

# Functional Encryption: Deterministic to Randomized Functions from Simple Assumptions

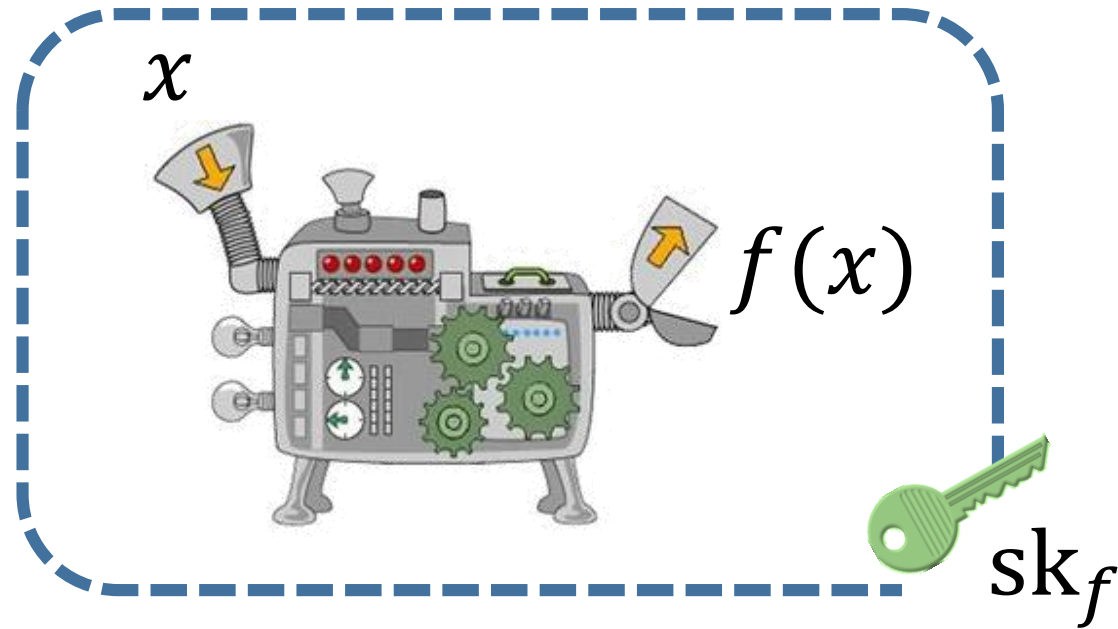
Shashank Agrawal

**VISA**  
Research

David J. Wu

**Stanford**  
University

# Public-Key Functional Encryption [BSW11, O'N10]



Keys are associated with deterministic functions  $f$



# Public-Key Functional Encryption [BSW11, O'N10]

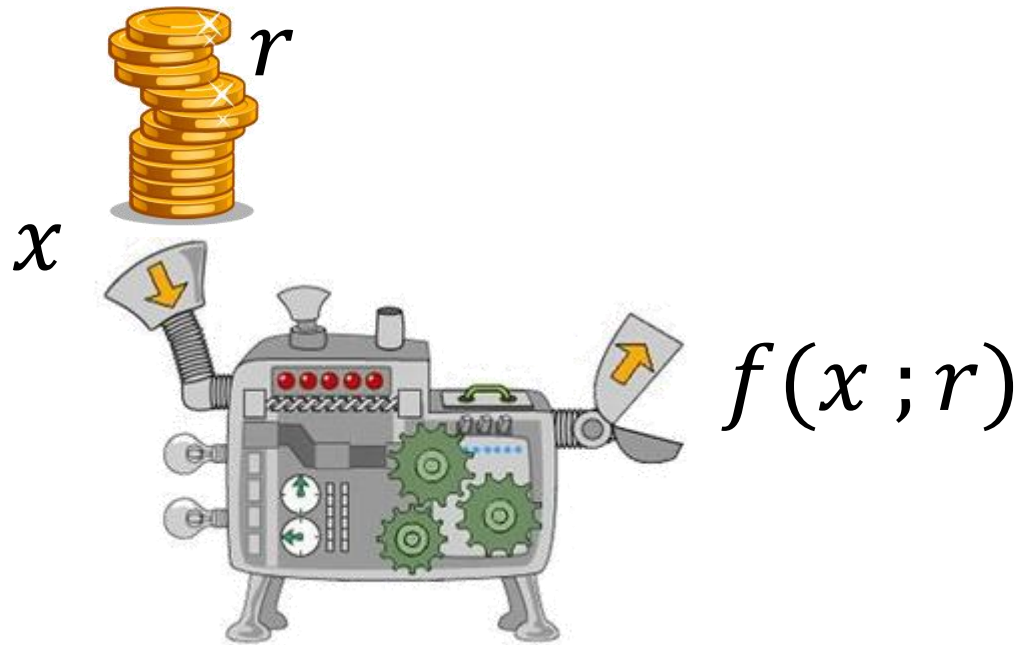
- $\text{Setup}(1^\lambda)$ : Outputs  $(\text{msk}, \text{mpk})$
- $\text{KeyGen}(\text{msk}, f)$ : Outputs decryption key  $\text{sk}_f$
- $\text{Encrypt}(\text{mpk}, m)$ : Outputs ciphertext  $\text{ct}_m$
- $\text{Decrypt}(\text{sk}_f, \text{ct}_m)$ : Outputs  $f(m)$

