INDISTINGUISHABILITY OBFUSCATION

Mark Zhandry – Stanford University

* Joint work with Dan Boneh

Program Obfuscation

Intuition: Mangle a program

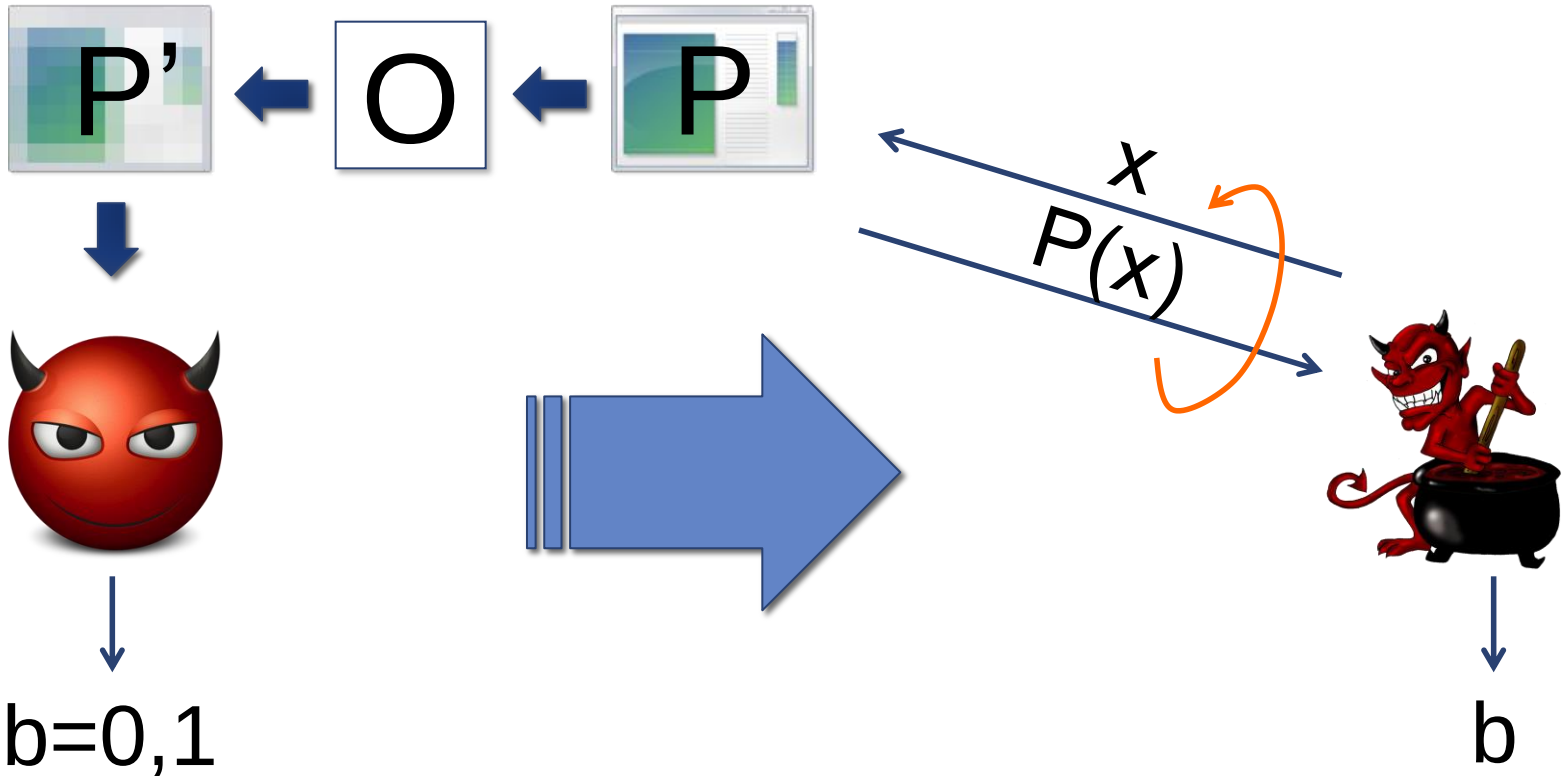
- Same functionality as original
- Hides all implementation details

Potential uses:

- IP protection
- Prevent tampering
- **Cryptography**

Virtual Black Box Obfuscation [BGI+'01]

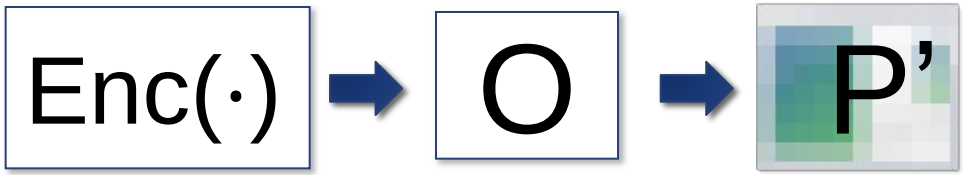
Having source code no better than black box access



Virtual Black Box Obfuscation

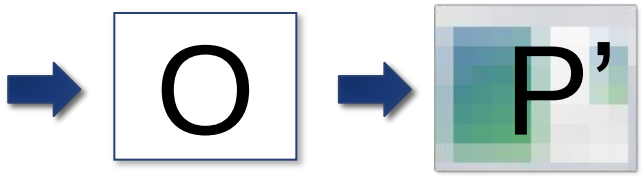
Potential Cryptographic Applications:

- Public key encryption from private key encryption:



- Homomorphic encryption:

```
P( c1 , c2 , ⊙∈{+, ×} ) {
    m1 ← Dec(c1)
    m2 ← Dec(c2)
    return Enc(m1⊙m2)
}
```

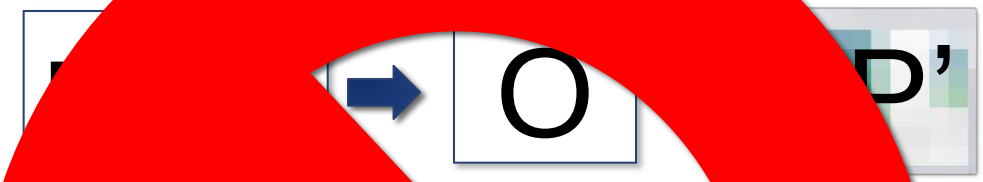


- Functional Encryption

Virtual Black Box Obfuscation

Potential Cryptographic Applications:

- Public key encryption:



- Homomorphic encryption:

