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?

```
print "Hello World!"
```


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

Defining Obfuscation

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

```
print "Hello World!"
```

```
(lambda _, __, ___, ____, _____, ______, _______, ________:
    getattr(
        __import__(True.__class__.__name__[_] + [].__class__.__name__[__]),
        ().__class__.__eq__.__class__.__name__[:__] +
        ().__iter__().__class__.__name__[_____:________]
    )(
        _, (lambda _, __, ___: _(_, __, ___))(
            lambda _, __, ___:
                chr(___ % __) + _(_, __, ___ // __) if ___ else
                (lambda: _).func_code.co_lnotab,
            _ << ________,
            (((_____ << ____) + _) << ((___ << _____) - ___)) + (((((___ << __)
            - _) << ___) + _) << ((_____ << ____) + (_ << _))) + (((_______ <<
            __) - _) << (((((_ << ___) + _)) << ___) + (_ << _))) + (((_______
            << ___) + _) << ((_ << ______) + _)) + (((_______ << ____) - _) <<
            ((_______ << ___))) + (((_ << ____) - _) << ((((___ << __) + _) <<
            __) - _)) - (_______ << ((((___ << __) - _) << __) + _)) + (_______
            << (((((_ << ___) + _)) << __))) - ((((((_ << ___) + _)) << __) +
            _) << ((((___ << __) + _) << _))) + (((_______ << __) - _) <<
            (((((_ << ___) + _)) << _))) + (((___ << ___) + _) << ((_____ <<
            _))) + (_____ << ______) + (_ << ___)
        )
    )
)(
    *(lambda _, __, ___: _(_, __, ___))(
        (lambda _, __, ___:
            [__(___[(lambda: _).func_code.co_nlocals])] +
            _(_, __, ___[(lambda _: _).func_code.co_nlocals:]) if ___ else []
        ),
        lambda _: _.func_code.co_argcount,
        (
            lambda _: _,
            lambda _, __: _,
            lambda _, __, ___: _,
            lambda _, __, ___, ____: _,
            lambda _, __, ___, ____, _____: _,
            lambda _, __, ___, ____, _____, ______: _,
            lambda _, __, ___, ____, _____, ______, _______: _,
            lambda _, __, ___, ____, _____, ______, _______, ________: _
        )
    )
)
```

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.

```
(lambda _, __, ___, ____, _____, ______, _______, ________:
    getattr(
        __import__(True.__class__.__name__[_] + [].__class__.__name__[__]),
        ().__class__.__eq__.__class__.__name__[:__] +
        ().__iter__().__class__.__name__[_____:________]
    )(
        _, (lambda _, __, ___: _(_, __, ___))(
            lambda _, __, ___:
                chr(___ % __) + _(_, __, ___ // __) if ___ else
                (lambda: _).func_code.co_lnotab,
            _ << ________,
            (((_____ << ____) + _) << ((___ << _____) - ___)) + (((((___ << __)
            - _) << ___) + _) << ((_____ << ____) + (_ << _))) + (((_______ <<
            __) - _) << (((((_ << ___) + _)) << ___) + (_ << _))) + (((_______
            << ___) + _) << ((_ << ______) + _)) + (((_______ << ____) - _) <<
            ((_______ << ___))) + (((_ << ____) - _) << ((((___ << __) + _) <<
            __) - _)) - (_______ << ((((___ << __) - _) << __) + _)) + (_______
            << (((((_ << ___) + _)) << __))) - ((((((_ << ___) + _)) << __) +
            _) << ((((___ << __) + _) << _))) + (((_______ << __) - _) <<
            (((((_ << ___) + _)) << _))) + (((___ << ___) + _) << ((_____ <<
            _))) + (_____ << ______) + (_ << ___)
        )
    )
)(
    *(lambda _, __, ___: _(_, __, ___))(
        (lambda _, __, ___:
            [__(___[(lambda: _).func_code.co_nlocals])] +
            _(_, __, ___[(lambda _: _).func_code.co_nlocals:]) if ___ else []
        ),
        lambda _: _.func_code.co_argcount,
        (
            lambda _: _,
            lambda _, __: _,
            lambda _, __, ___: _,
            lambda _, __, ___, ____: _,
            lambda _, __, ___, ____, _____: _,
            lambda _, __, ___, ____, _____, ______: _,
            lambda _, __, ___, ____, _____, ______, _______: _,
            lambda _, __, ___, ____, _____, ______, _______, ________: _
        )
    )
)
```

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]

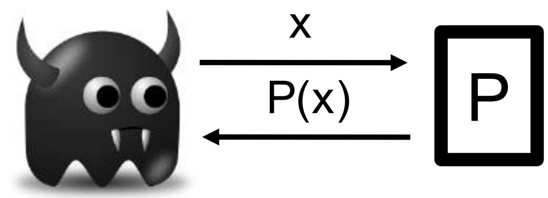
Observation: Given the obfuscation of a program P, we will can *always* learn its input/output behavior.

Thus, the goal of VBB security is to hide everything beyond what we can learn from running the program.

Virtual Black Box (VBB) Obfuscation [BGI+01]

PPT

PPT



Pr[ outputs 1] – Pr[ outputs 1] < 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]

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

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

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
```

$b \in \{0,1\}$

b'

Obf(P_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
```

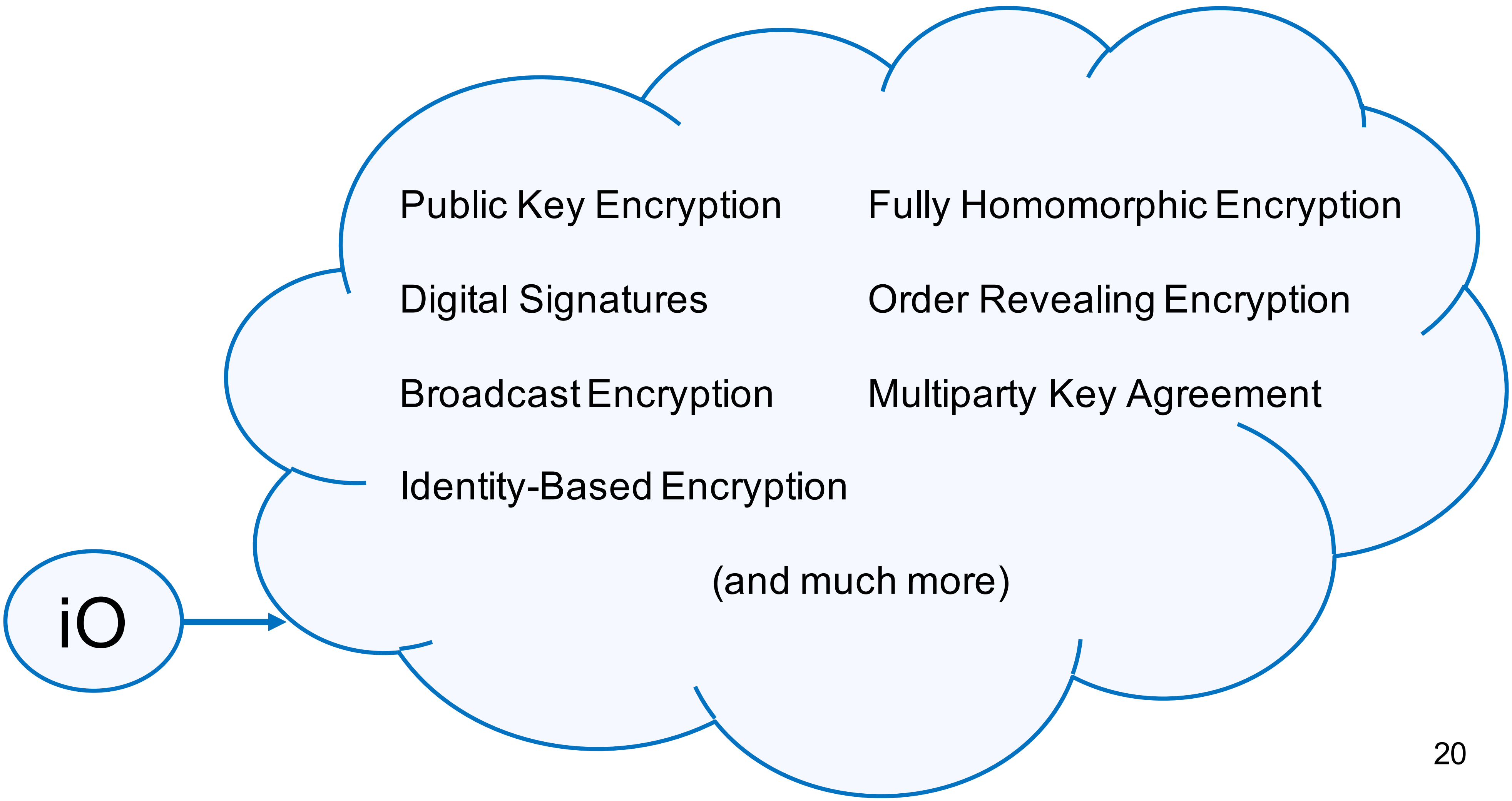
$b \in \{0,1\}$

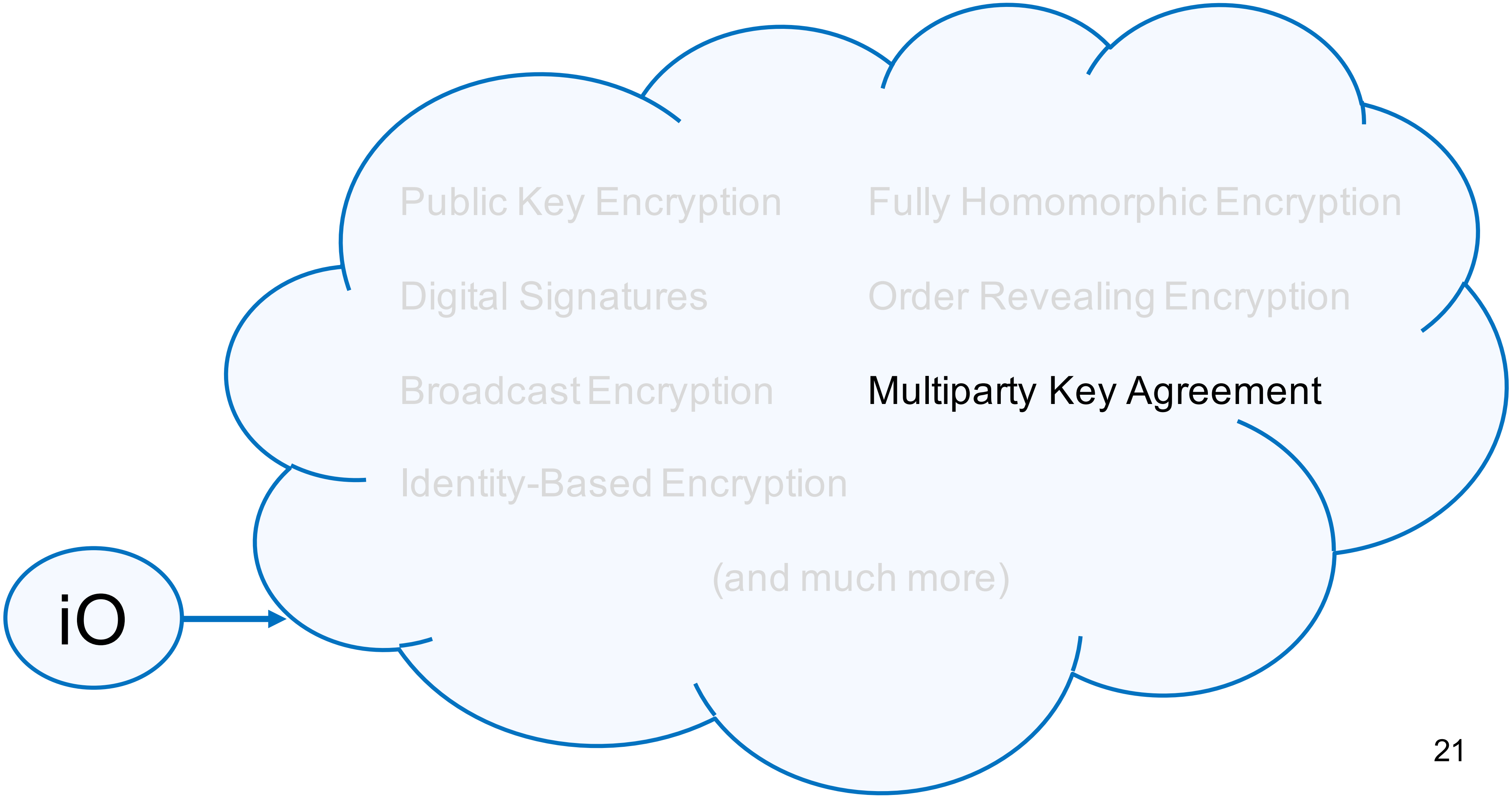
$\xleftarrow{Obf(P_b)}$

b’

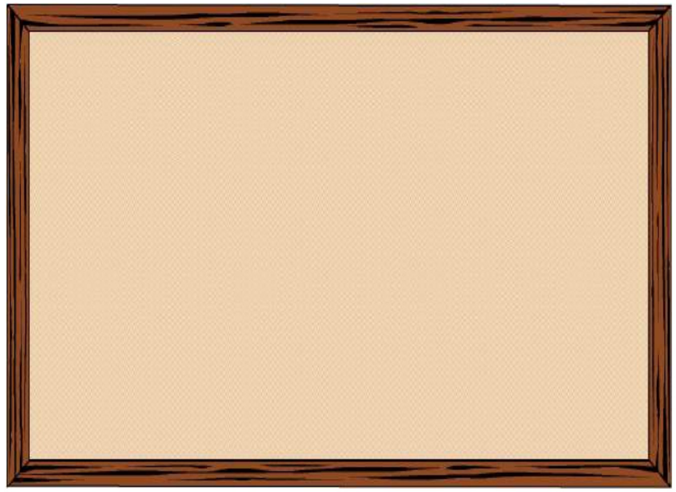
For any PPT ,

Pr[b’ = b] ≤ ½ + negl

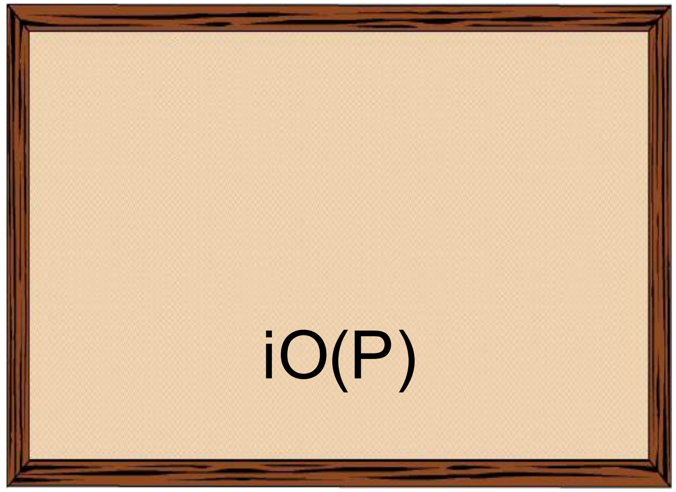

iO
Public Key Encryption
Fully Homomorphic Encryption
Digital Signatures
Order Revealing Encryption
Broadcast Encryption
Multiparty Key Agreement
Identity-Based Encryption
(and much more)


iO
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]