# Public-Key Functional Encryption [BSW11, O'N10]

- $\text{Setup}(1^\lambda)$ : Outputs  $(\text{msk}, \text{mpk})$
- $\text{KeyGen}(\text{msk}, f)$ : Outputs decryption key  $\text{sk}_f$
- $\text{Encrypt}(\text{mpk}, m)$ : Outputs ciphertext  $\text{ct}_m$
- $\text{Decrypt}(\text{sk}_f, \text{ct}_m)$ : Outputs  $m$

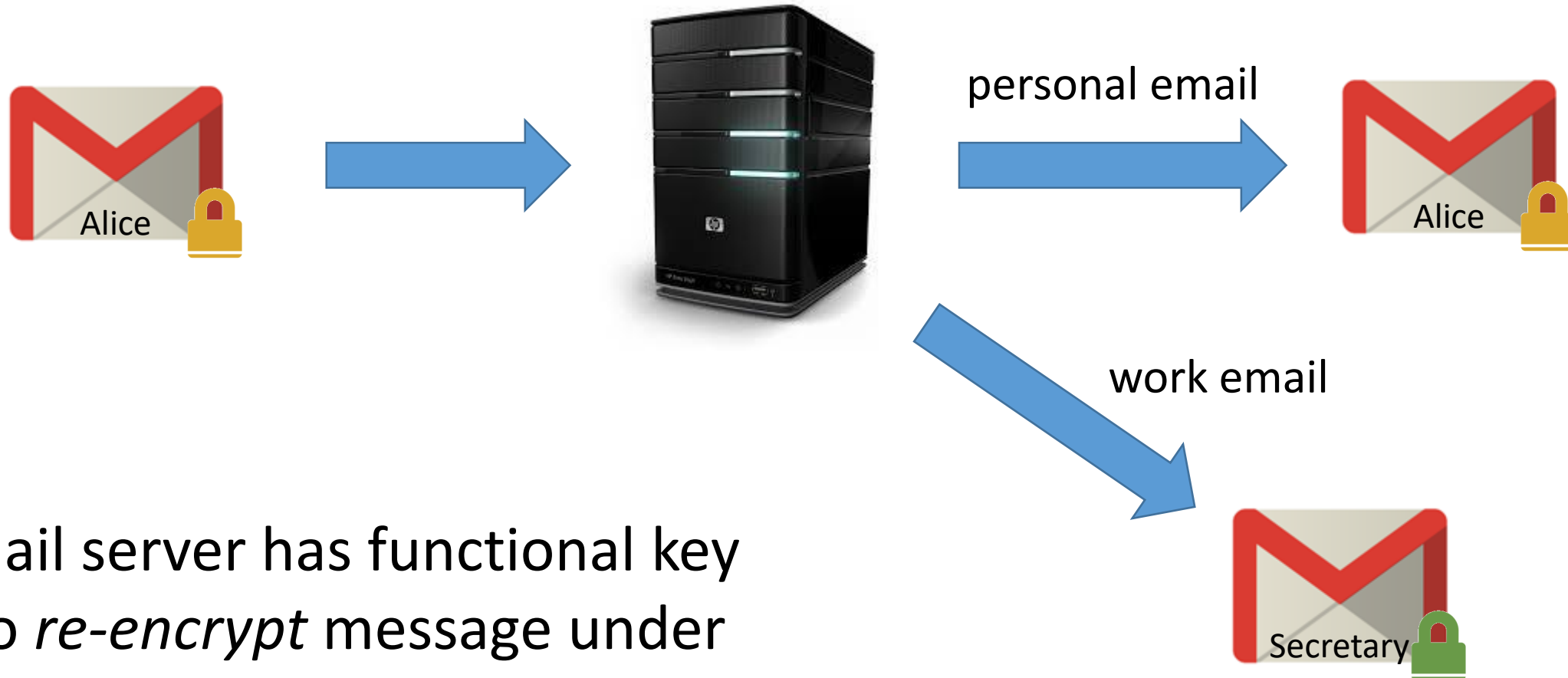
Deterministic  
function  $f$

