

**Circuit $C[E_i, E_j](r)$:**
Generate $E_{ij}, D_{ij}$ from PKE with randomness $r$.
Output $E_{ij}, E_i(D_{ij}), E_j(D_{ij})$

Samp($C[E_i, E_j]$) outputs $E_{ij}, E_i(D_{ij}), E_j(D_{ij})$ for “fresh” $E_{ij}, D_{ij}$

Given $D_1$ or $D_2$, run Samp($C[E_1, E_2]$) to get $E_{12}, E_1(D_{12}), E_2(D_{12})$. Decrypt to recover $D_{12}$. 
Encryptor Combiners from Universal Samplers

**Circuit C[E_i,E_j](r):**
Generate $E_{ij}, D_{ij}$ from PKE with randomness $r$.
Output $E_{ij}, E_i(D_{ij}), E_j(D_{ij})$

Samp(C[E_i,E_j]) outputs $E_{ij}, E_i(D_{ij}), E_j(D_{ij})$ for “fresh” $E_{ij}, D_{ij}$

$$E^* = E_{1234}$$
$$D^* = D_{1234}$$
Outline

1. Define Encryptor Combiners ✓
2. Identity Based Encryption from Encryptor Combiners ✓
3. Encryptor Combiner Construction from Universal Samplers ✓
4. Multiparty NIKE from Encryptor Combiners
5. Encryptor Combiners for Dual Regev Encryption
Application: Multiparty NIKE

**Theorem:** Can build Multiparty Non-Interactive Key Exchange from Encryptor Combiners*.

*but we need a new definition!
Multiparty NIKE from Encryptor Combiners
Multiparty NIKE from Encryptor Combiners

\[
\text{params} \leftarrow \text{EC.Setup()}
\]
Multiparty NIKE from Encryptor Combiners

User $i$ generates $(pk_i, sk_i) \leftarrow PKE.Gen()$

Setup() generates $\text{params} \leftarrow \text{EC.Setup()}$
Multiparty NIKE from Encryptor Combiners

Setup() generates
params ← EC.Setup()
Multiparty NIKE from Encryptor Combiners

```latex
\text{Setup()} \text{ generates } \text{params} \leftarrow \text{EC.Setup()}
```
Multiparty NIKE from Private ECs and Public Key Encryption

Define $E_i = \text{Enc}(\text{pk}_i, \cdot)$ and $D_i = \text{Dec}(\text{sk}_i, \cdot)$.

User $i$ computes $D^* \leftarrow \text{EC.Extract}(\text{params}, E_1, E_2, E_3, E_4, i, D_i)$

Interpret $D^*$ as string of bits, use this as shared key.
Multiparty NIKE from Private ECs and Public Key Encryption

Define $E_i = \text{Enc}(pk_i, \cdot)$ and $D_i = \text{Dec}(sk_i, \cdot)$.

User $i$ computes $D^* \leftarrow \text{EC.Extract(params,} E_1, E_2, E_3, E_4, i, D_i\text{)}$

Interpret $D^*$ as string of bits, use this as shared key.

Problem: $D^*$ might not look pseudorandom
“Public” Encryptor Combiners

\[ E_1, E_2, E_3, \ldots, E_n \]

\[ \rightarrow \]

Setup

\[ \rightarrow \]

params

\[ E_1, E_2, E_3, \ldots, E_n \]

\[ \rightarrow \]

Combine

\[ E^* \]

Extract

\[ D^* \]

\[ \rightarrow \]

\[ i, D_i \]
Defining Private Encryptor Combiners

Setup

defs

params

E₁ E₂ E₃ … Eₙ

Combine

E*  

Extract

D*

i, Dᵢ
Defining Private Encryptor Combiners

![Diagram showing the process of defining private encryptor combinators]

Setup

params

$E_1 \quad E_2 \quad E_3 \quad \ldots \quad E_n$

$i, D_i$

Extract

Output key K

$D^*$
Defining Private Encryptor Combiners
Defining Private Encryptor Combiners

**Correctness:** For any pair of indices $j, k$, with probability $1 - \negl$, Extract($\text{params}, E_1, \ldots, E_n, j, D_j$) and Extract($\text{params}, E_1, \ldots, E_n, k, D_k$) generate the same key.
Defining Private Encryptor Combiners

**Correctness:** For any pair of indices $j, k$, with probability $1 - \text{negl}$, \( \text{Extract}(\text{params}, E_1, \ldots, E_n, j, D_j) \) and \( \text{Extract}(\text{params}, E_1, \ldots, E_n, k, D_k) \) generate the same key.