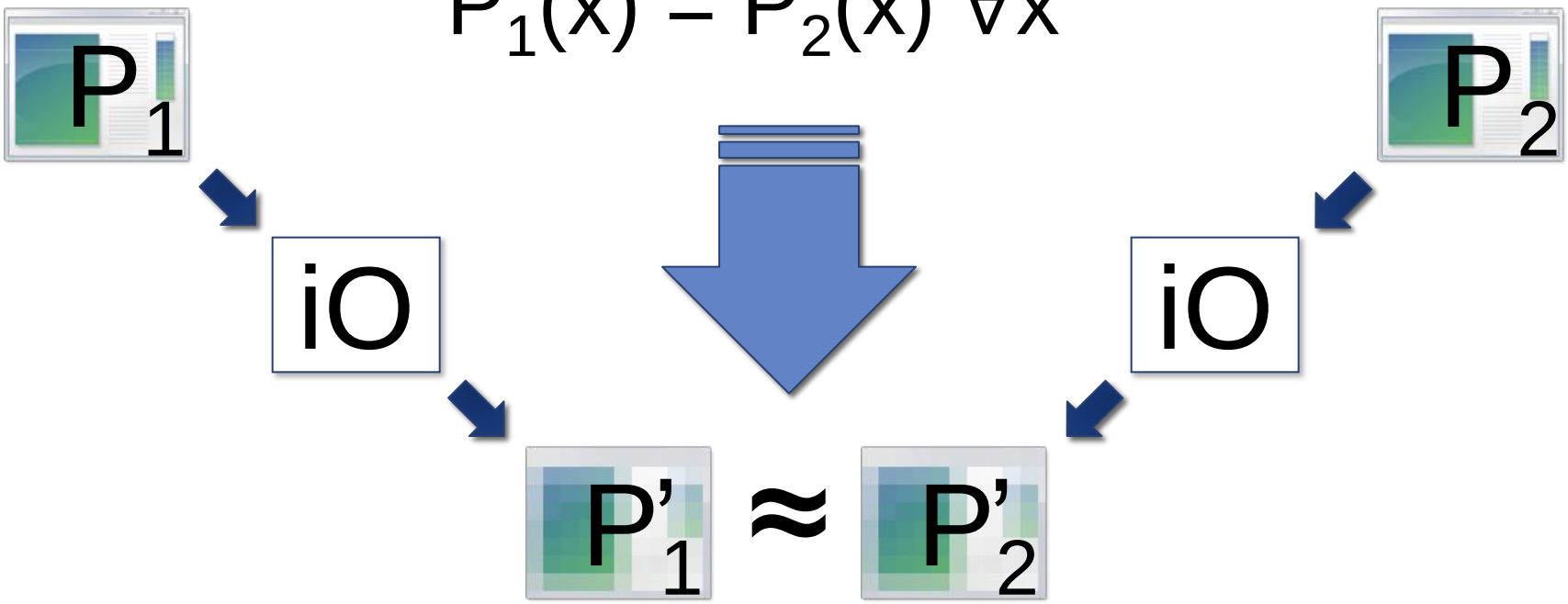
Theorem ([BGI+'01]): VBB for all programs is impossible

- Functional Encryption

Indistinguishability Obfuscation (iO) [BGI+'01]

If two programs have same functionality, obfuscations are indistinguishable

$$P_1(x) = P_2(x)\ \forall x$$



Indistinguishability Obfuscation (iO)

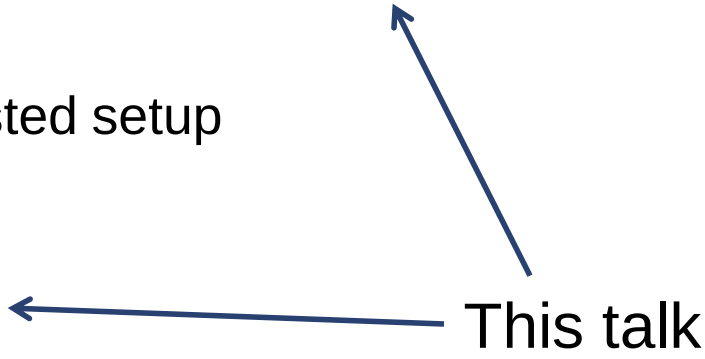
BGI^{+} counter example does not apply to iO

An exploding field:

- [BGI^{+}'01] Original definition
- [GR'07] Further investigation
- **[GGH^{+}'13] First candidate construction**
  - Functional encryption
- [BR'13, BGK^{+}'13, …] Additional constructions
- [SW'13, HSW'13, GGHR'13, BZ'13, …] Uses
  - Public key encryption, signatures, deniable encryption, multiparty key exchange, MPC, …
- [BCPR'13, MR'13, BCP'13, …] Further Investigation

Our Results

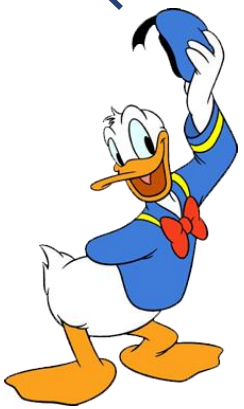
- Non-interactive multiparty key exchange without trusted setup
  - All existing protocols required trusted setup
- Efficient broadcast encryption
  - Distributed
  - Use existing keys
- Efficient traitor tracing
  - Shortest secret keys and ciphertexts known