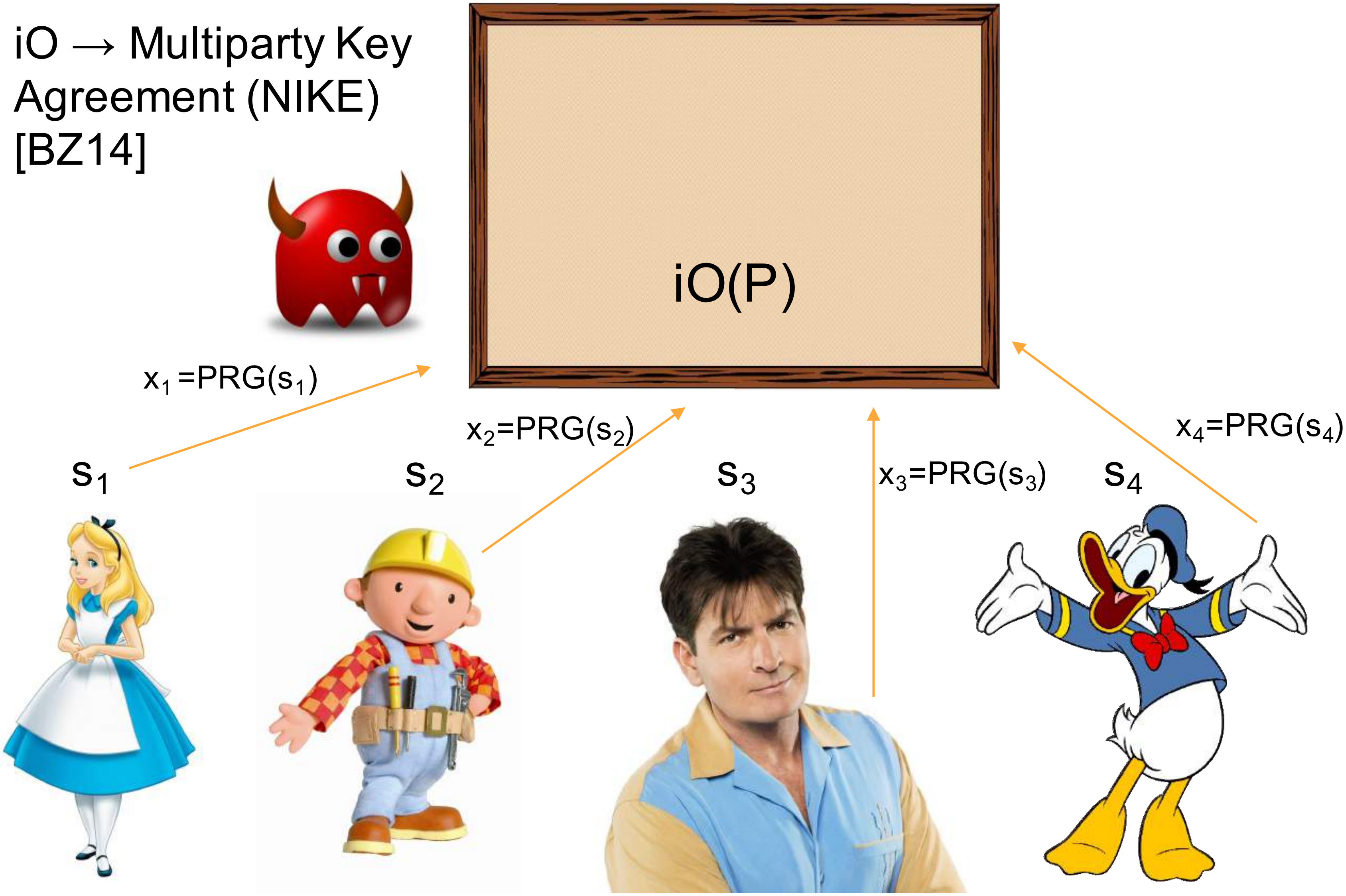
s_1

s_2

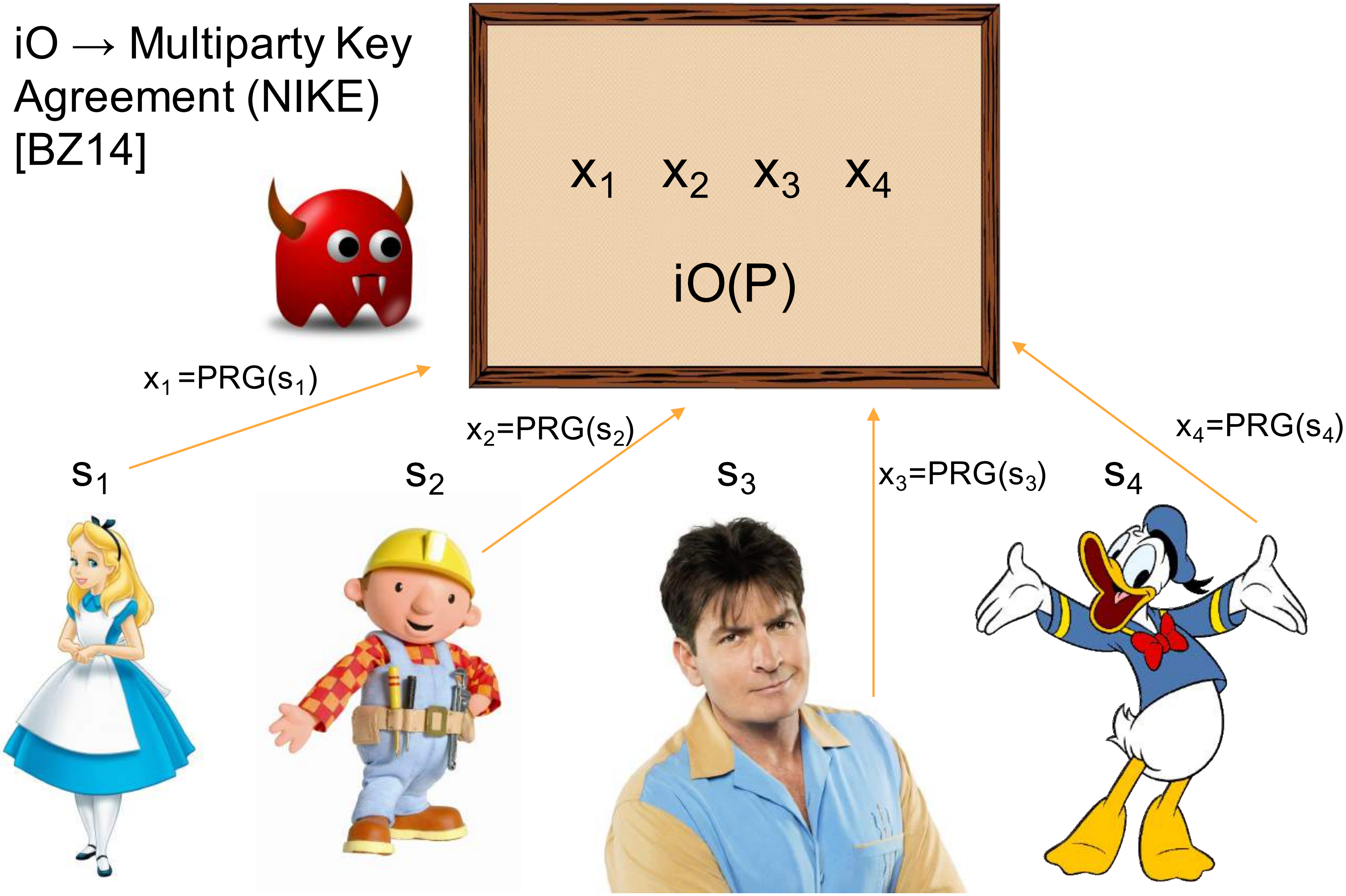
s_3

s_4

iO → Multiparty Key Agreement (NIKE) [BZ14]



iO → Multiparty Key Agreement (NIKE) [BZ14]



iO → Multiparty Key Agreement (NIKE) [BZ14]

x_1 x_2 x_3 x_4

iO(P)

```
P(i,s_i,x_1,x_2,x_3,x_4):
  if x_i ≠ PRG(s_i):
    output ⊥
  else:
    output PRF(x_1,x_2,x_3,x_4)
```

x_1 =PRG(s_1)

x_2=PRG(s_2)

x_3=PRG(s_3)

x_4=PRG(s_4)

s_1

s_2

s_3

s_4

iO → Multiparty Key Agreement (NIKE) [BZ14]

$x_1 \quad x_2 \quad x_3 \quad x_4$

iO(P)

P(i,s_i,x_1,x_2,x_3,x_4):
  if $x_i \neq$ PRG(s_i):
    output $\perp$
  else:
    output PRF(x_1,x_2,x_3,x_4)

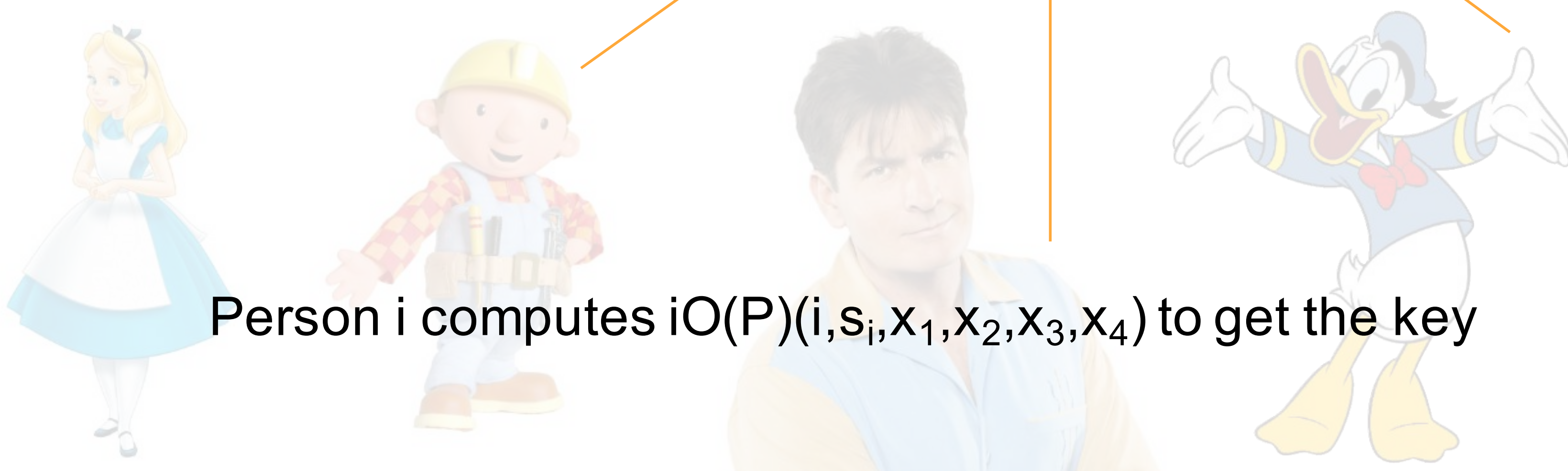
x_1 =PRG(s_1)

x_2=PRG(s_2)

x_3=PRG(s_3)

x_4=PRG(s_4)

$s_1 \quad s_2 \quad s_3 \quad s_4$



Person i computes 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

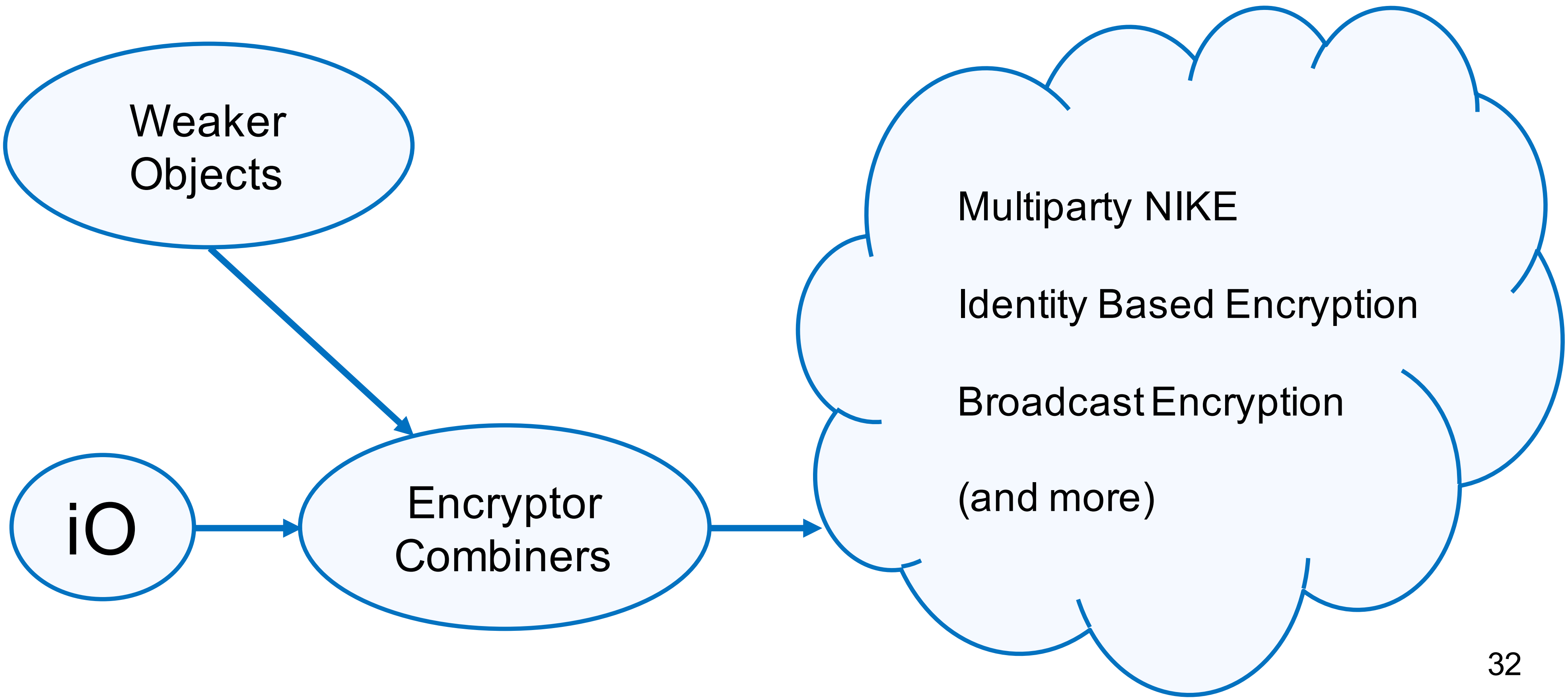
$$e(g_1, \ldots, g_i^{\alpha}, \ldots, g_k) = e(g_1, \ldots, g_i, \ldots, g_k)^{\alpha}$$

- 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 → 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



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 $E(m) \to c$
- A decryptor is a deterministic algorithm $D(c) \to m$

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

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

Example: In public key encryption, $\text{Gen}() \to (pk,sk)$

$E(m) = \text{Enc}(pk,m)$, $D(c) = \text{Dec}(sk,c)$

Encryptor Combiner Definition

An encryptor combiner consists of:

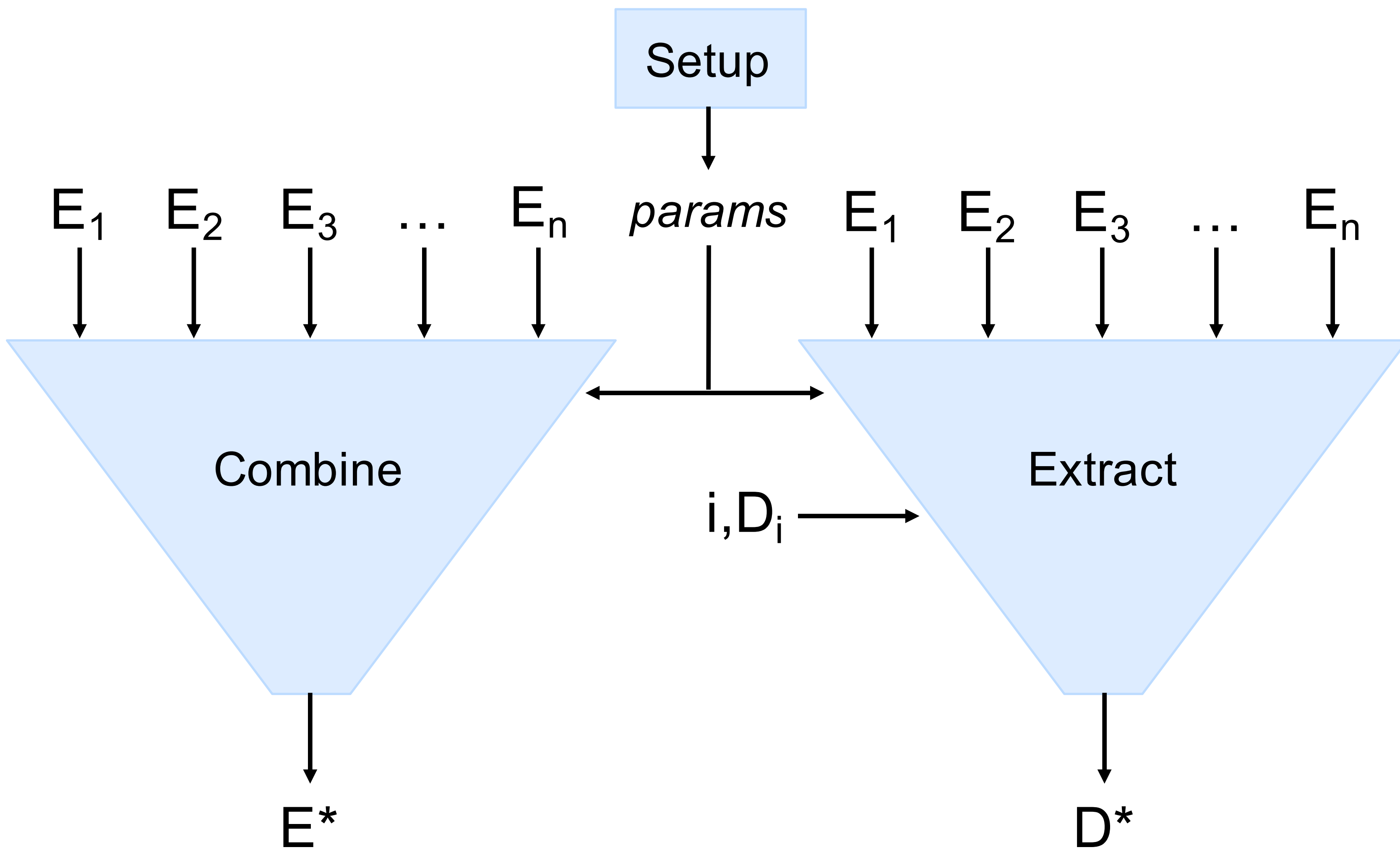
- Setup(): Output *params*
- Combine(*params*,$E_1,\ldots,E_n$): Output encryptor E*
- Extract(*params*, $E_1,\ldots,E_n$,i,D_i): Output decryptor D*

Encryptor Combiner Definition

An encryptor combiner consists of:

- Setup(): Output *params*
- Combine(*params*, $E_1,\ldots,E_n$): Output encryptor E*
- Extract(*params*, $E_1,\ldots,E_n,i,D_i$): Output decryptor D*

Key point: One decryptor is sufficient to compute D*


Setup
params
E1 E2 E3 … En
E1 E2 E3 … En
Combine
Extract
i,Di
E*
D*

Encryptor Combiner Definition

Correctness: If D_i is valid for $E_{i,}$ and

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

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

D* must be valid for E*.

Encryptor Combiner Definition

Correctness: If D_i is valid for E_i, and

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

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

D^* must be valid for E^*.