**Security:** (Intuition) An adversary who does not know any $D_i$ cannot distinguish $K$ from random $K$ drawn from the keyspace.
Building Private Combiners From Public Combiners

Pub-Setup \(\rightarrow\) pub-params

Pub-Extract

\(E_1, E_2, E_3, \ldots, E_n\) \(\downarrow\)

\(D^*\)

\(i, D_i\) \(\rightarrow\) pub-params
Building Private Combiners From Public Combiners

Priv-Setup

Pub-Setup → pub-params

Random bit-string R. Output (R, pub-params)

(R, pub-params)

pub-params

Pub-Extract

E₁ → E₂ → E₃ → ... → Eₙ

i, Dᵢ

D*
Building Private Combiners From Public Combiners

Priv-Setup

Pub-Setup → pub-params

Random bit-string \( R \). Output \((R, \text{pub-params})\)

Priv-Extract

\((R, \text{pub-params})\) → pub-params

\( i, D_i \) → pub-params

\( i, D_i \) → pub-params

Pub-Extract

\( D^* \)

Interpret \( D^* \) as bit-string.
Output \( b = < R, D^* > \)

\( E_1 \)

\( E_2 \)

\( E_3 \)

\( \ldots \)

\( E_n \)

b
**Building Private Combiners From Public Combiners**

- **Pub-Setup** → pub-params
- **Priv-Setup** → pub-params
- Random bit-string \( R \). Output \((R, \text{pub-params})\)

\[(R, \text{pub-params}) \rightarrow \text{pub-params}\]

\[i, D_i \rightarrow \text{pub-params}\]

\[\text{Priv-Extract}\]

Interpret \( D^* \) as bit-string.
Output \( b = < R, D^* > \)

**Security:** \(< R, D^* >\) is a Goldreich-Levin hardcore bit that is pseudorandom for unpredictable source \( D^* \).
Do we get an $m$-bit combiner if we run $m$ 1-bit combiners in parallel?

$$\{E_1, E_2, \ldots, E_n\}$$
Security: Want to show $b_1 b_2 b_3 \ldots b_m$ is indistinguishable from $m$ random bits.

$\{E_1, E_2, \ldots, E_n\}$
**Security:** Want to show $b_1 b_2 b_3 \ldots b_m$ is indistinguishable from $m$ random bits.

**Proof Attempt:** At step $i$, replace bit $b_i$ with a random bit.

$$\{E_1, E_2, \ldots, E_n\}$$
Security: Want to show $b_1 b_2 b_3 ... b_m$ is indistinguishable from $m$ random bits.

Proof Attempt: At step $i$, replace bit $b_i$ with a random bit.
Security: Want to show $b_1 b_2 b_3 \ldots b_m$ is indistinguishable from $m$ random bits.

Proof Attempt: At step $i$, replace bit $b_i$ with a random bit.
Security: Want to show $b_1b_2b_3...b_m$ is indistinguishable from $m$ random bits.

Proof Attempt: At step $i$, replace bit $b_i$ with a random bit.
**Security:** Want to show \( b_1b_2b_3...b_m \) is indistinguishable from \( m \) random bits.

Proof Attempt: At step \( i \), replace bit \( b_i \) with a random bit.

\( \{E_1, E_2, ..., E_n\} \)
**Security**: Want to show $b_1b_2b_3\ldots b_m$ is indistinguishable from $m$ random bits.

**Proof Attempt**: At step $i$, replace bit $b_i$ with a random bit.
**Security:** Want to show $b_1b_2b_3...b_m$ is indistinguishable from $m$ random bits.

**Proof Attempt:** At step $i$, replace bit $b_i$ with a random bit.

Want to show that if an adversary can distinguish step $i$ from step $i-1$ for some $i$, adversary breaks security of the 1-bit combiner.
**Security:** Want to show 
\(b_1 b_2 b_3 \ldots b_m\) is indistinguishable from \(m\) random bits.

**Proof Attempt:** At step \(i\), replace bit \(b_i\) with a random bit.

Want to show that if an adversary can distinguish step \(i\) from step \(i-1\) for some \(i\), adversary breaks security of the 1-bit combiner.

But this won’t work; reduction has no way to generate the honest bits without knowing a \(D_i\).
Security: Want to show
\[ b_1 b_2 b_3 \ldots b_m \] is indistinguishable from \( m \) random bits.

Proof Attempt: At step \( i \), replace bit \( b_i \) with a random bit.

Want to show that if an adversary can distinguish step \( i \) from step \( i-1 \) for some \( i \), adversary breaks security of the 1-bit combiner.

But this won’t work; reduction has no way to generate the honest bits without knowing a \( D_i \).

Solution: Build a 1-bit combiner with a trapdoor!
Trapdoored Encryptor Combiners

**Theorem:** Can build Trapdoored Encryptor Combiners from Public Key Encryption + Encryptor Combiners
Trapdoored Encryptor Combiners