All constructions from iO and one-way functions

(Non-Interactive) Multiparty Key Exchange

Public bulletin board

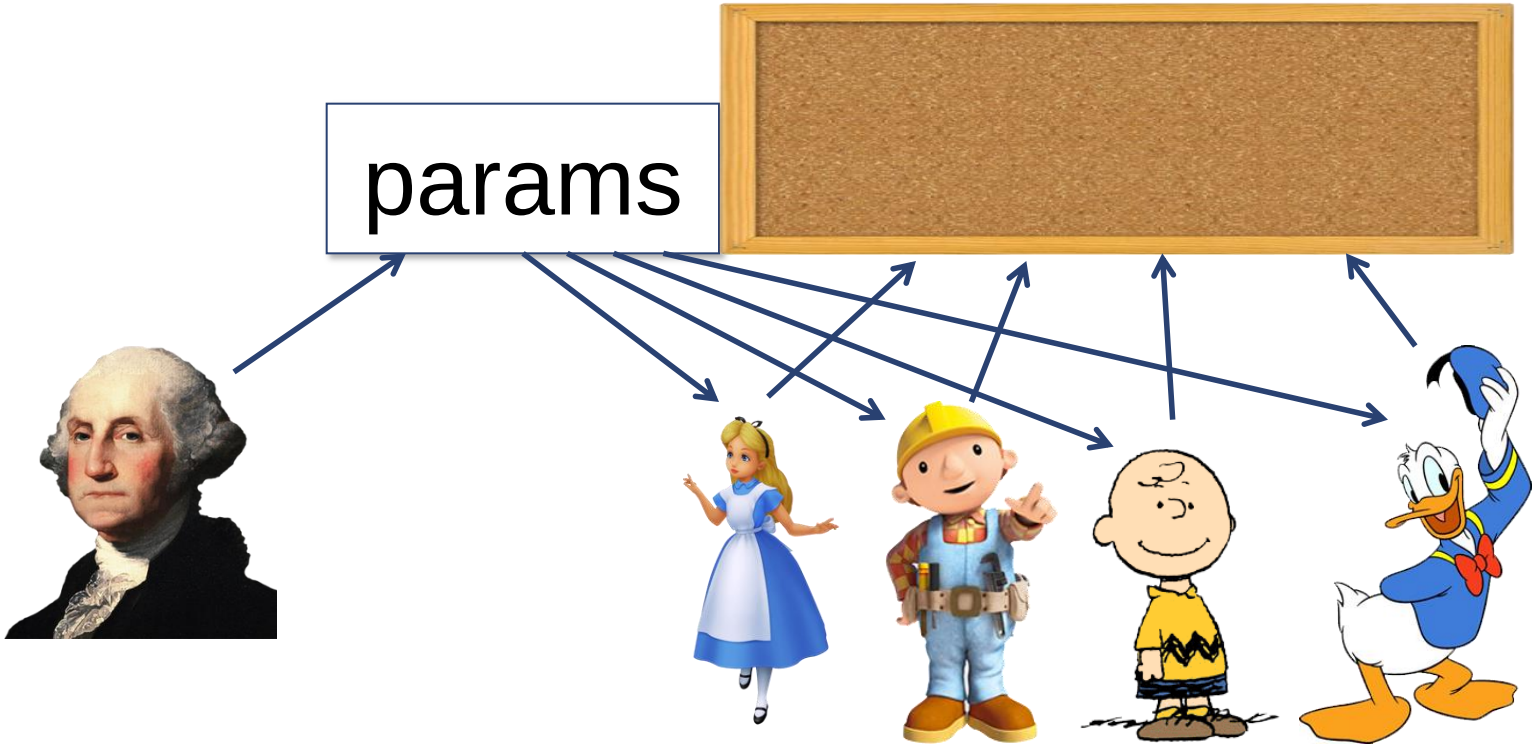


K_{ABCD} K_{ABCD} K_{ABCD} K_{ABCD}

Prior Constructions

First achieved using multilinear maps

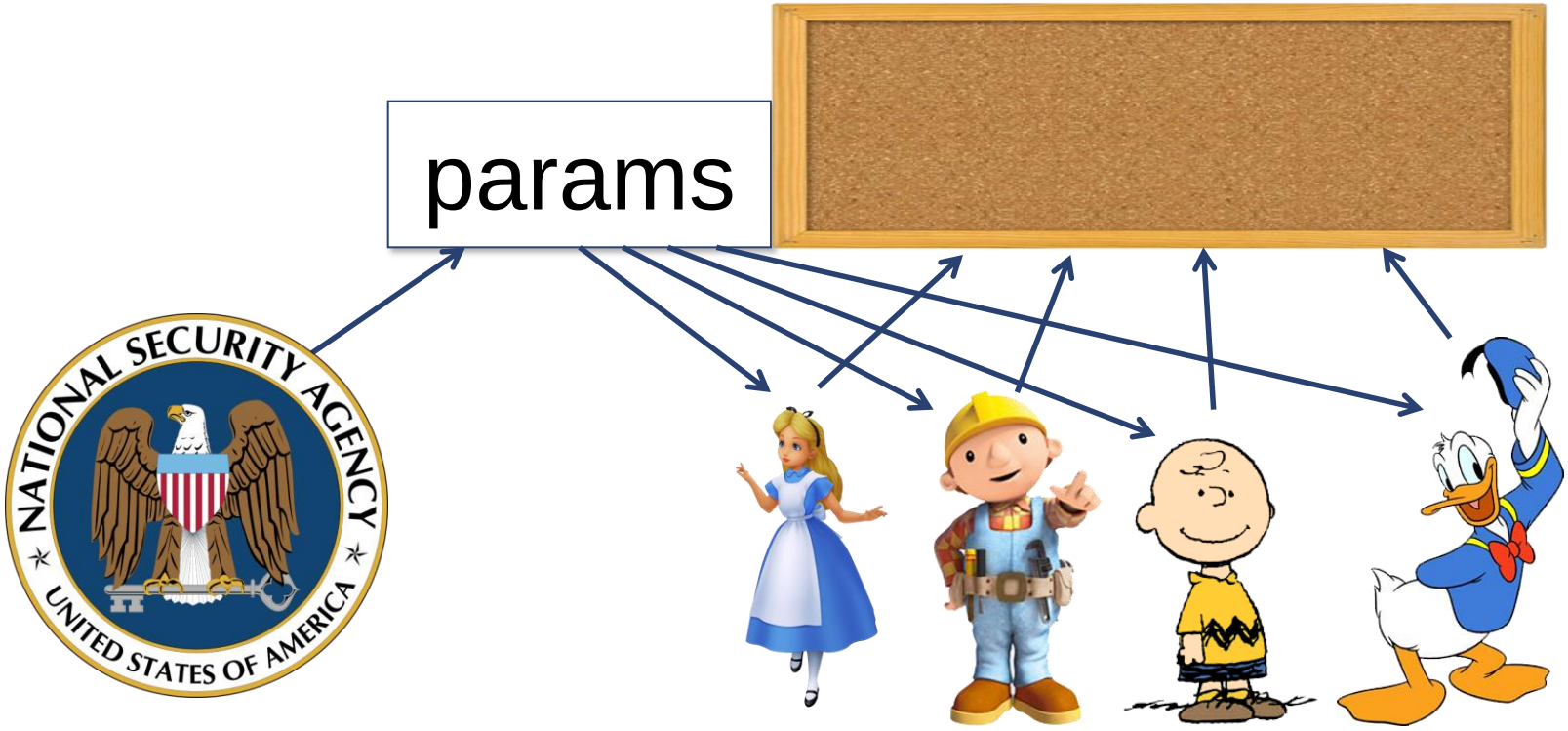
- These constructions all require trusted setup **before** protocol is run
- Trusted authority can also learn group key



Prior Constructions

First achieved using multilinear maps

- These constructions all require trusted setup **before** protocol is run
- Trusted authority can also learn group key



Our Construction (w/ Trusted Setup)