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

Encryptor Combiner Properties

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

Extract($params$,E_1,…,E_n,i,D_i) = Extract($params$,E_1,…,E_n,j,D_j)

Encryptor Combiner Properties

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

$$\mathrm{Extract}(\mathit{params}, E_1, \ldots, E_n, i, D_i) = \mathrm{Extract}(\mathit{params}, E_1, \ldots, E_n, j, D_j)$$

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

$$\mathrm{Dist}(D_i^*) \approx \mathrm{Dist}(D_j^*)$$

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

$D_j^* \leftarrow \mathrm{Extract}(\mathit{params}, E_1, \ldots, E_n, j, D_j)$

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

$$E^*(m) = E_1(m) \mid\mid E_2(m) \mid\mid \ldots \mid\mid E_n(m)$$

$D_i^*(m)$: Given D_i, decrypt $E_i(m)$

Encryptor Combiner Properties

Example: Consider a “trivial” encryptor combiner

$$E^*(m) = E_1(m) \,||\, E_2(m) \,||\, \ldots \,||\, E_n(m)$$

$D_i^*(m)$: Given D_i, 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

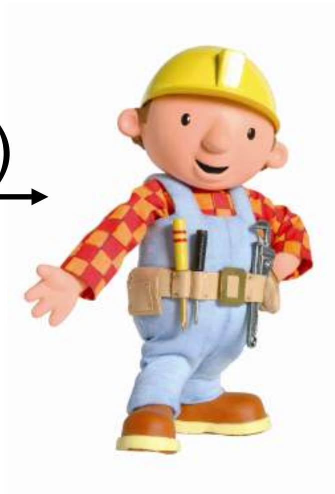
Setup() → (mpk,msk)

Application: Identity-Based Encryption

Setup() → (mpk,msk)

E_{Bob} ← EncGen(mpk,“Bob@pton.edu”)

$\xrightarrow{E_{Bob}(m)}$



Application: Identity-Based Encryption

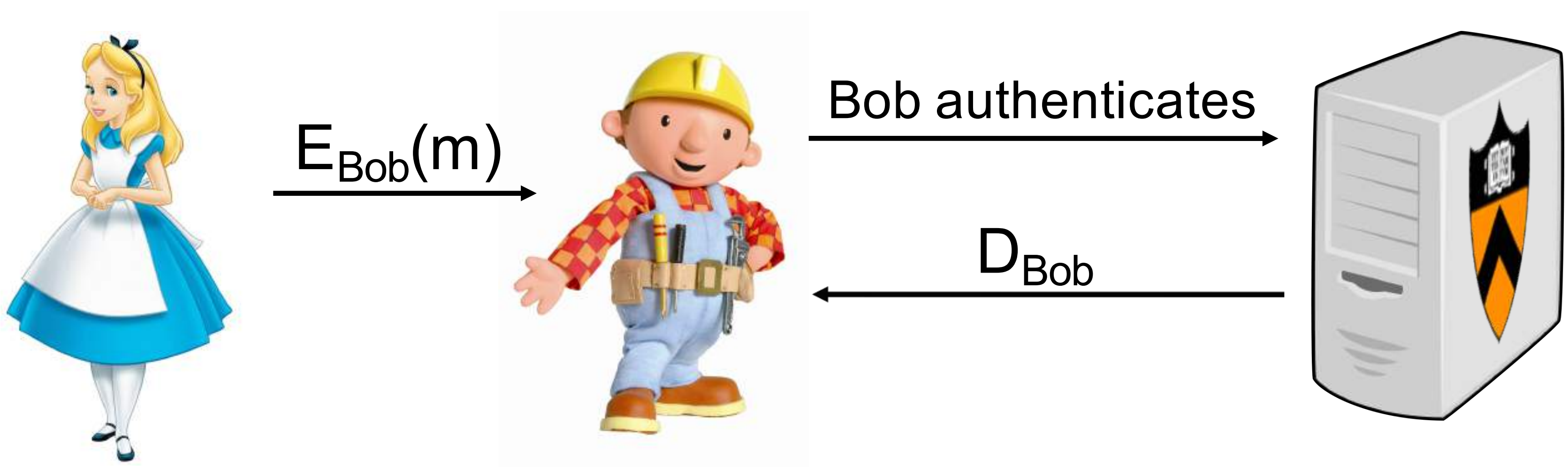
Setup() → (mpk,msk)

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

Application: Identity-Based Encryption

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

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



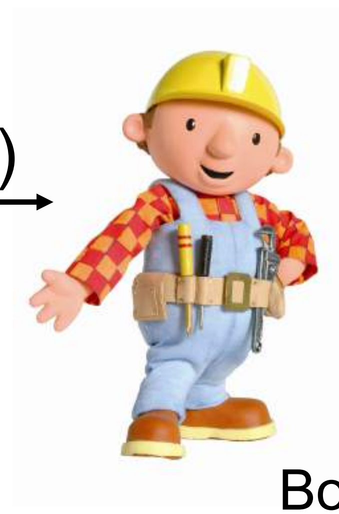
Application: Identity-Based Encryption

Setup() $\rightarrow$ (mpk,msk)

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

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

$E_{Bob}(m)$



Bob authenticates

D_{Bob}

Bob recovers $m = D_{Bob}(E_{Bob}(m))$

Application: Identity-Based Encryption

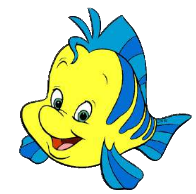
Security?





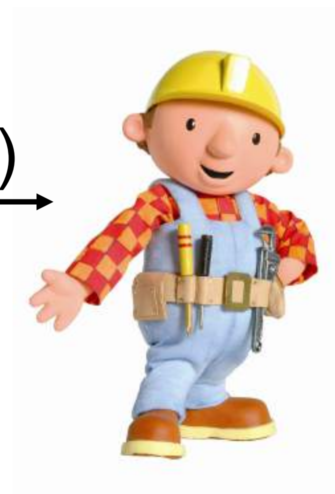

Application: Identity-Based Encryption

Security?

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

$E_{Bob}(m)$



Application: Identity-Based Encryption

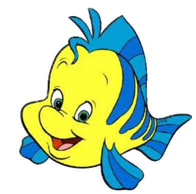
Security?

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

$D_{Charlie}$

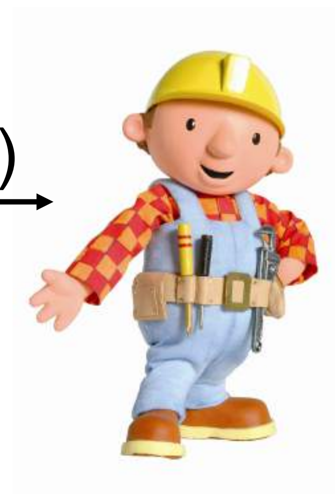
D_{Donald}

D_{Elsa}



$D_{Flounder}$

$\xrightarrow{E_{Bob}(m)}$



Other users cannot decrypt E_{Bob} even if they collude!

Application: Identity-Based Encryption

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 =

$E_{1,0}$	$E_{2,0}$	$E_{3,0}$	$E_{4,0}$
$E_{1,1}$	$E_{2,1}$	$E_{3,1}$	$E_{4,1}$

msk =

$D_{1,0}$
$D_{1,1}$

Application: Identity-Based Encryption

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

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,$D_{1,1}$)

Application: Identity-Based Encryption

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

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

Universal Samplers

*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 *params*
- Samp(params,C): Deterministically output p_C, a valid output from C.

Encryptor Combiners from Universal Samplers

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

Encryptor Combiners from Universal Samplers

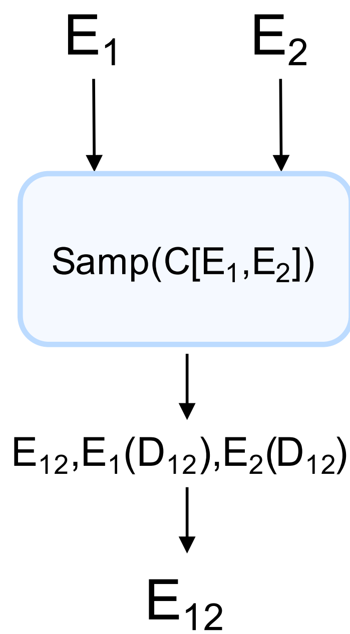
Circuit $C[E_i,E_j](r)$:
Generate E_{ij},D_{ij} from PKE with randomness r.
Output $E_{ij},E_i(D_{ij}),E_j(D_{ij})$

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 $E_{ij},E_i(D_{ij}),E_j(D_{ij})$

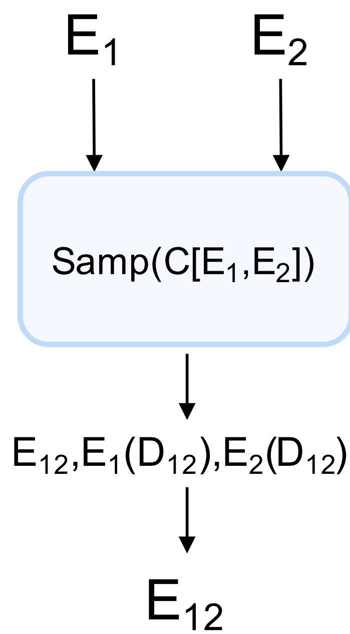
Samp($C[E_i,E_j]$) outputs $E_{ij},E_i(D_{ij}),E_j(D_{ij})$ for “fresh” E_{ij},D_{ij}



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}

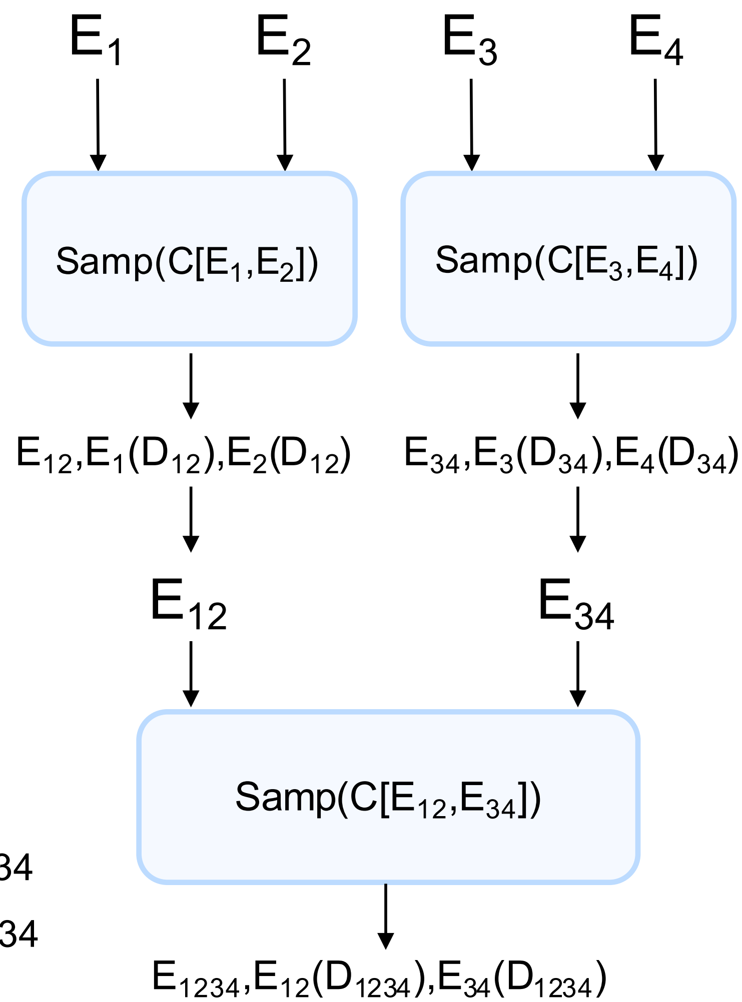


Given D_1 or D_2, run Samp($C[E_1,E_2]$) to get $E_{12},E_1(D_{12}),E_2(D_{12})$. Decrypt to recover D_{12}.

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

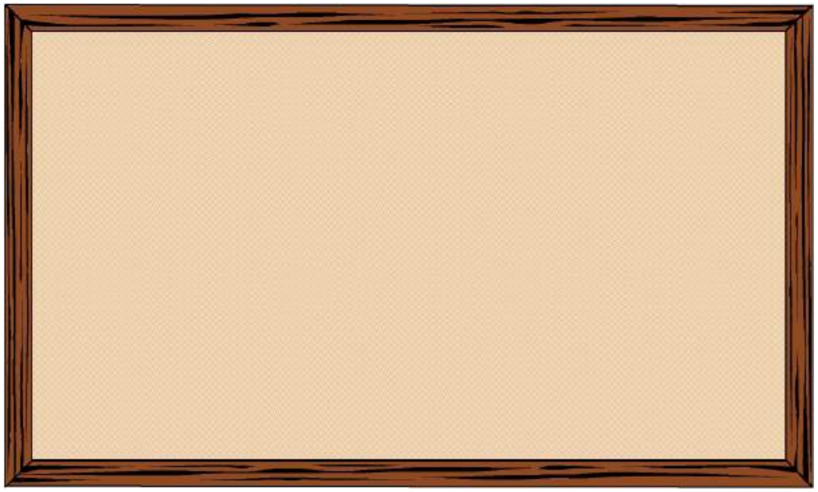
1. Define Encryptor Combiners ✓
2. Identity Based Encryption from Encryptor Combiners ✓
3. Encryptor Combiner Construction from Universal Samplers ✓
4. Multiparty NIKE from Encryptor Combiners
5. Encryptor Combiners for Dual Regev Encryption

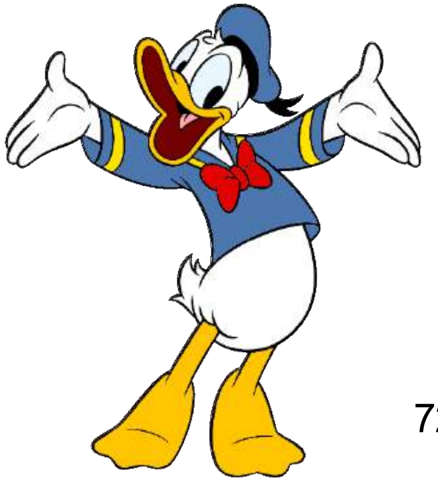
Application: Multiparty NIKE

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

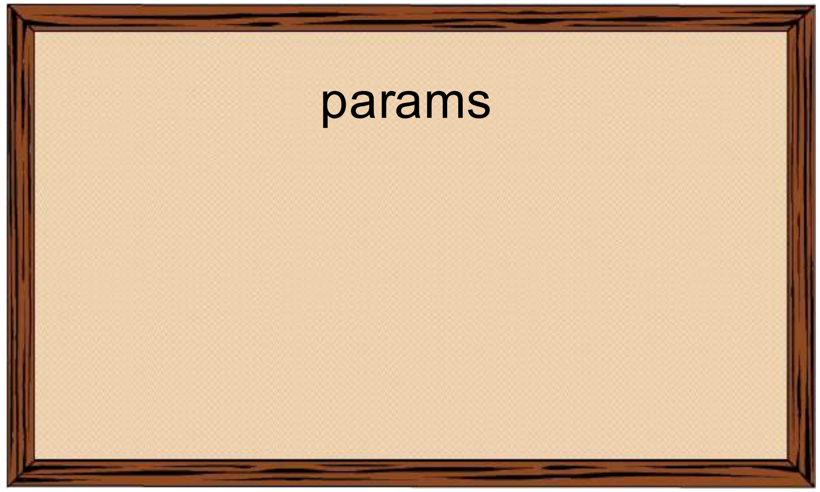
*but we need a new definition!