# Functional Encryption for Randomized Functionalities (rFE) [ABFGGTW13, GJKS15]



Many interesting functions are  
*randomized*

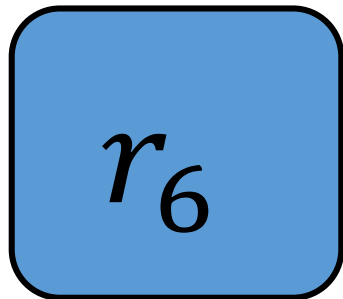
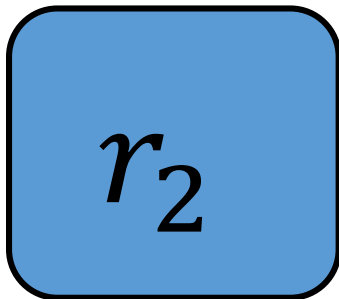
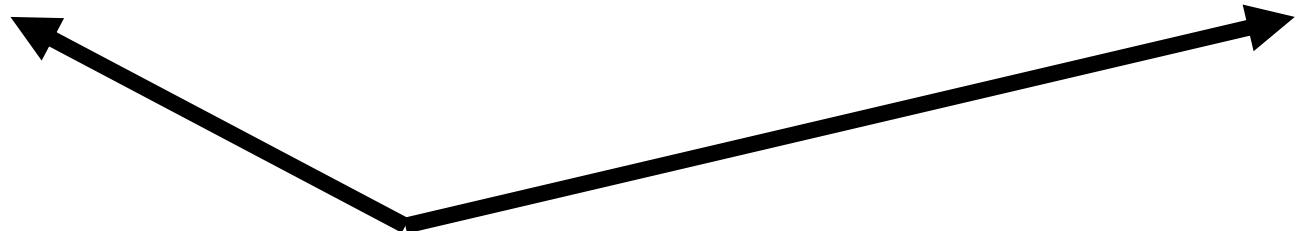
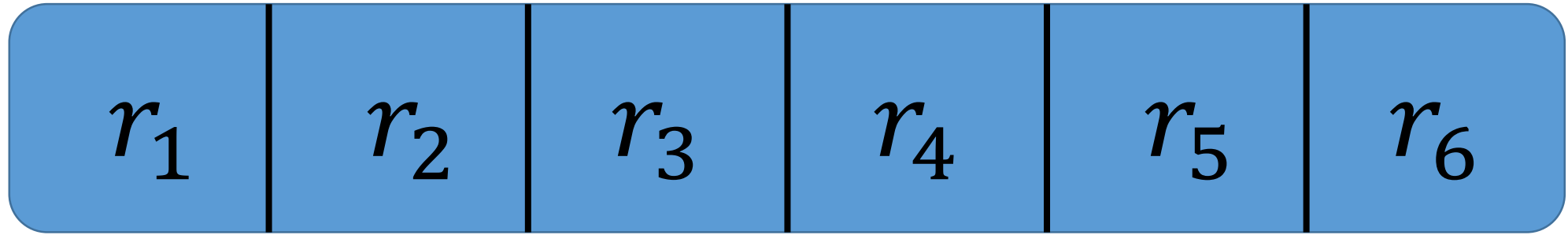
# Application 1: Proxy Re-Encryption



Mail server has functional key  
to *re-encrypt* message under  
secretary's public key