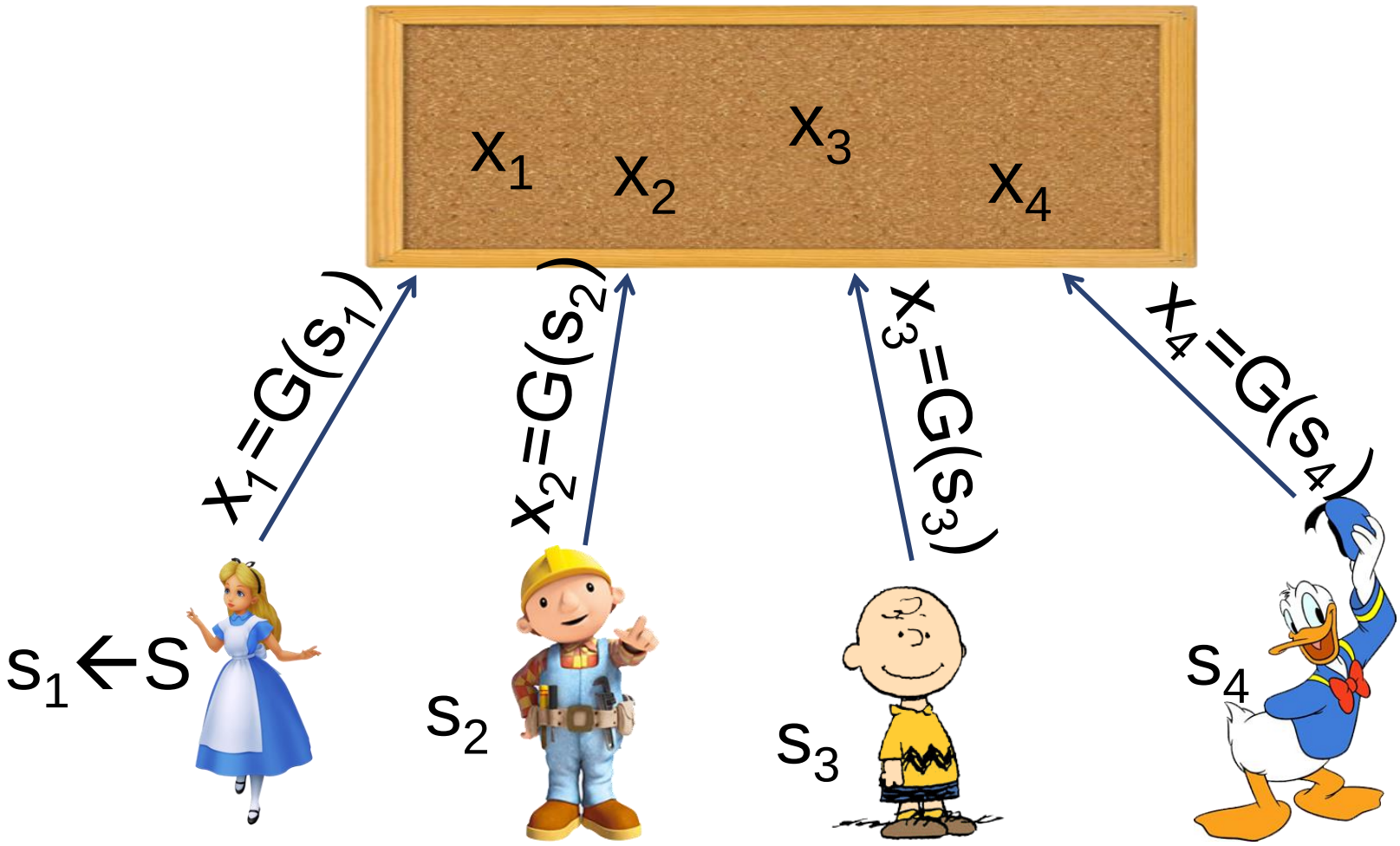
Building blocks:

- iO
- Pseudorandom function **F**
- Pseudorandom generator **G: S→X**

Idea: shared key is **F** applied to published values

- **F** itself kept secret
- Publish program that computes **F**,
  - but only if user supplies proof that they are allowed to

Our Construction (w/ Trusted Setup)



How to establish shared group key?

Our Construction (w/ Trusted Setup)