Multiparty NIKE from Encryptor Combiners



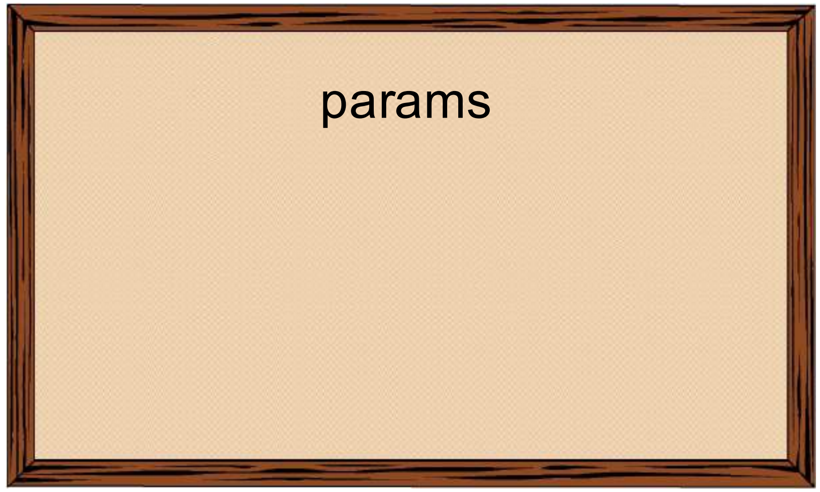


Multiparty NIKE from
Encryptor Combiners



Setup() generates
params ← EC.Setup()

Multiparty NIKE from Encryptor Combiners

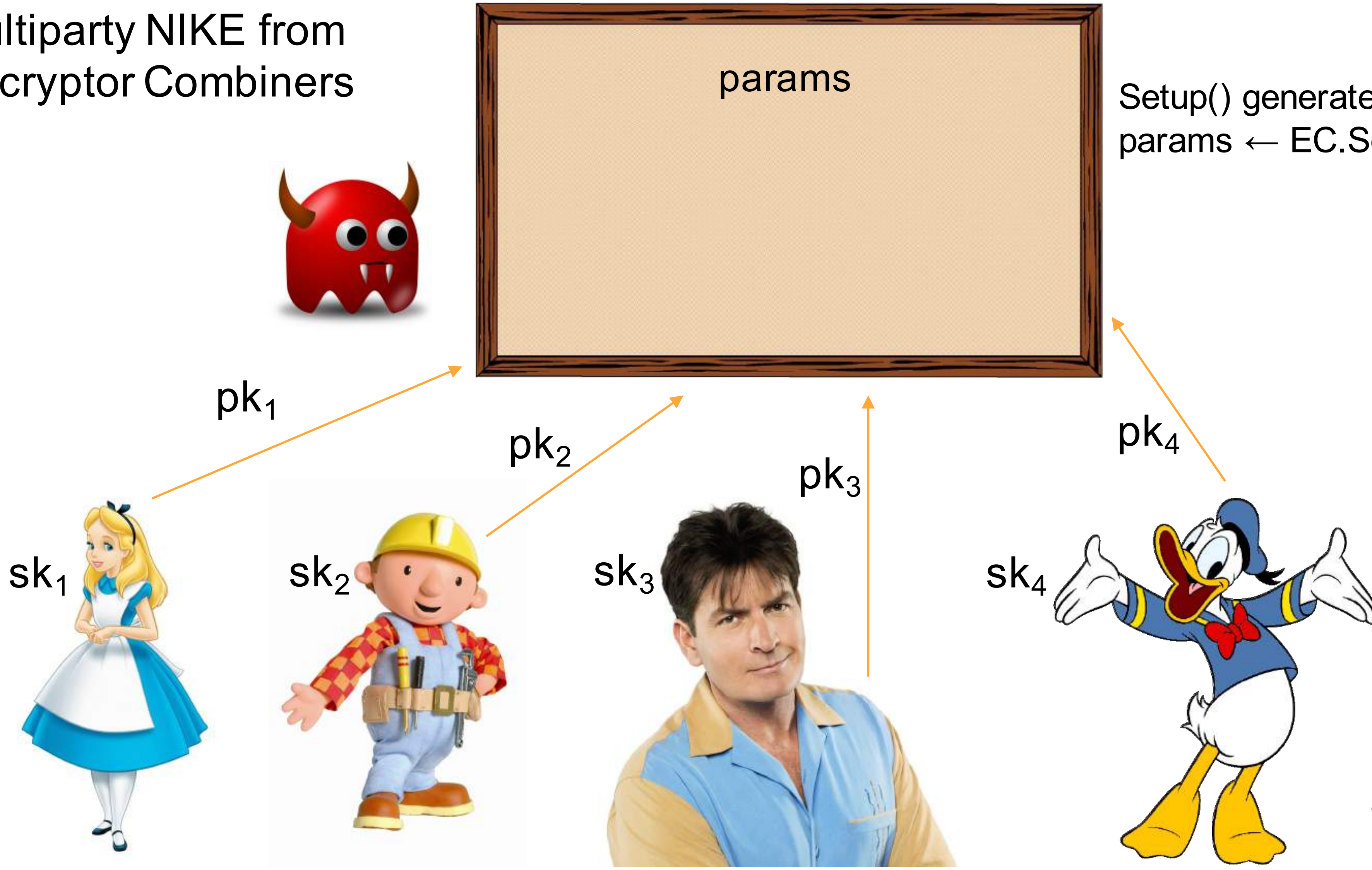


Setup() generates
params $\leftarrow$ EC.Setup()

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

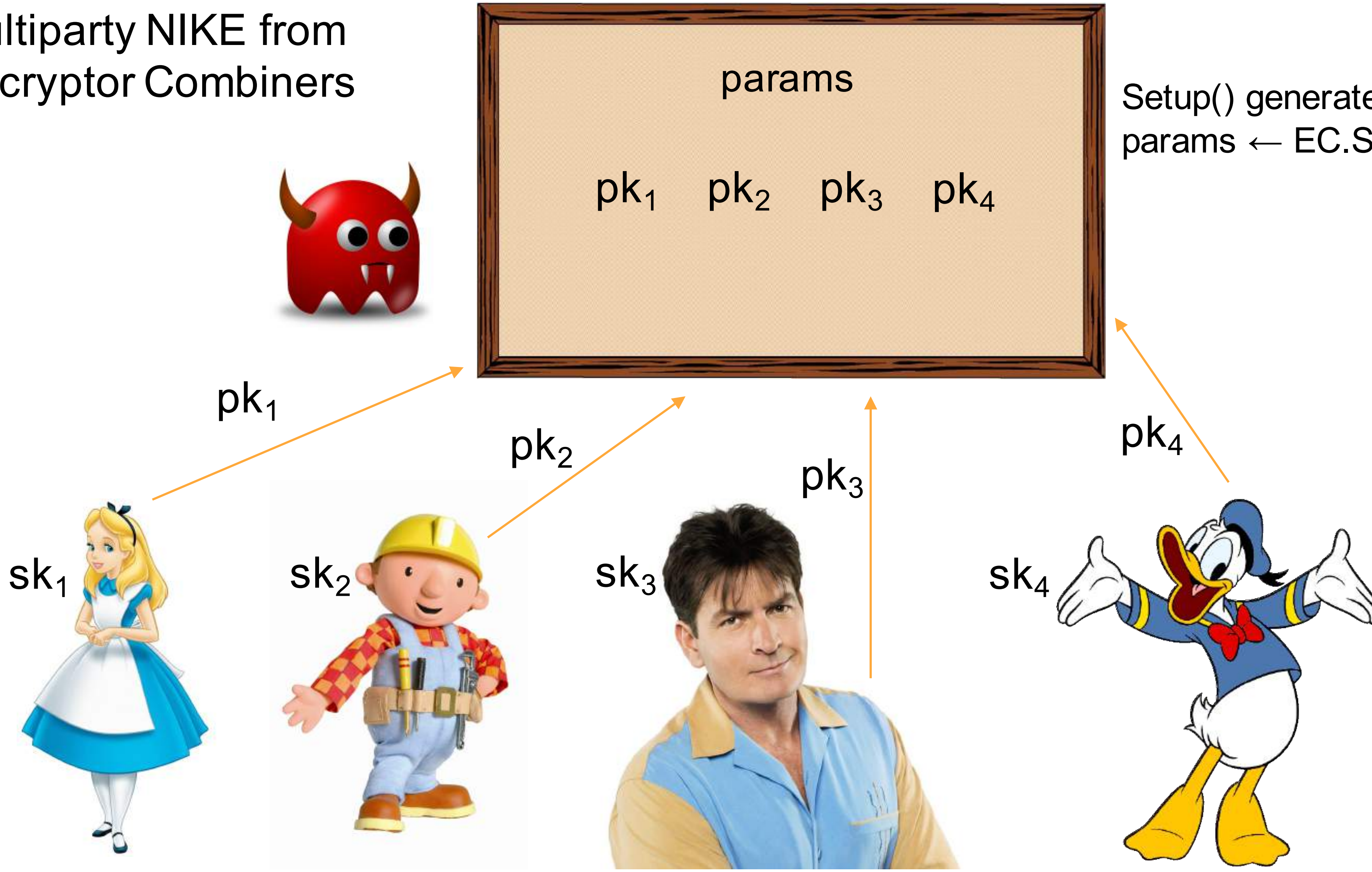
Multiparty NIKE from Encryptor Combiners

Setup() generates
params $\leftarrow$ EC.Setup()



Multiparty NIKE from Encryptor Combiners

Setup() generates
params $\leftarrow$ EC.Setup()



Multiparty NIKE from
Private ECs and
Public Key Encryption

params

pk_1 pk_2 pk_3 pk_4

Define E_i = Enc(pk_i,·) and D_i = Dec(sk_i,·).

sk_1 sk_2 sk_3 sk_4

User i computes $D^* \leftarrow$ EC.Extract(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

params

pk_1 pk_2 pk_3 pk_4

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

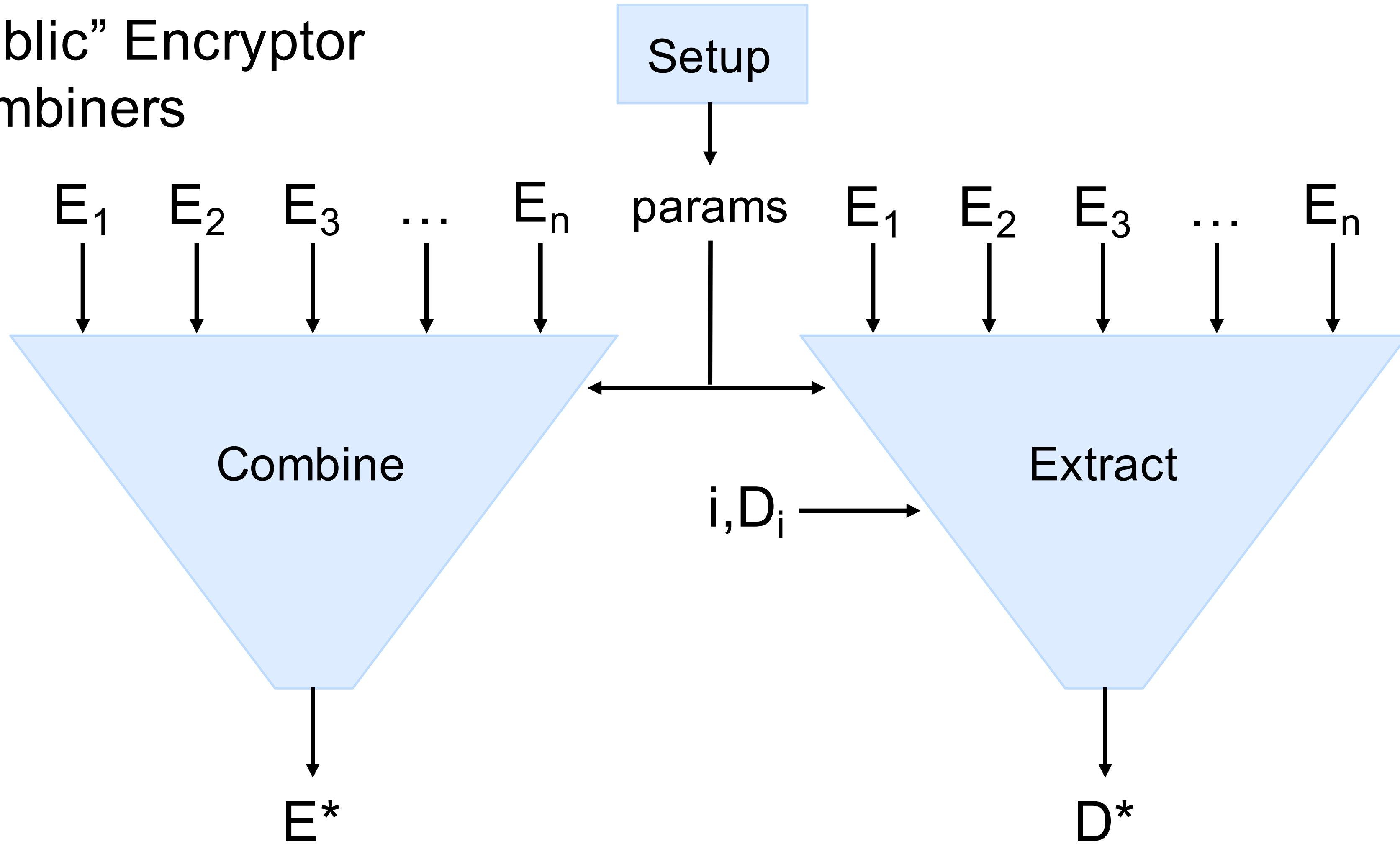
sk_1 sk_2 sk_3 sk_4

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

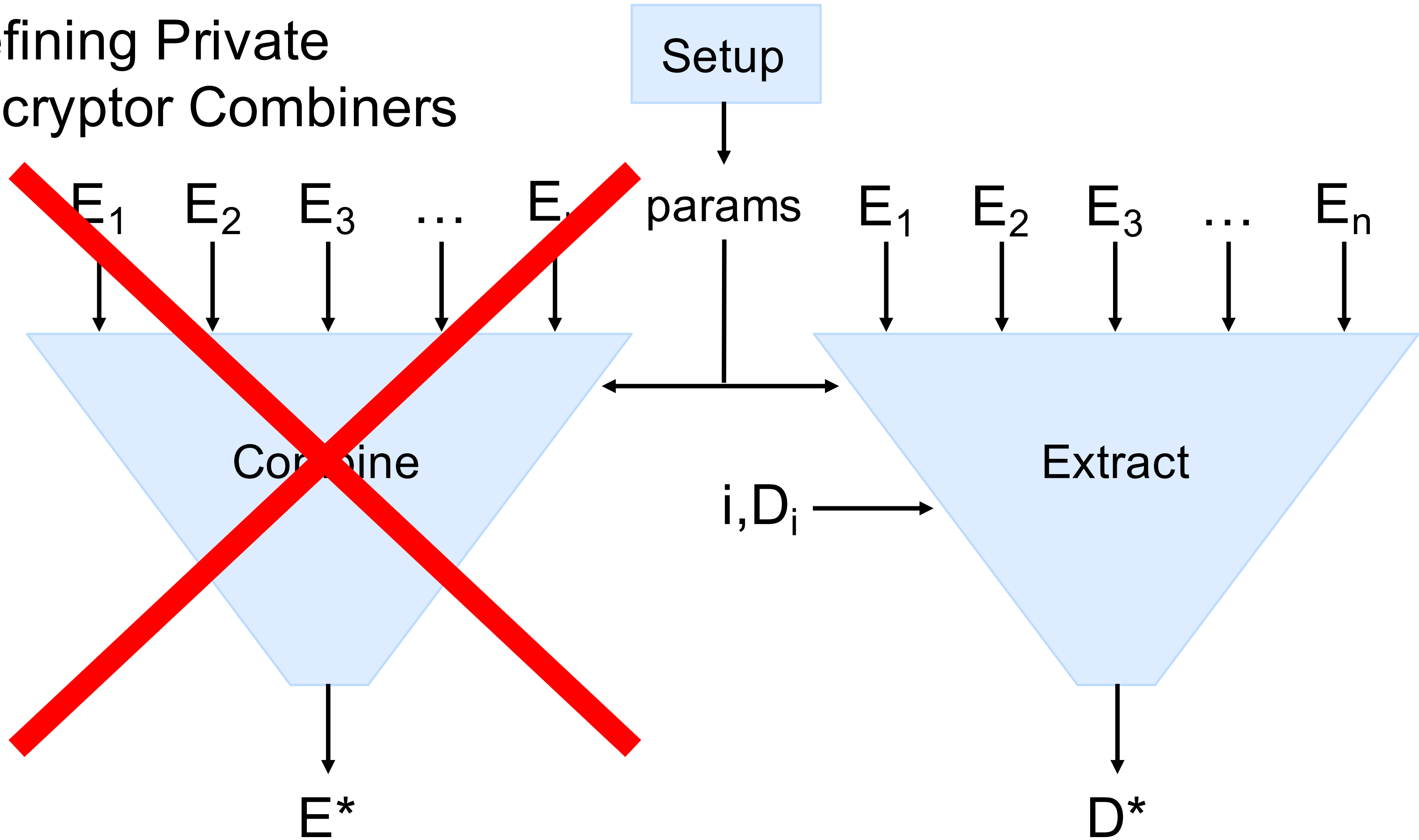
Interpret D^* as string of bits, use this as shared key.