- Modify Setup() to also output an encryptor/decryptor pair $E,D$, where $D$ is the trapdoor. $E$ becomes part of $\text{params}$.
- Whenever we extract on $E_1,...,E_n$, we also include $E$.
- Thus, trapdoor $D$ can always be used to compute a combined decryptor $D^*$. 
Setup $\rightarrow$ E,D. (E is public, D is trapdoor)

params

$E_1 \rightarrow E_2 \rightarrow \ldots \rightarrow E_n \rightarrow E$

E,D. $\rightarrow$ Extract $\rightarrow$ K

i,D_i $\rightarrow$ E,D. (E is public, D is trapdoor)
Setup → E,D. (E is public, D is trapdoor)

params → E₁, E₂, ..., Eₙ, E

Extract → K

i, Dᵢ → E₁, E₂, ..., Eₙ, E

Trapdoor-Extract → K

D →
Recap

Public EC

1-bit Private EC

Trapdoored 1-bit Private EC

m-bit Private EC

Multiparty NIKE

Public Key Encryption
Outline

1. Define Encryptor Combiners ✓
2. Identity Based Encryption from Encryptor Combiners ✓
3. Encryptor Combiner Construction from Universal Samplers ✓
4. Multiparty NIKE from Encryptor Combiners ✓
5. Encryptor Combiners for Dual Regev Encryption
Dual Regev Encryption

All integers mod q
T is “short” full-rank matrix
over the rationals satisfying
A·T = 0 mod q
Dual Regev Encryption

Encryption

To encrypt $b = 1$, output a random $1 \times m$ vector $c$: $c = \ldots$

To encrypt $b = 0$, pick a random $n \times 1$ vector $s$ and a random $m \times 1$ “short” vector $e$. Output:

All integers mod $q$

$T$ is “short” full-rank matrix over the rationals satisfying $A \cdot T = 0 \mod q$
Dual Regev Encryption

Encryption

To encrypt $b = 1$, output a random $1 \times m$ vector $c$:

To encrypt $b = 0$, pick a random $n \times 1$ vector $s$ and a random $m \times 1$ "short" vector $e$. Output:

Learning With Errors Assumption:
Hard for PPT adversary to distinguish between these ciphertexts.
Dual Regev Encryption

<Diagram>

 Encryption

To encrypt $b = 1$, output a random $1 \times m$ vector $c$:  

To encrypt $b = 0$, pick a random $n \times 1$ vector $s$ and a random $m \times 1$ "short" vector $e$. Output:

Decryption

<Diagram>

Output 0 iff all components small
Dual Regev Encryption

**Theorem:** There exists a Dual Regev Encryptor Combiner that is unbounded and has distributional independence.
Combining Dual Regev Encryptors

\[ A^* = \begin{array}{ccc}
A_1 & A_2 & \cdots & A_m \\
\end{array} \]

Given full rank “short” \( T_i \) such that \( A_i \cdot T_i = 0 \mod q \), we can compute corresponding \( T^* \) via known basis extension techniques [CHK+10, ABB10].
Combining Dual Regev Encryptors

\[ A^* = \begin{array}{c} A_1 \end{array} \begin{array}{c} A_2 \\ \vdots \end{array} \begin{array}{c} A_m \end{array} \]

Given full rank “short” \( T_i \) such that \( A_i \cdot T_i \equiv 0 \mod q \), we can compute corresponding \( T^* \) via known basis extension techniques [CHK+10, ABB10].

Unfortunately, procedure for generating \( T^* \) is randomized, so this cannot be used for Multiparty NIKE.

Resulting encryptor combiners can be used to build Identity-Based Encryption.
Thank you!

Questions?
Broadcast Encryption

(originally after IBE section)

Space of identities ID = \{“A”, “B”, “C”, …\}

NETFLIX

114
Broadcast Encryption

Space of identities ID = \{“A”, “B”, “C”, …\}

Setup(ID) → (mpk, msk)

NETFLIX
Broadcast Encryption

Space of identities ID = {“A”, “B”, “C”, …}

Setup(ID) → (mpk, msk)

Send D_{id} ← Extract(msk,id) to each user

NETFLIX

D_A

D_B

D_C

D_D

D_E

D_F
Broadcast Encryption

Space of identities $\text{ID} = \{\text{“A”, “B”, “C”, …}\}$

Setup($\text{ID}$) $\rightarrow$ (mpk,msk)

*NETFLIX* Encrypt to set $\text{S} = \{\text{“Alice”, “Charlie”, “Elsa”}\}$

\[ D_A \quad D_B \quad D_C \quad D_D \quad D_E \quad D_F \]
Broadcast Encryption

Space of identities ID = \{“A”, “B”, “C”, …\}

Setup(ID) → (mpk,msk)

Encrypt to set S = \{“Alice”, “Charlie”, “Elsa”\}

First, E ← EncGen(mpk,S).

Broadcast E(m) publicly
Broadcast Encryption

Space of identities ID = \{“A”, “B”, “C”, …\}

Setup(ID) → (mpk,msk)

Encrypt to set S = \{“Alice”, “Charlie”, “Elsa”\}

First, E ← EncGen(mpk,S).

Each user in S can compute

D ← DecGen(mpk,id,D_{id},S)

and recover D(E(m)) = m
Broadcast Encryption

**Security**: Users outside of S cannot decrypt a message intended for S, even if they collude.
BE from Encryptor Combiners and IBE

• Recall IBE generates a unique $E_{id}, D_{id}$ for each id.

• High level idea: To encrypt to a set of $S$, use EC.Combine to combine $E_{id}$ for each $id \in S$. To decrypt, any user with $id \in S$ can use their $D_{id}$ to compute $D^* \leftarrow EC.Extract$. 