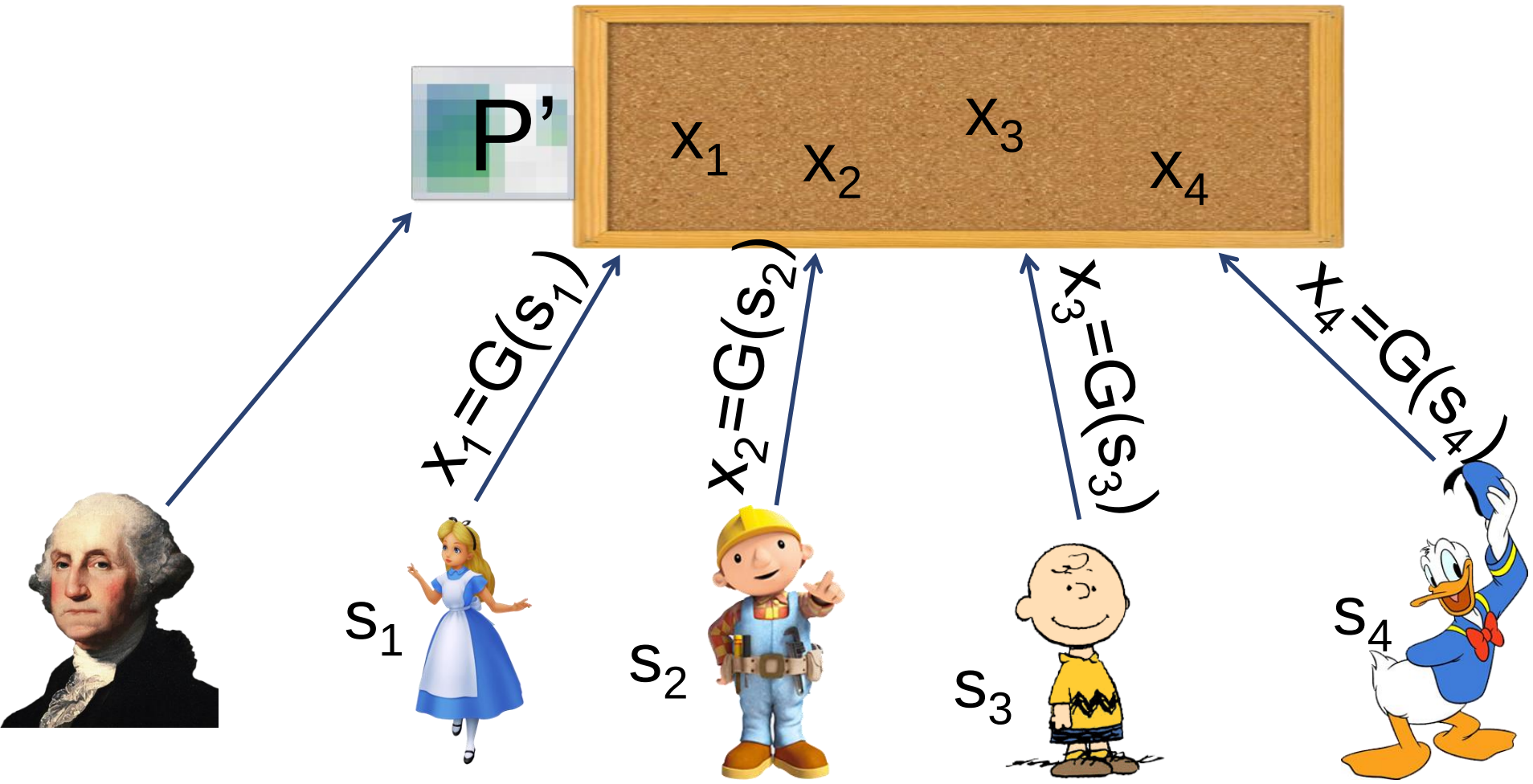
F

P(y_1, ..., y_n, s, i) {
  If G(s) ≠ y_i, output ⊥
  Otherwise, output F(y_1, ..., y_n)
}

iO

P’

Our Construction (w/ Trusted Setup)



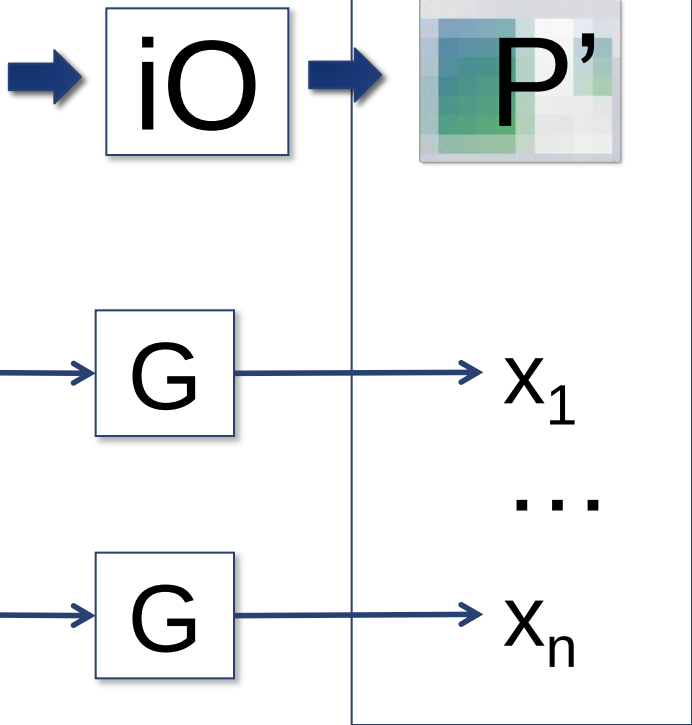
$K_{ABCD} = P'(x_1, x_2, x_3, x_4, s_1, 1)$

Security of Our Construction

Adversary sees **P'** and the $\mathbf{X_i}$, wants to learn $\mathbf{F(x_1,...,x_n)}$

F

$P(y_1, ..., y_n, s, i)$ {
  If $G(s) \neq y_i$, output $\perp$
  Otherwise, output $F(y_1, ..., y_n)$
}



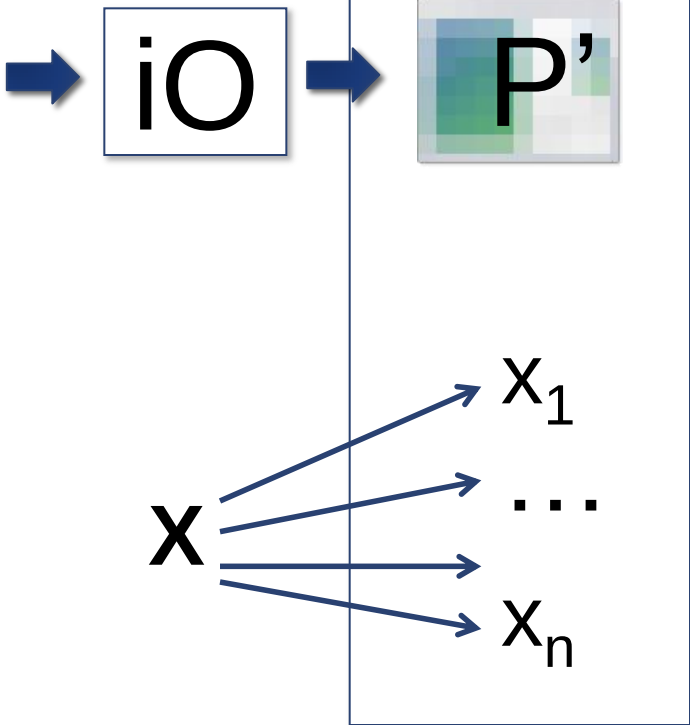
Step 1: Replace $\mathbf{x_i}$

Draw $\mathbf{x_i}$ uniformly at random

- Security of **G**: adversary cannot tell difference

F

```
P( y_1, ..., y_n, s, i ) {
    If G(s) ≠ y_i, output ⊥
    Otherwise, output F(y_1, ..., y_n)
}
```