Problem: D^* might not look pseudorandom

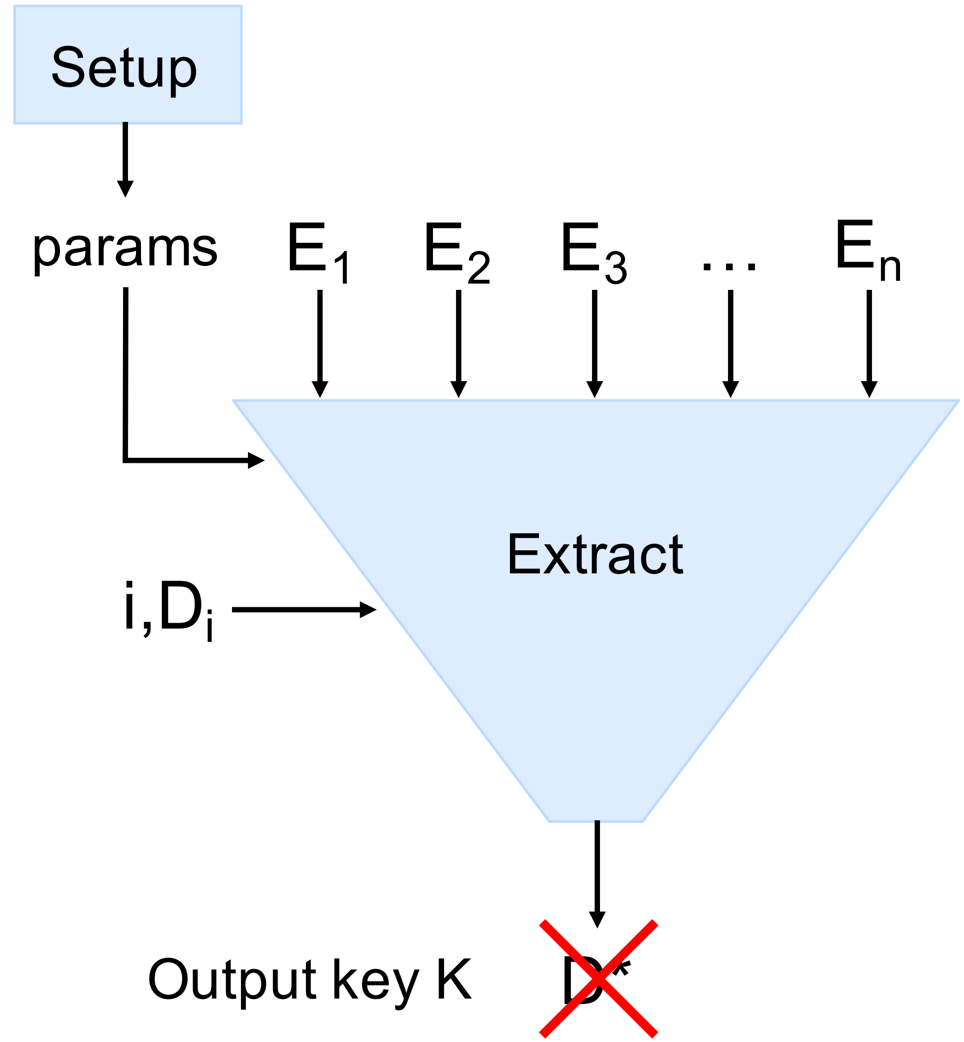
“Public” Encryptor Combiners



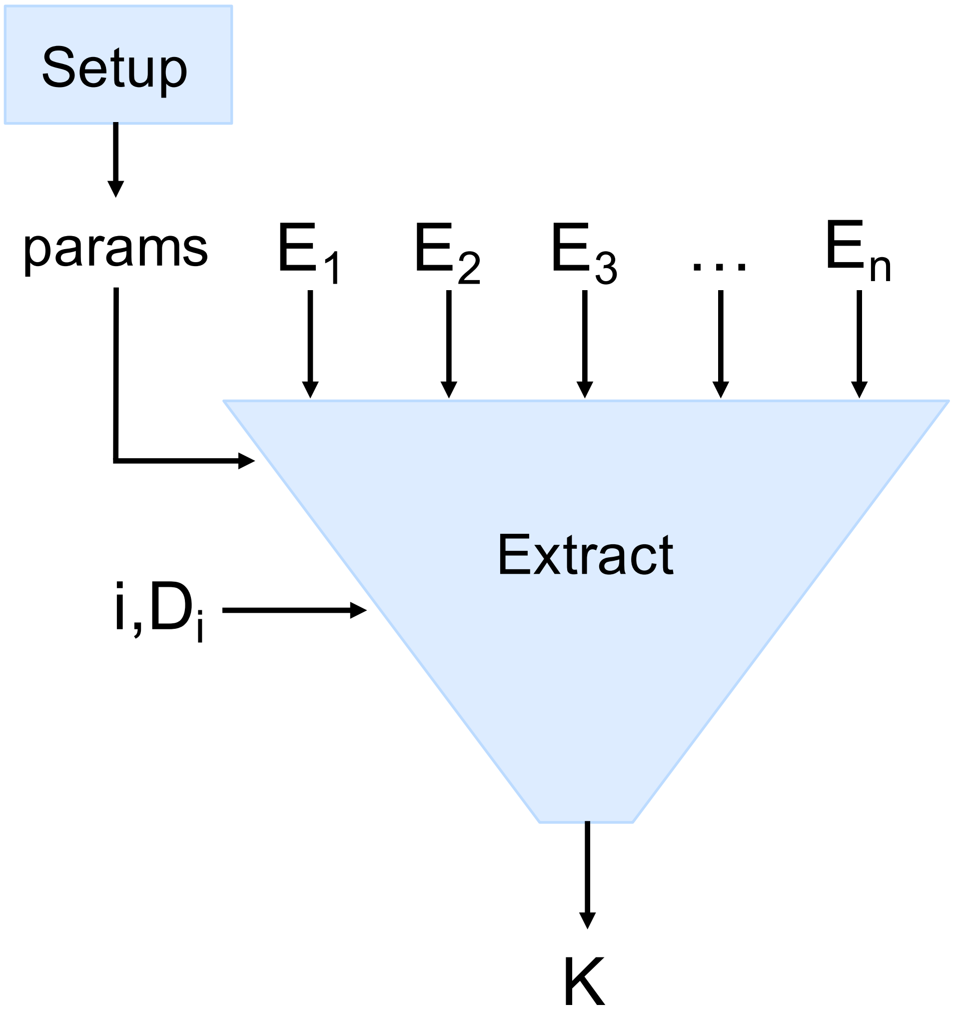
Defining Private Encryptor Combiners



Defining Private Encryptor Combiners

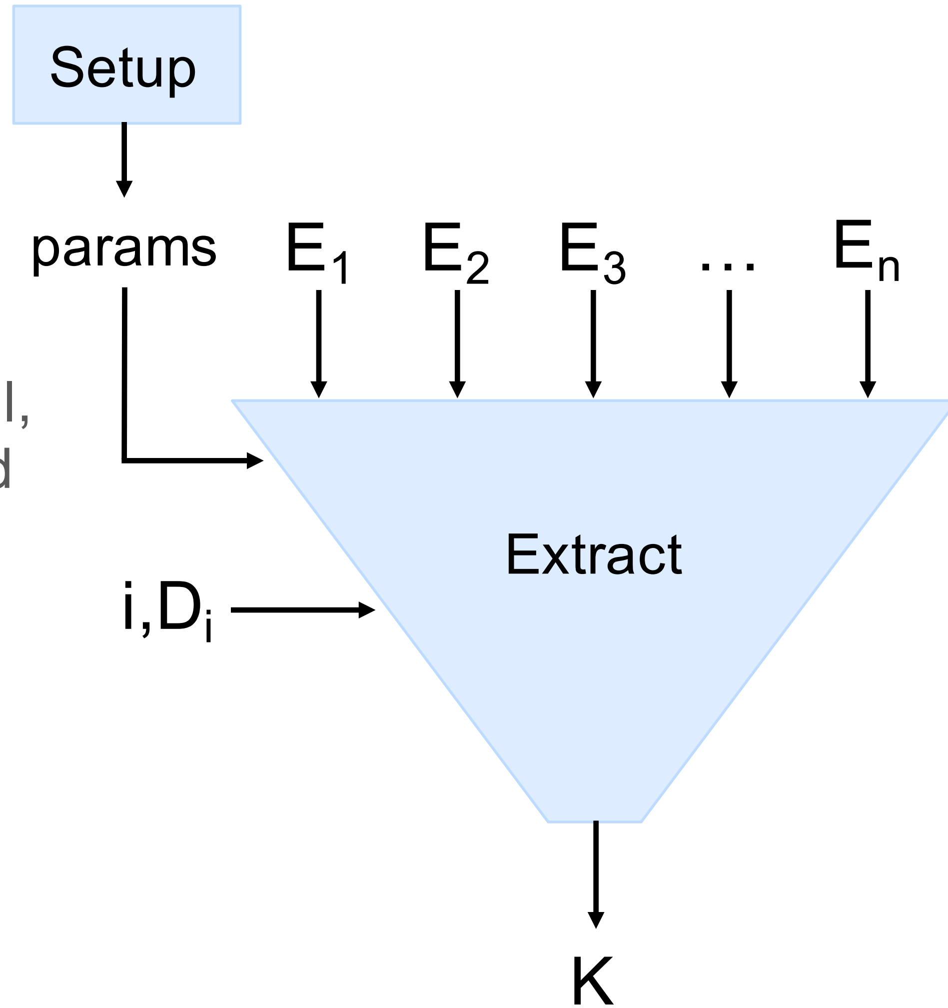


Defining Private Encryptor Combiners



Defining Private Encryptor Combiners

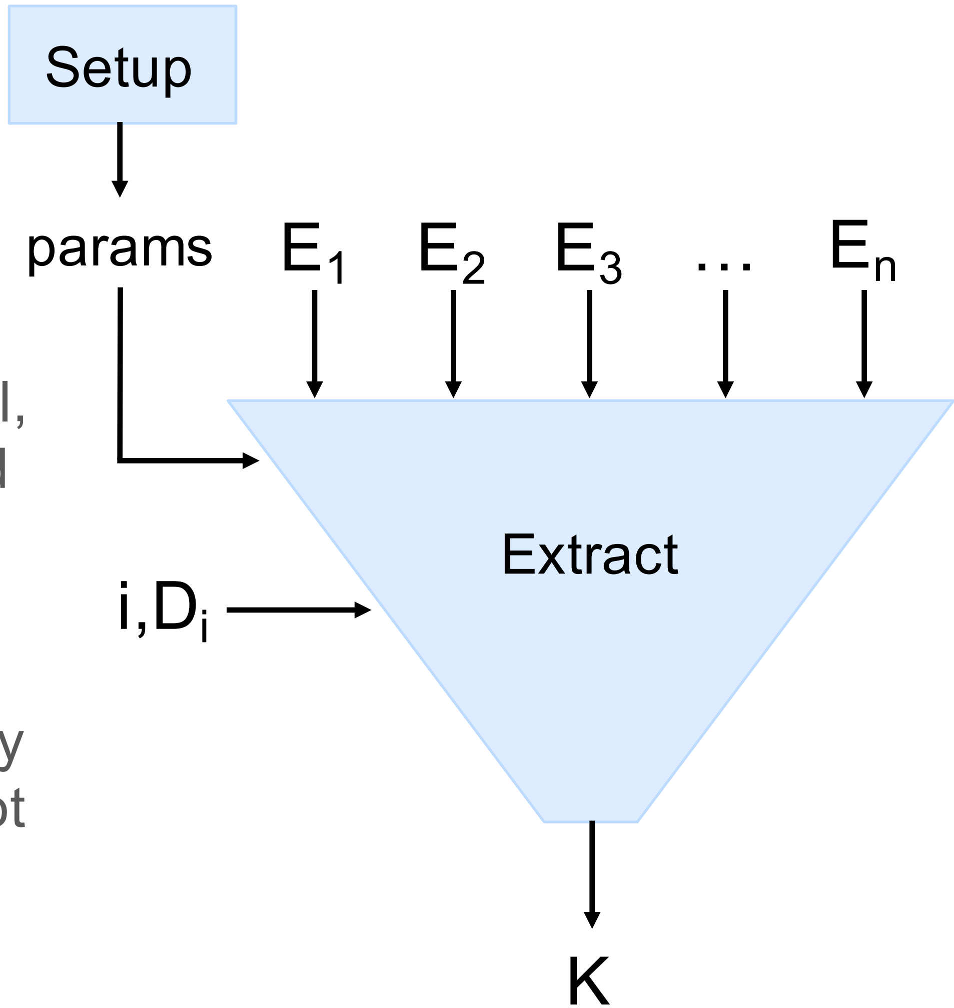
Correctness: For any pair of indices j,k, with probability 1-negl, Extract(*params*,E_1,…,E_n,j,D_j) and Extract(*params*,E_1,…,E_n,k,D_k) generate the same key.

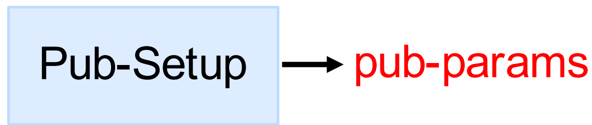


Defining Private Encryptor Combiners

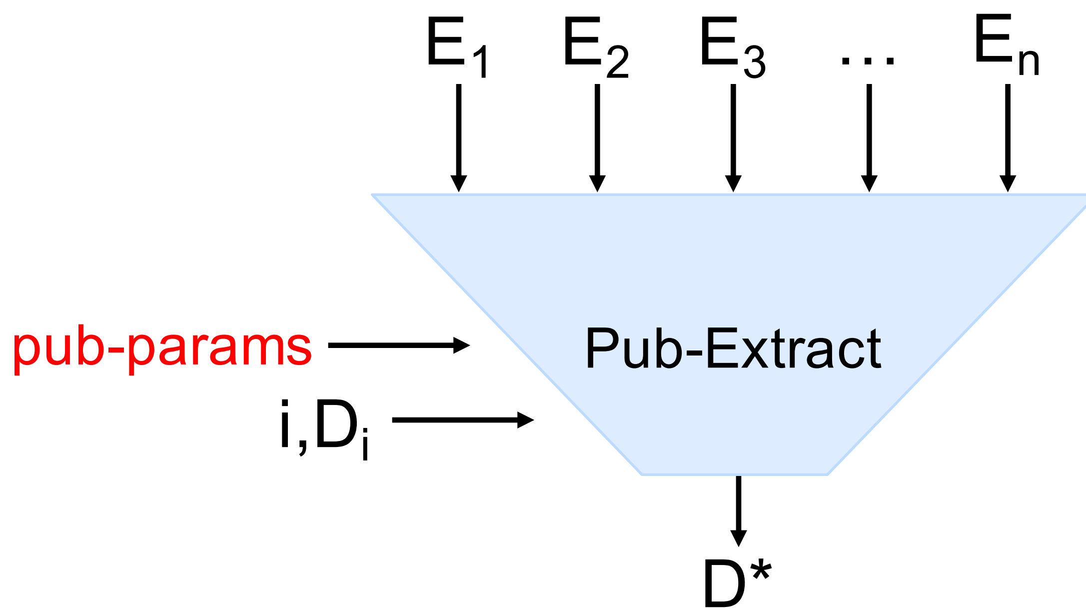
Correctness: For any pair of indices j,k, with probability 1-negl, Extract(*params*,E_1,…,E_n,j,D_j) and Extract(*params*,E_1,…,E_n,k,D_k) generate the same key.

Security: (Intuition) An adversary who does not know any D_i cannot distinguish K from random K drawn from the keyspace.

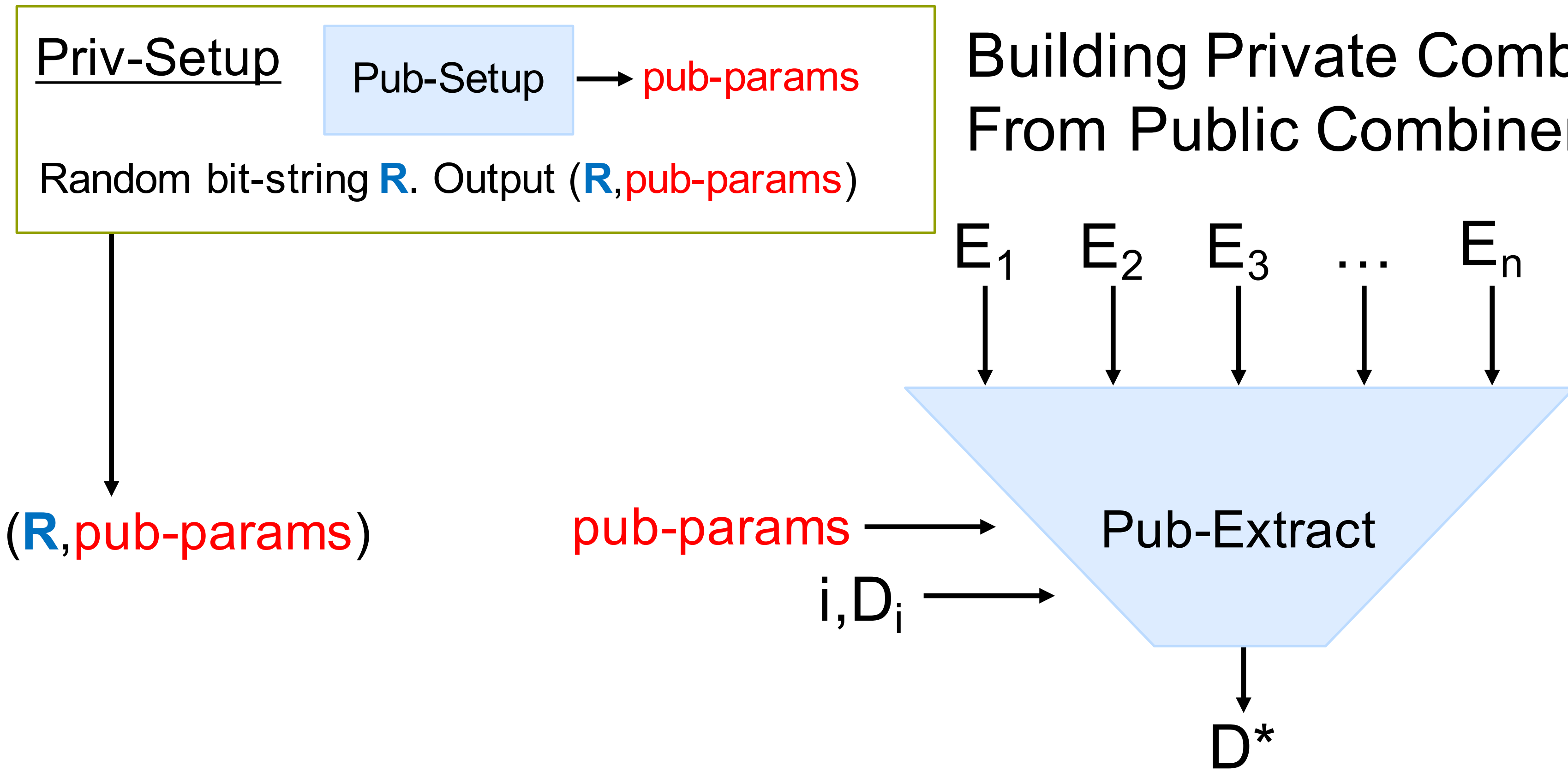




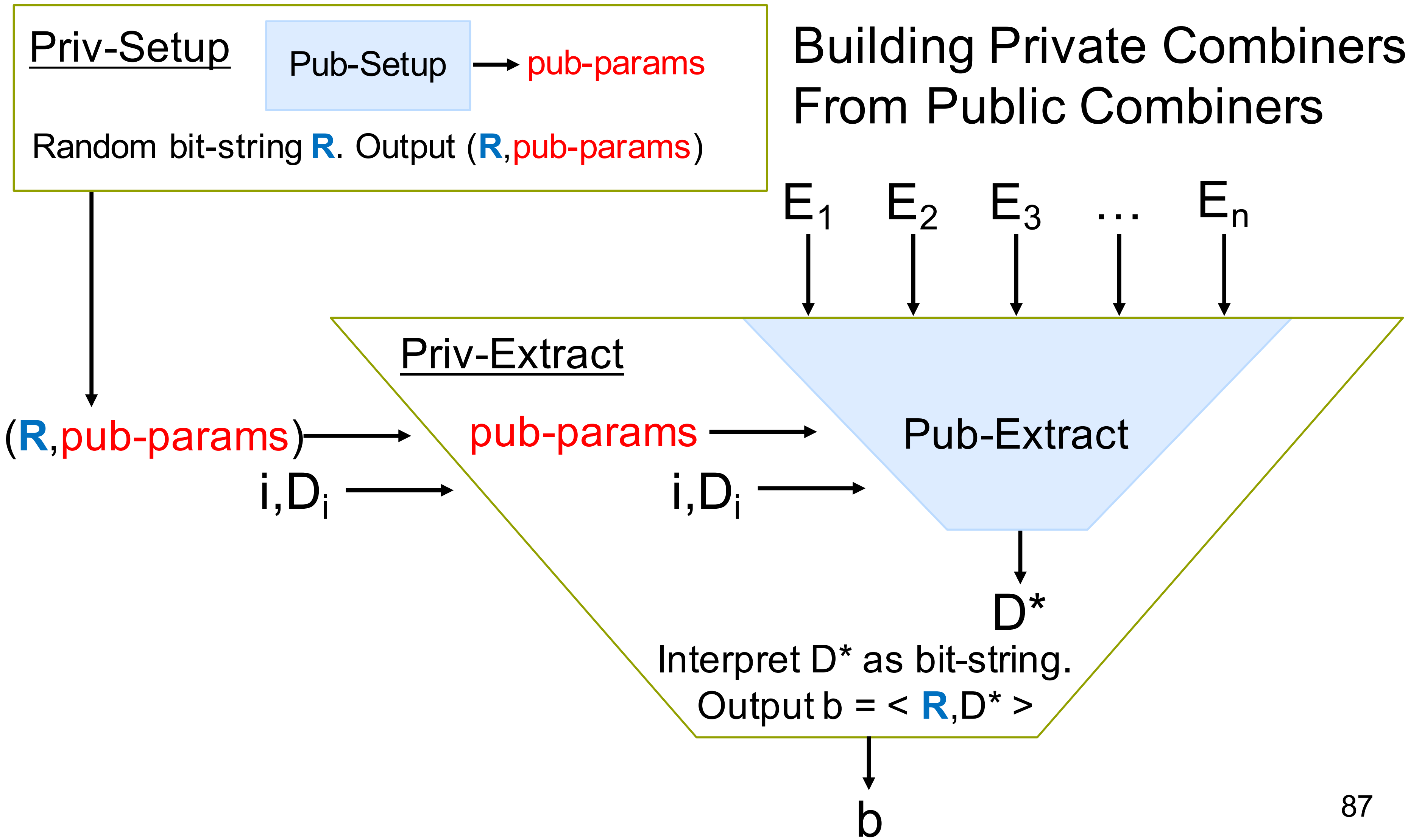
Building Private Combiners From Public Combiners