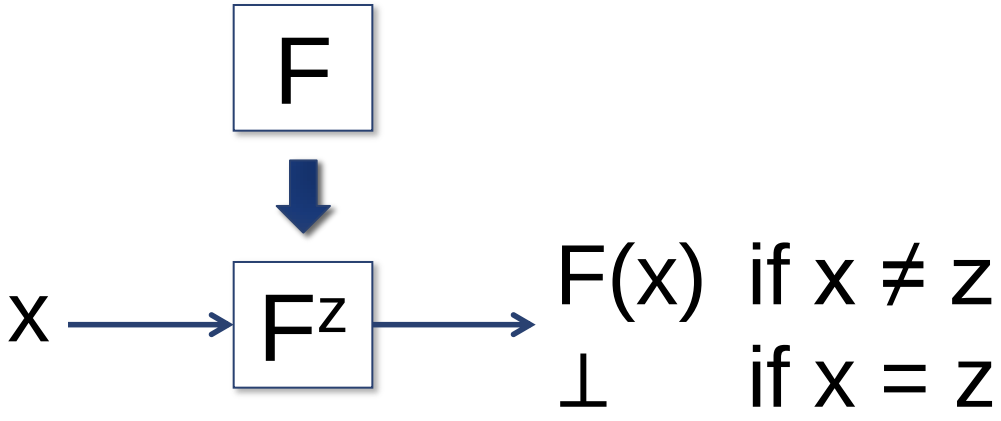


Observation: if **X** is much larger than **S**,

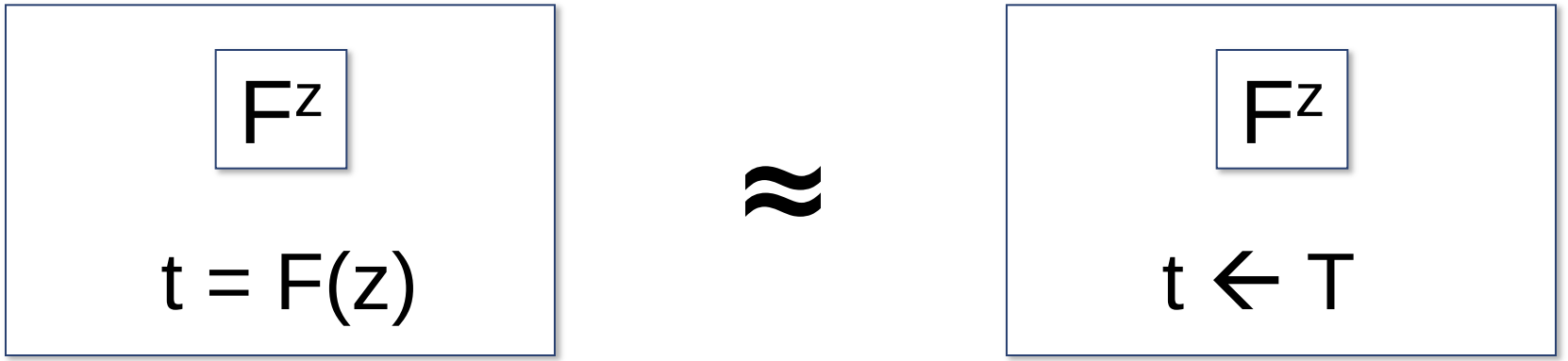
all $\mathbf{x_i}$ are outside range of **G**, w.h.p.

Punctured PRFs [BW'13, KPTZ'13, BGI'13,SW'13]

Can give out code to evaluate **F** at all but a single point **z**



Security: given $\mathbf{F^z}$, **t=F(z)** indistinguishable from random



Step 2: Puncture **F**

Let $\mathbf{z = (x_1, ..., x_n)}$

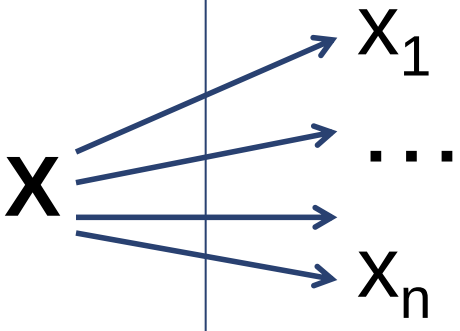
Puncture **F** at **z**, and abort if input is **z**

F^z

$P_2(y_1, ..., y_n, s, i)$ {
  If $G(s) \neq y_i$, output $\perp$
  If $(y_1, ..., y_n) = z$, output $\perp$
  Otherwise, output $F^z(y_1, ..., y_n)$
}

Inputs where $\mathbf{P_2}$ differs from **P**?

- Only $\mathbf{(x_1,...,x_n,s,i)}$ where $\mathbf{G(s) = x_i}$
- W.h.p. no such input exists
- iO: $\mathbf{P_2}$ indistinguishable from **P**

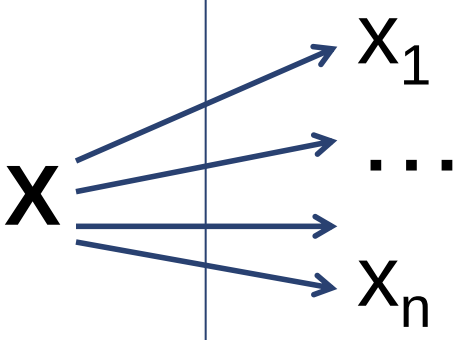


Step 3: Simulate

Simulate view of adversary, given $\mathbf{F^z}$

F^z

$P_2(y_1, ..., y_n, s, i) \{$
If $G(s) \neq y_i$, output $\perp$
If $(y_1, ..., y_n) = z$, output $\perp$
Otherwise, output $F^z(y_1, ..., y_n)$
$\}$



Security of **F**: **k = F(z)** indist.
from a random key

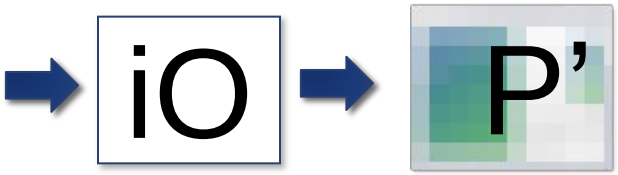
Removing Trusted Setup

As described, our scheme needs trusted setup

Observation: Obfuscated program can be generated independently of publishing step

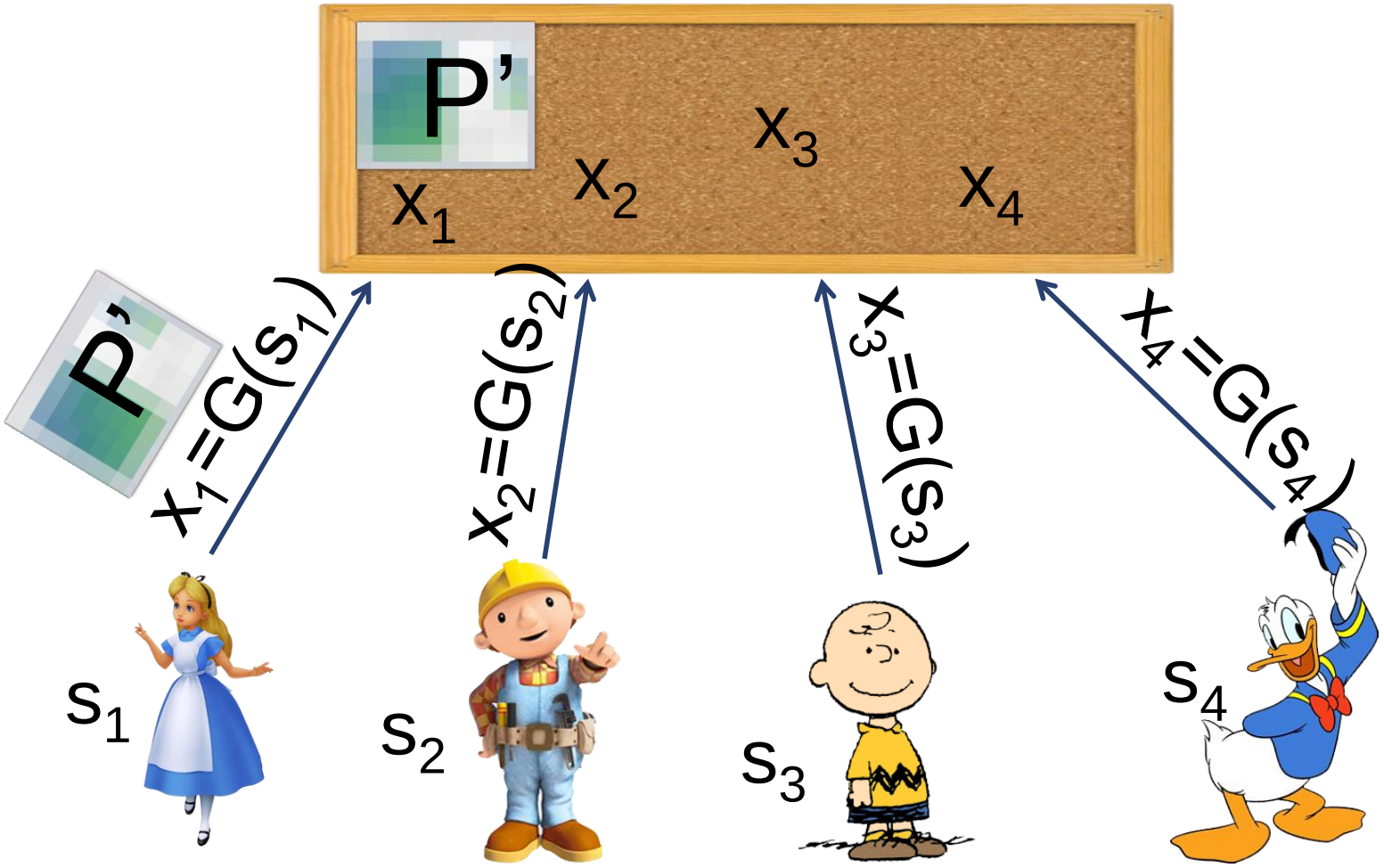
F

$P(y_1, ..., y_n, s, i) \{$
  If $G(s) \neq y_i$, output $\perp$
  Otherwise, output $F(y_1, ..., y_n)$
$\}$