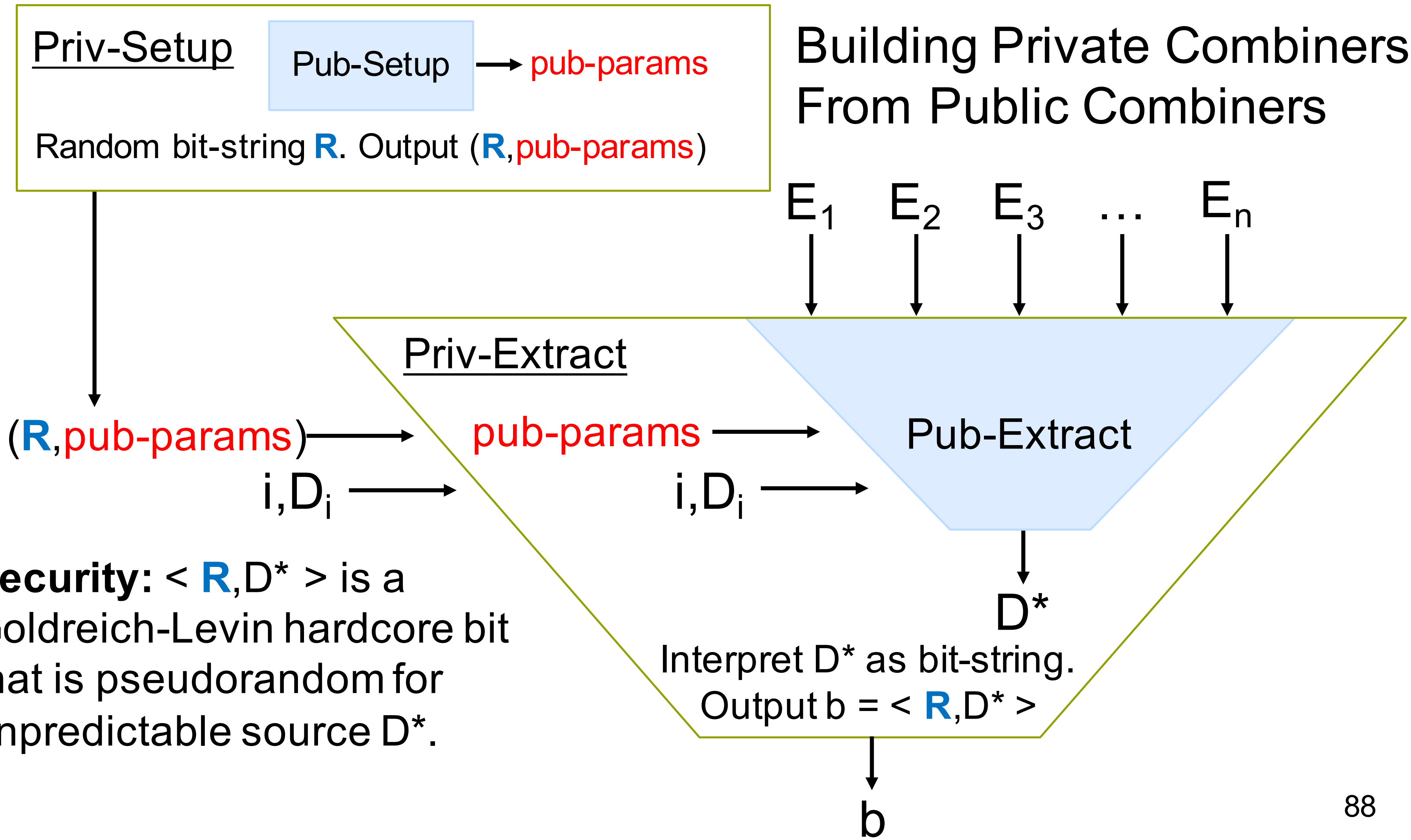
Building Private Combiners From Public Combiners



Building Private Combiners From Public Combiners

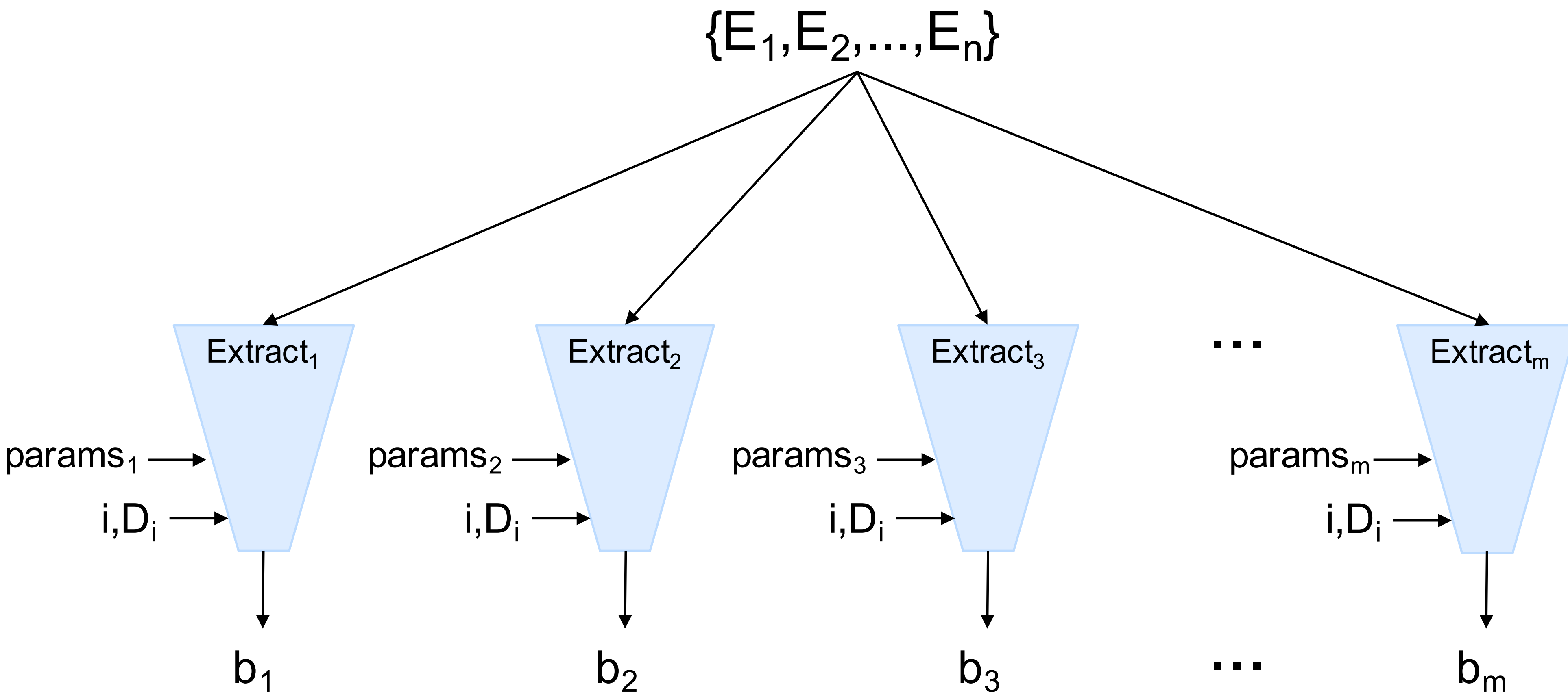


Building Private Combiners From Public Combiners

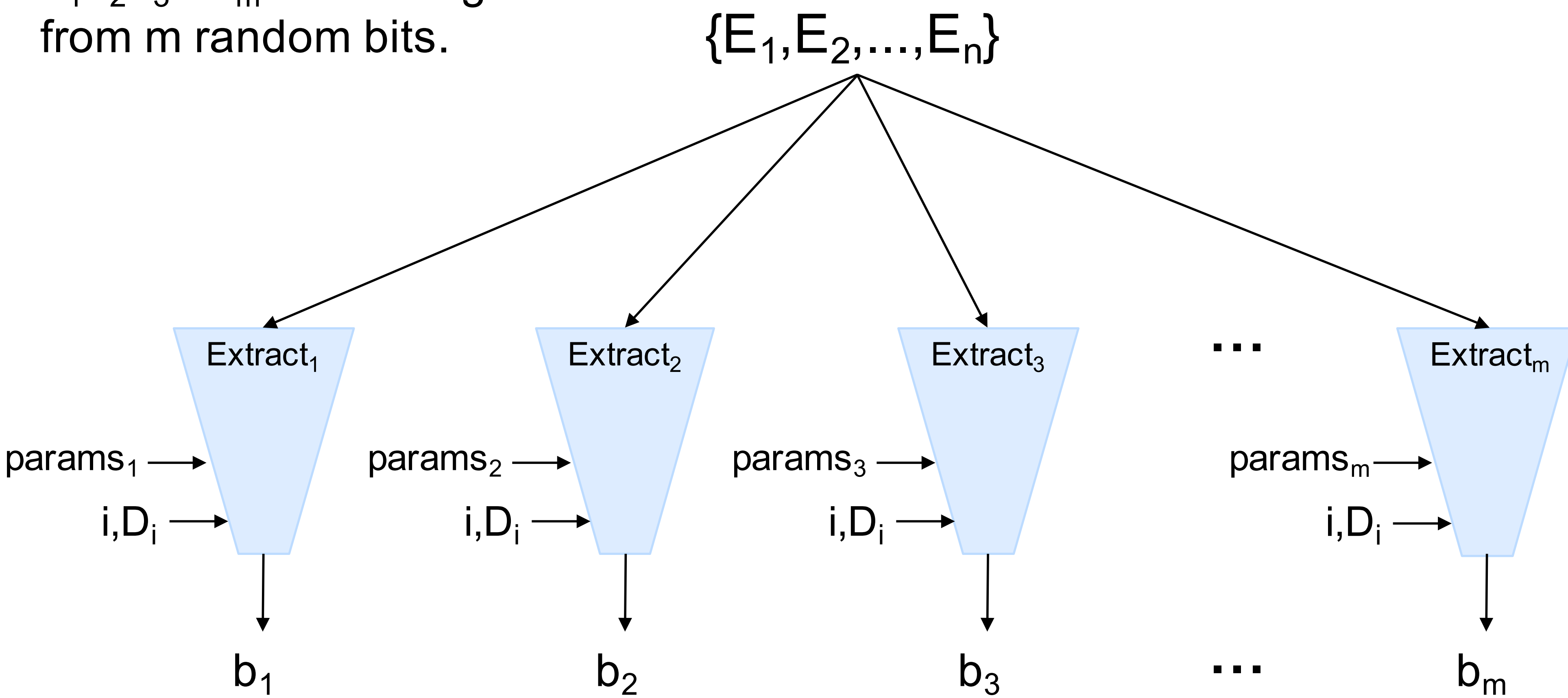


Security: < R,D* > is a Goldreich-Levin hardcore bit that is pseudorandom for unpredictable source D*.

Do we get an m-bit combiner if we run m 1-bit combiners in parallel?

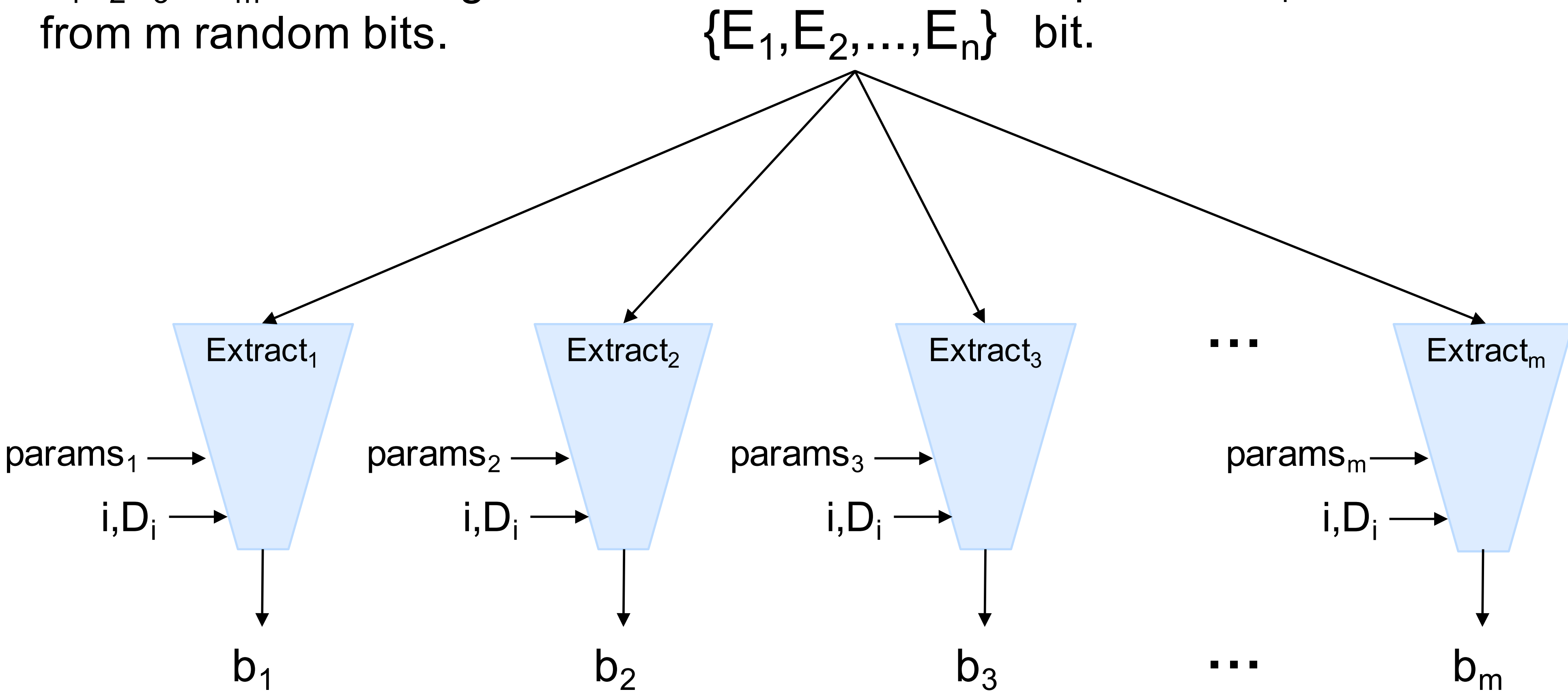


Security: Want to show $b_1b_2b_3...b_m$ is indistinguishable from m random bits.