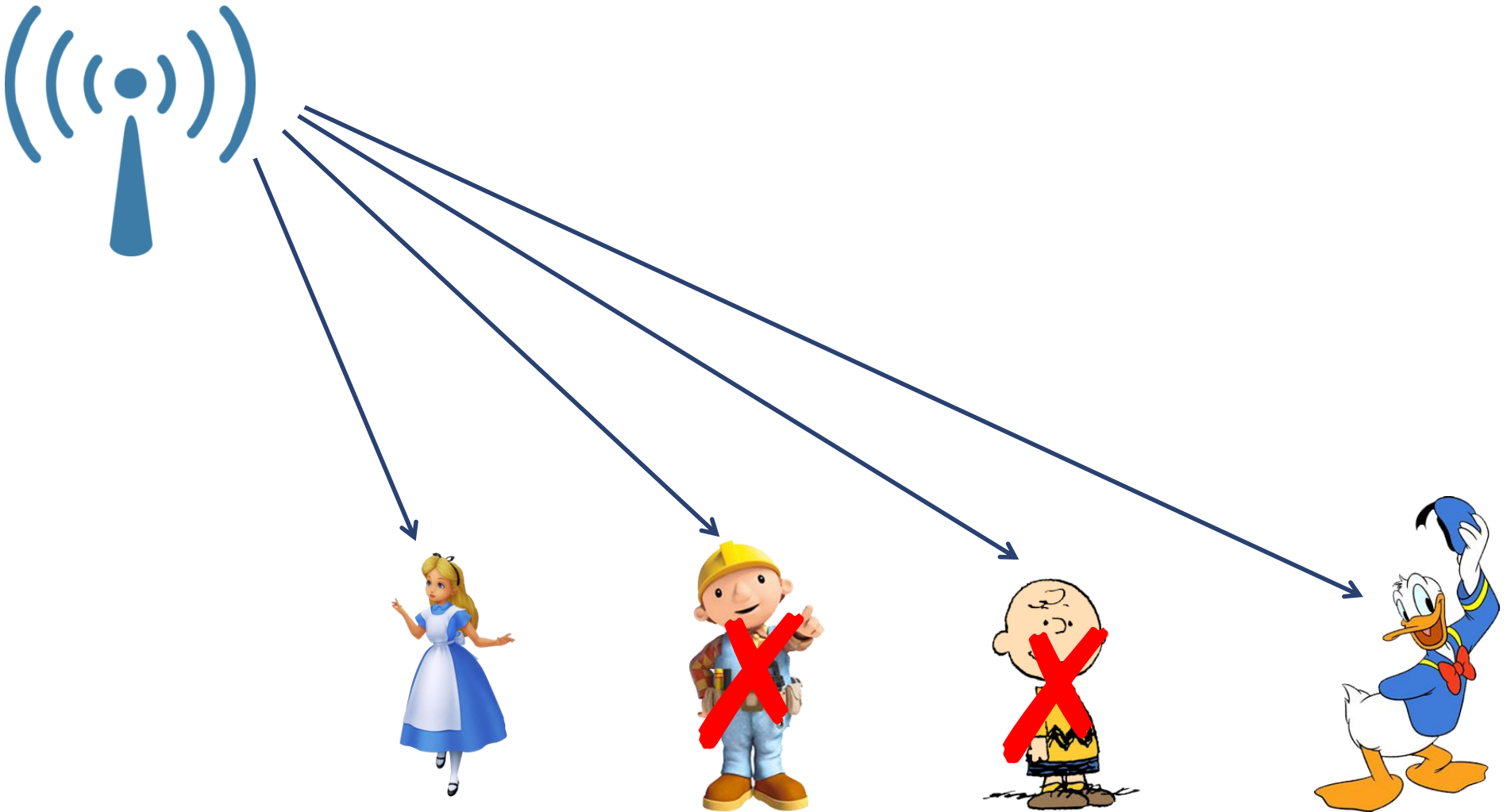


Untrusted setup: user 1 generates **P’**, sends with $\mathbf{x_1}$

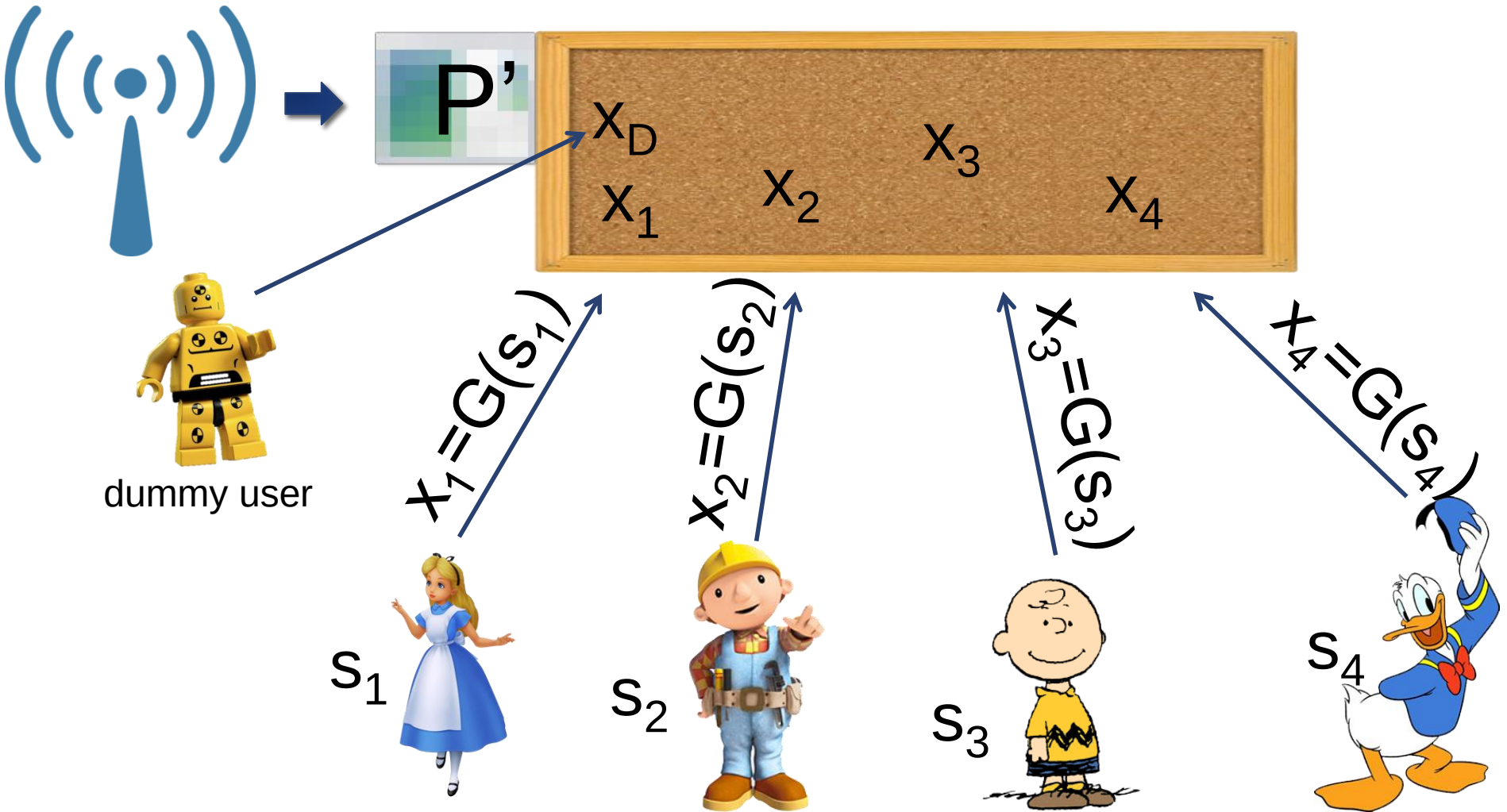
Multiparty Key Exchange Without Trusted Setup



Broadcast Encryption

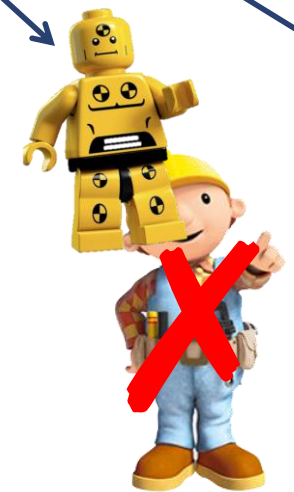


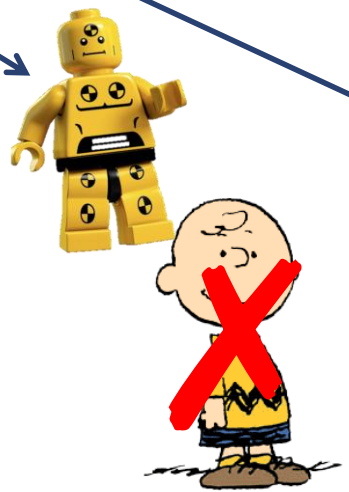
Broadcast Encryption

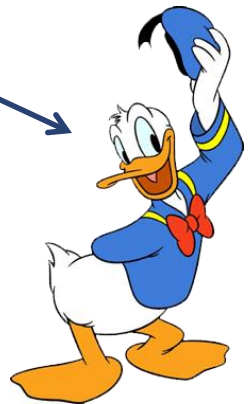


Broadcast Encryption

- Replace unintended recipients with dummy
- Compute shared key for protocol
  - Ex: $\mathbf{k = F(x_1, x_D, x_D, x_4)}$
- Use shared key to encrypt message







Broadcast Encryption

Private key scheme: empty ciphertext header

Public broadcast key scheme: a single $\mathbf{x}_i$ value

Additional Properties:

- Distributed – users and broadcaster each generate their own parameters
- Can be used with existing RSA keys (under plausible assumptions)

Other Constructions

Recipient private broadcast encryption

- Ciphertext size: **λ+n**
- Secret key size: **λ**
- Public key size: **poly(n, λ)**

Traitor tracing

- Ciphertext size: **λ+log(n)**
- Secret key size: **λ**
- Public key size: **poly(log(n), λ)**

Open Questions

Reduce public key sizes

- Using *differing-inputs* obfuscation [ABGSZ'13]
- From iO?

Other primitives from iO

- FHE?

Thanks!