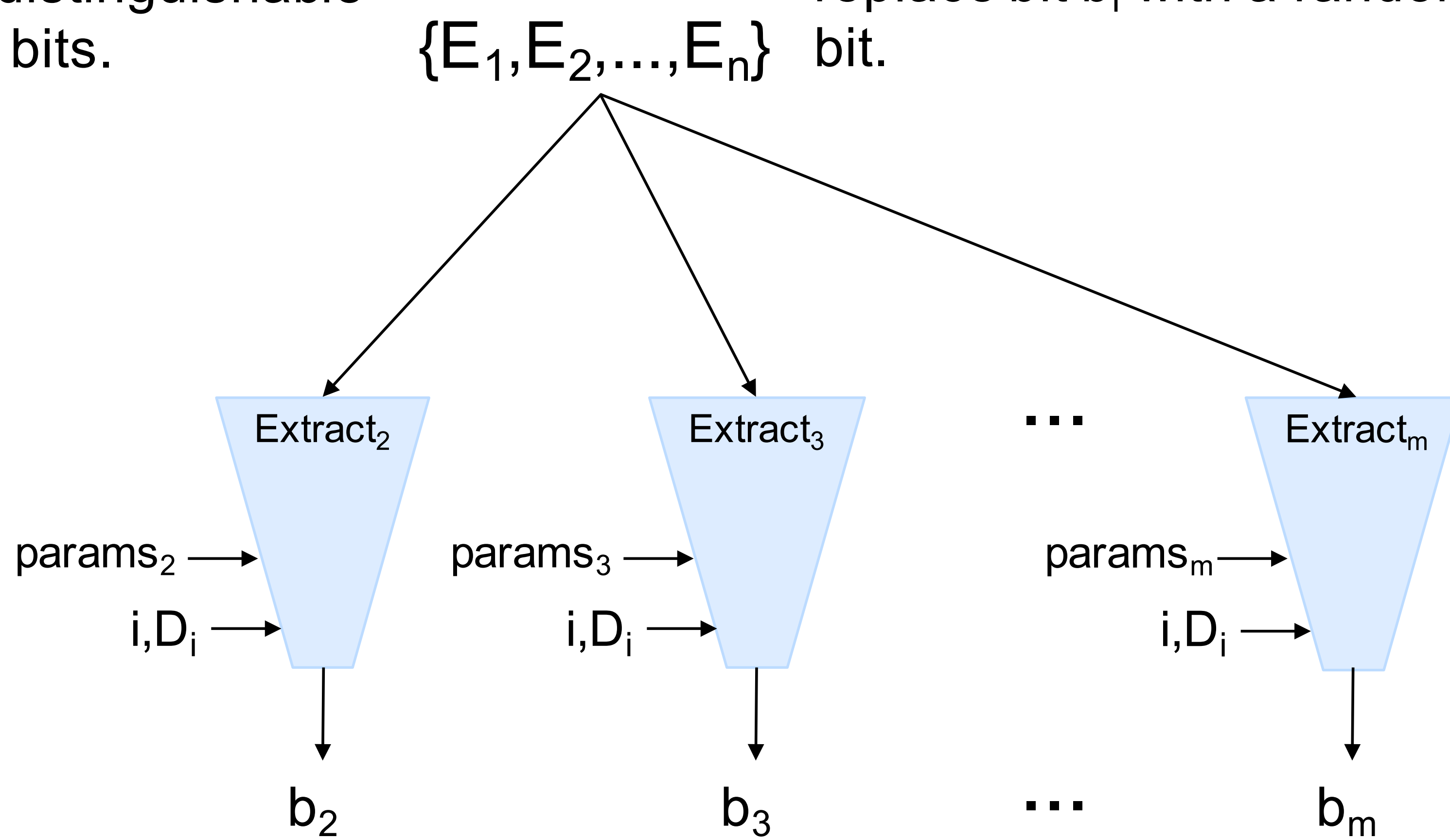
Security: Want to show $b_1 b_2 b_3 \ldots b_m$ is indistinguishable from m random bits.

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



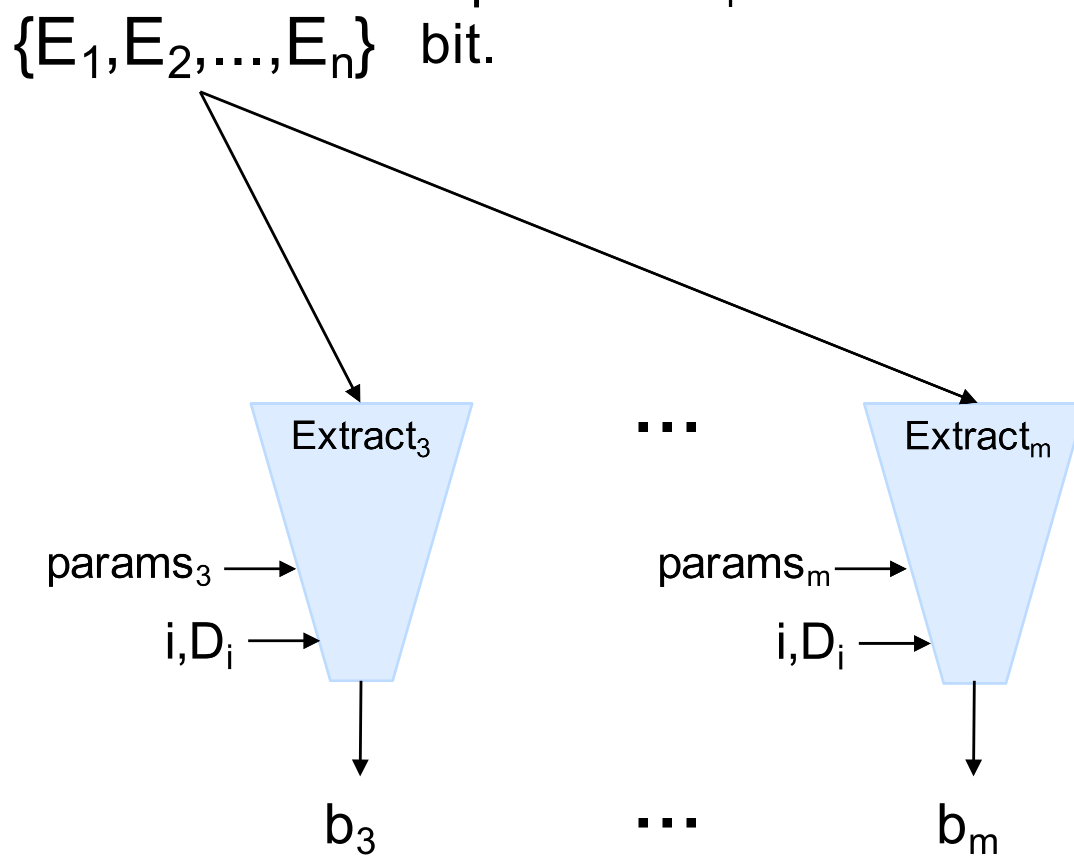
Security: Want to show $b_1b_2b_3...b_m$ is indistinguishable from m random bits.

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



Security: Want to show $b_1 b_2 b_3 \ldots b_m$ is indistinguishable from m random bits.

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

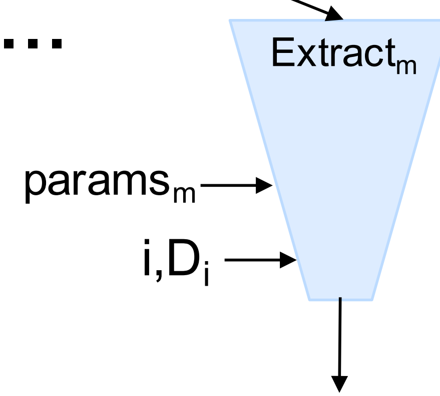


Security: Want to show $b_1b_2b_3...b_m$ is indistinguishable from m random bits.

$\{E_1,E_2,...,E_n\}$

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

...



r_1 r_2 r_3 ... b_m

Security: Want to show $b_1b_2b_3...b_m$ is indistinguishable from m random bits.

$\{E_1,E_2,...,E_n\}$

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

r_1 r_2 r_3 ... r_m

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.

r_1 r_2 r_3 … r_m

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.

r_1 r_2 r_3 … r_m

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.

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

r_1 r_2 r_3 … r_m

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.

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!

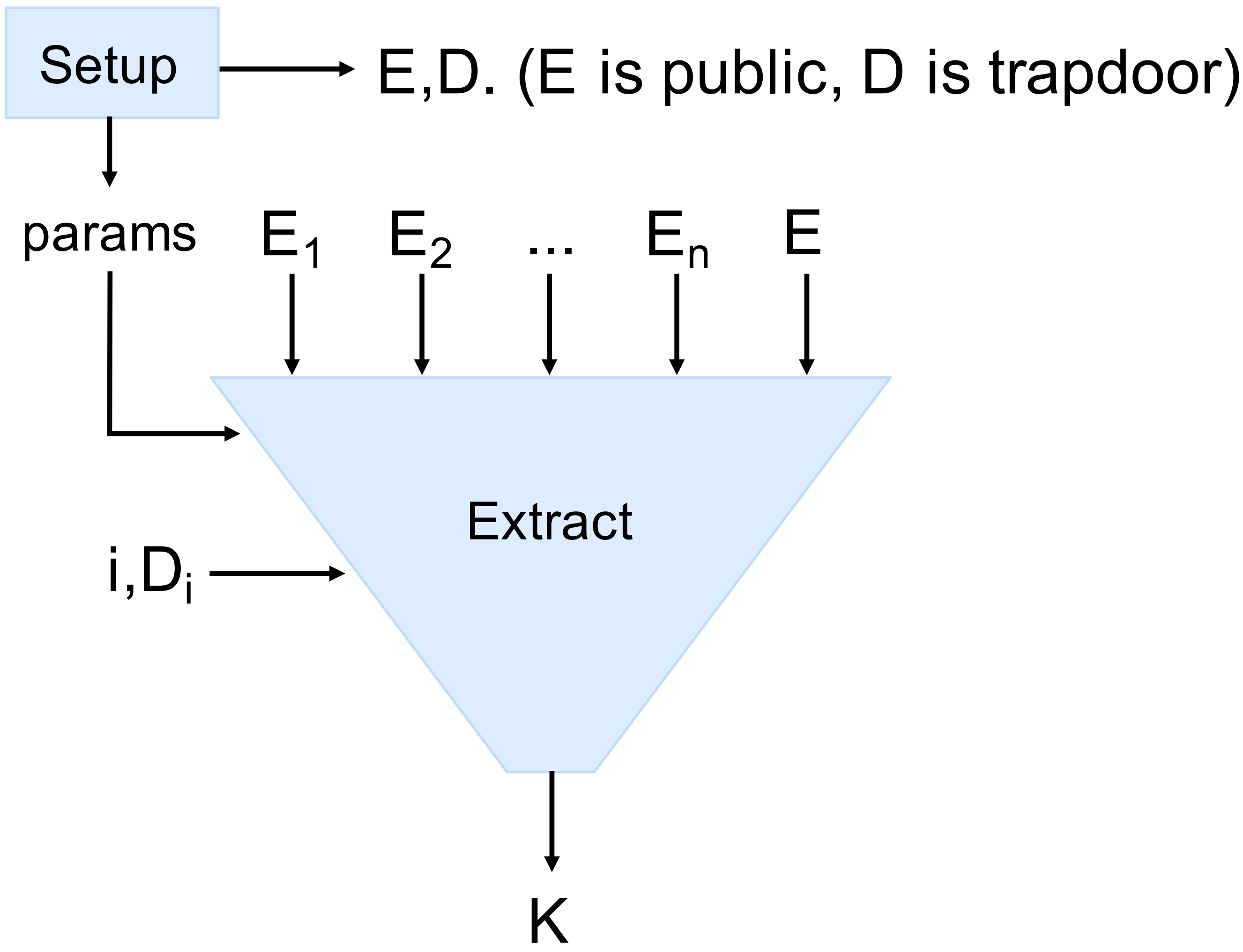
r_1 r_2 r_3 ... r_m

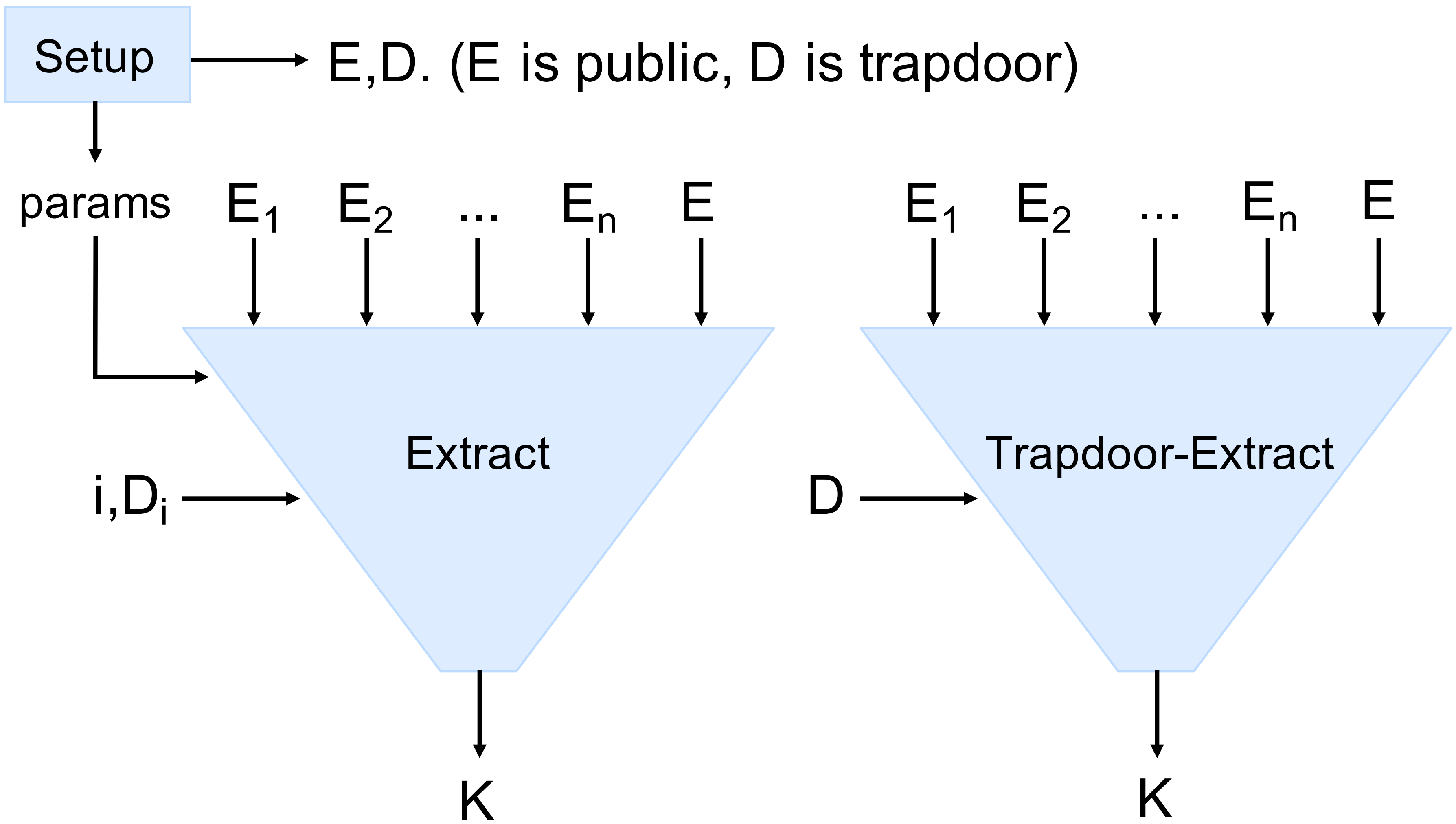
Trapdoored Encryptor Combiners

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

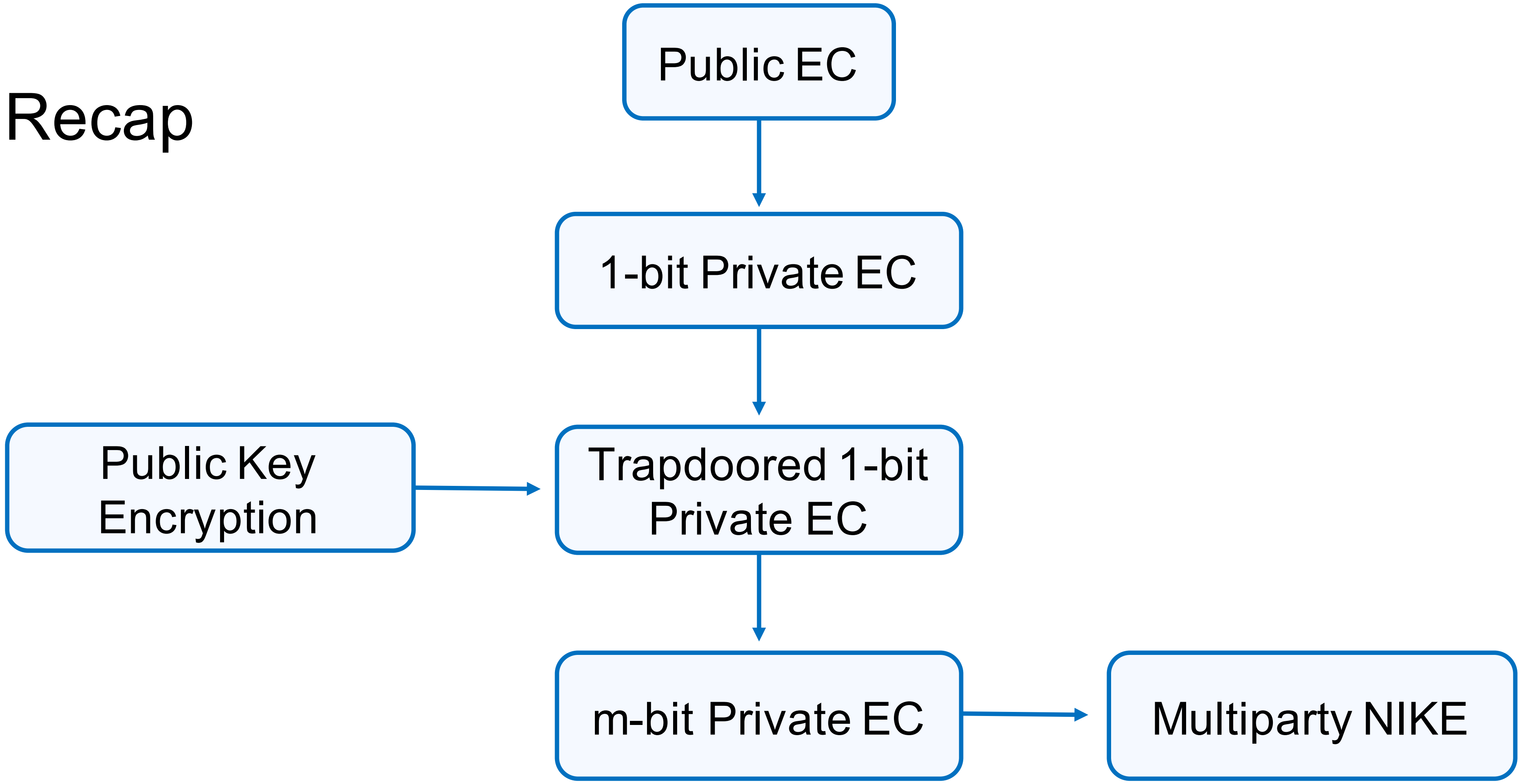
Trapdoored Encryptor Combiners

- Modify Setup() to also output an encryptor/decryptor pair E,D, where D is the trapdoor. E becomes part of *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
E,D. (E is public, D is trapdoor)
params
E1
E2
...
En
E
Extract
i,Di
K


Setup
E,D. (E is public, D is trapdoor)
params
E_1 E_2 ... E_n E
Extract
i,D_i
K
E_1 E_2 ... E_n E
Trapdoor-Extract
D
K

Recap

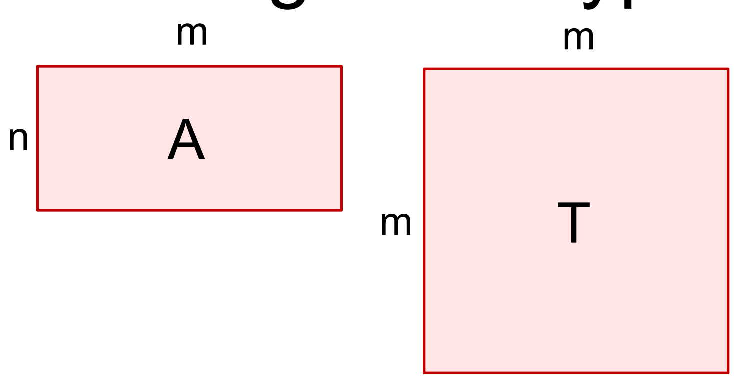


Public EC
1-bit Private EC
Public Key Encryption
Trapdoored 1-bit Private EC
m-bit Private EC
Multiparty NIKE

Outline

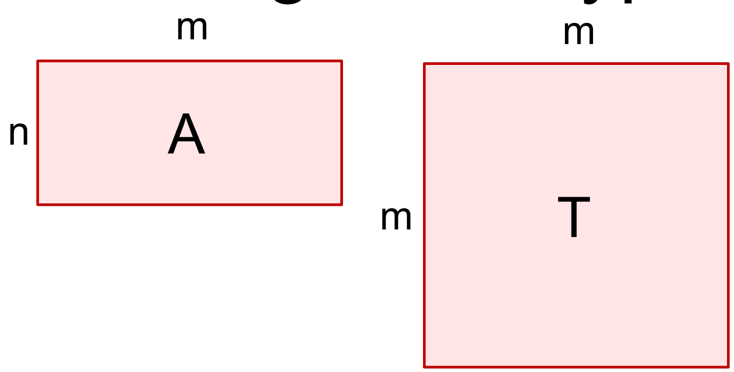
1. Define Encryptor Combiners ✓
2. Identity Based Encryption from Encryptor Combiners ✓
3. Encryptor Combiner Construction from Universal Samplers ✓
4. Multiparty NIKE from Encryptor Combiners ✓
5. Encryptor Combiners for Dual Regev Encryption

Dual Regev Encryption



All integers mod q
T is "short" full-rank matrix
over the rationals satisfying
A·T = 0 mod q

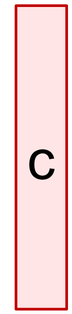
Dual Regev Encryption



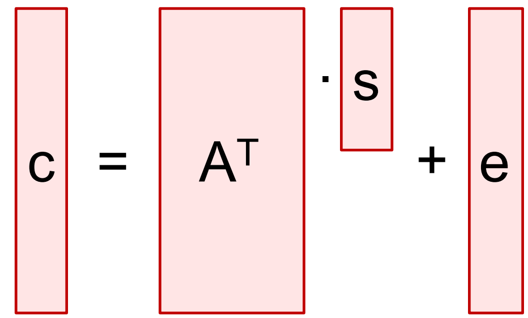
All integers mod q
T is “short” full-rank matrix
over the rationals satisfying
A·T = 0 mod q

Encryption

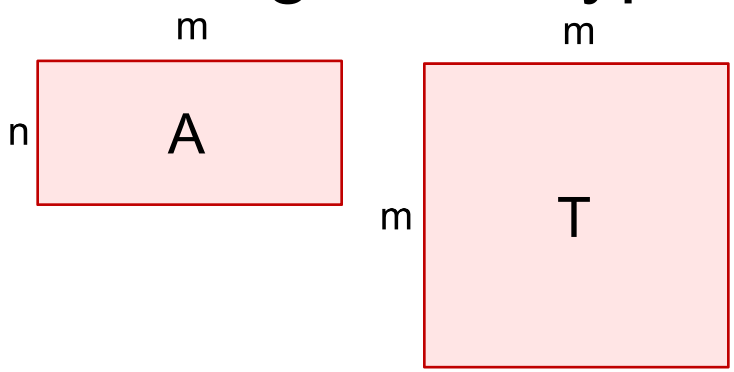
To encrypt b = 1, output a
random 1 x m vector c:



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



Dual Regev Encryption



All integers mod q
T is "short" full-rank matrix
over the rationals satisfying
$A \cdot T = 0 \bmod q$

Encryption

To encrypt b = 1, output a
random 1 x m vector c:

c

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

$$c = A^T \cdot s + e$$

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

Dual Regev Encryption

A ($n \times m$), T ($m \times m$)

All integers mod q
T is “short” full-rank matrix over the rationals satisfying $A \cdot T = 0 \bmod q$

Encryption

To encrypt b = 1, output a random 1 x m vector c:

$$c$$

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

$$c = A^T \cdot s + e$$

Decryption

$$\left(c^T \; T \; (\text{mod } q) \right) T^{-1}$$

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

$$A^* = \boxed{A_1}\;\boxed{A_2}\;\cdots\;\boxed{A_m}$$

Given full rank "short" T_i such that $A_i \cdot T_i = 0 \bmod q$, we can compute corresponding T^* via known basis extension techniques [CHK+10, ABB10].

Combining Dual Regev Encryptors

$$A^* = \boxed{A_1}\ \boxed{A_2}\ \cdots\ \boxed{A_m}$$

Given full rank "short" T_i such that $A_i \cdot T_i = 0 \bmod q$, 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






Broadcast Encryption

Space of identities ID = {“A”, “B”, “C”, ...}
Setup(ID) → (mpk,msk)

NETFLIX



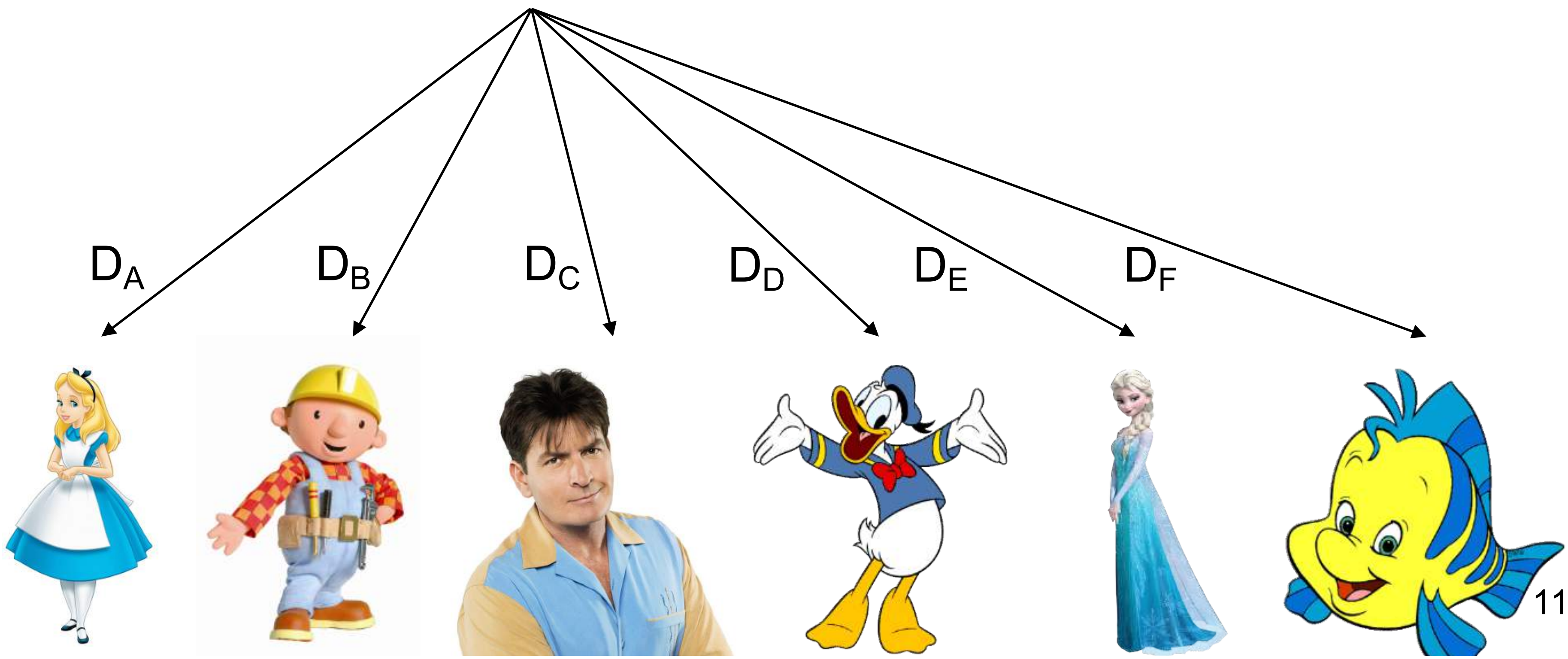



Broadcast Encryption

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

Setup(ID) → (mpk,msk)

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



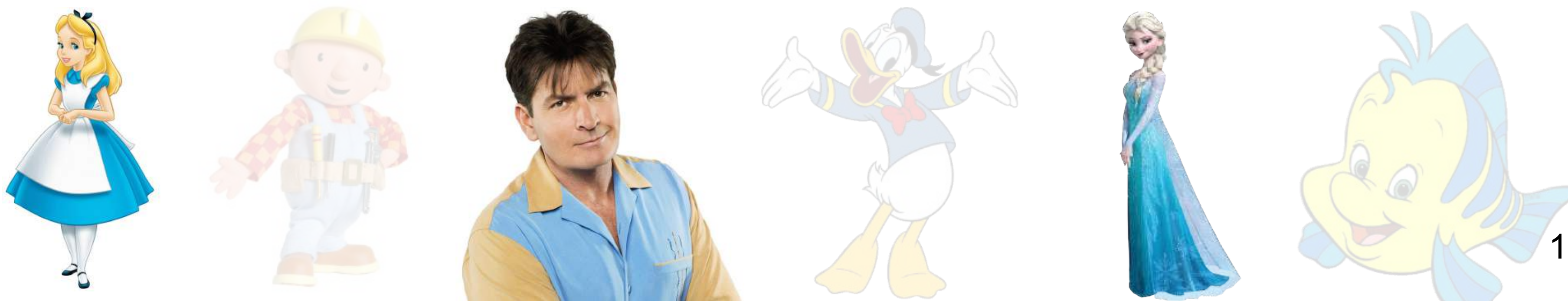
Broadcast Encryption

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

Setup(ID) → (mpk,msk)

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

D_A D_B D_C D_D D_E D_F



Broadcast Encryption

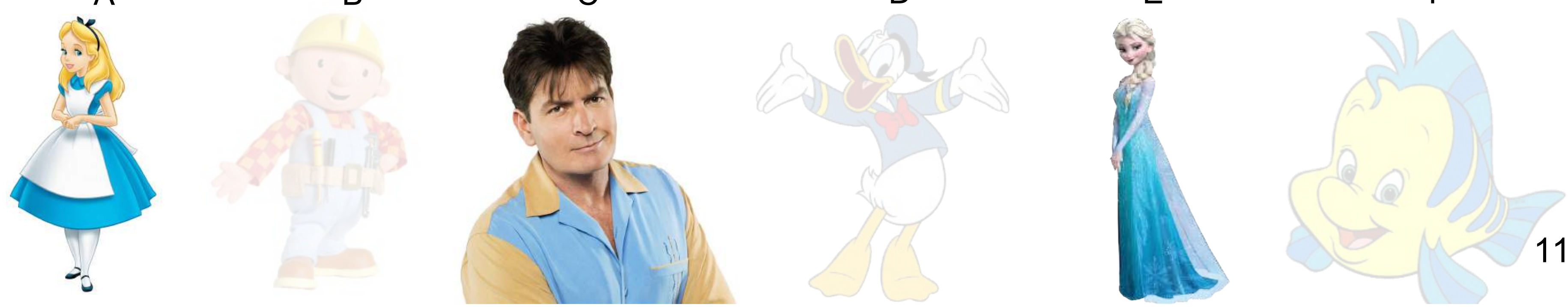
Space of identities ID = {“A”, “B”, “C”, ...}
Setup(ID) → (mpk,msk)

NETFLIX

Encrypt to set S = {“Alice”,“Charlie”,“Elsa”}
First, E ← EncGen(mpk,S).

Broadcast E(m) publicly

D_A D_B D_C D_D D_E D_F



Broadcast Encryption

Space of identities ID = {"A", "B", "C", ...}

Setup(ID) $\rightarrow$ (mpk,msk)

NETFLIX

Encrypt to set S = {"Alice","Charlie","Elsa"}

First, E $\leftarrow$ EncGen(mpk,S).

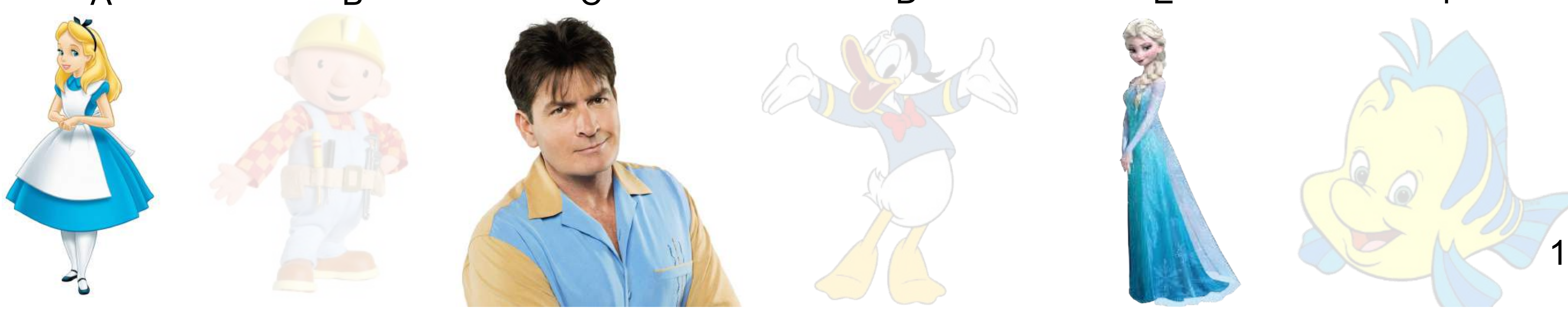
Broadcast E(m) publicly

Each user in S can compute

D $\leftarrow$ DecGen(mpk,id,D_{id},S)

and recover D(E(m)) = m

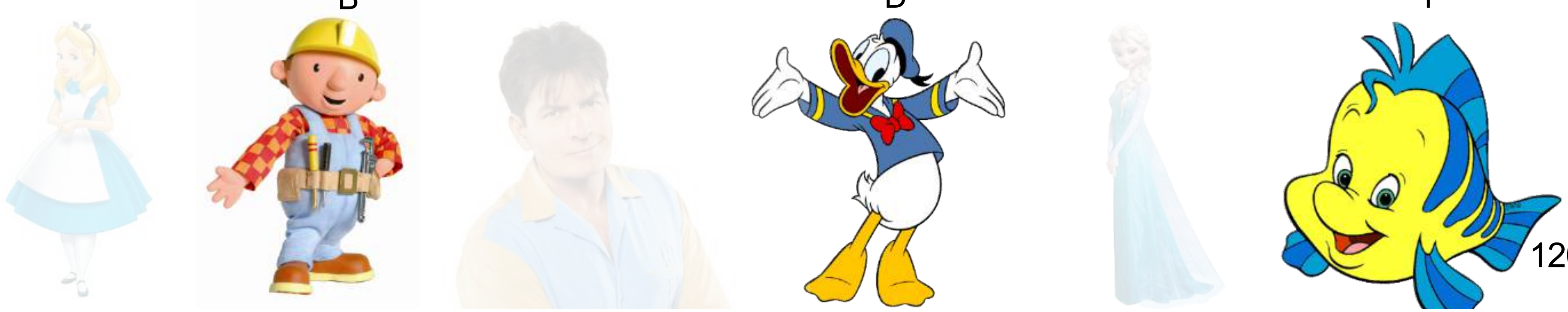
D_A D_B D_C D_D D_E D_F



Broadcast Encryption

Security: Users outside of S cannot decrypt a message intended for S, even if they collude.

D_B D_D D_F



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.