# Application 2: Auditing an Encrypted Database

Encrypted database of records



Sample a *random* subset to audit

# Does Public-Key rFE Exist?





# Public-Key Functional Encryption [BSW11, O'N10]

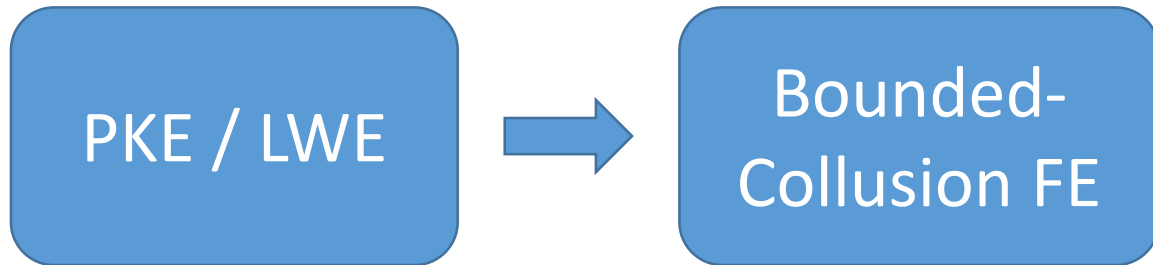
Can be instantiated from a wide range of assumptions



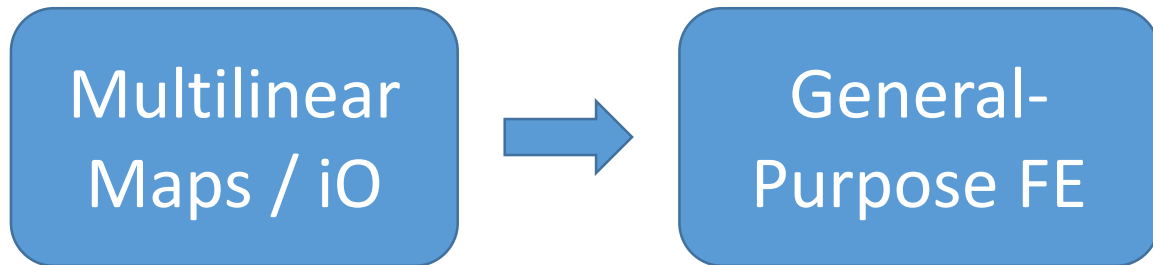
# The Landscape of (Public-Key) Functional Encryption

Deterministic functionalities

[SS10, GVW12, ...]



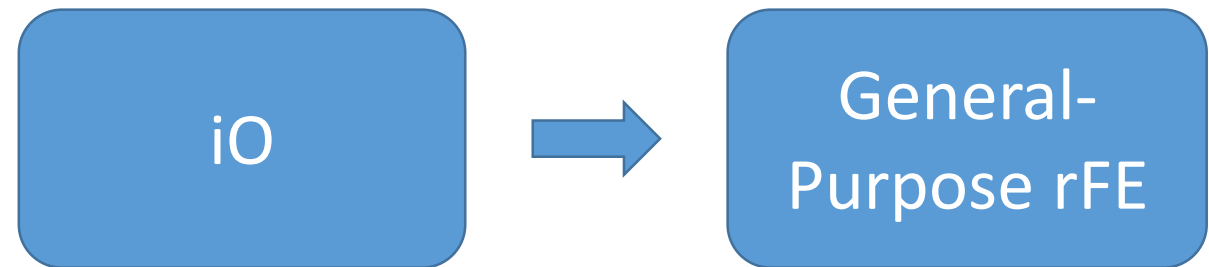
[GGHRSW13, GGHZ16, ...]



Generally adaptively  
secure

Randomized functionalities

[GJKS15]



Selectively secure

# The Landscape of (Public-Key) Functional Encryption

Deterministic functionalities

Randomized functionalities

PKE / LWE

*Does extending FE to support randomized functionalities require much stronger tools?*

General-purpose rFE

Multilinear  
Maps / i

Generally adaptively  
secure

Selectively secure

# Our Main Result

General-purpose FE  
for deterministic  
functionalities

Number Theory

(e.g., DDH, RSA)

General-purpose FE  
for randomized  
functionalities

**Implication:** *randomized FE is not much more difficult to construct than standard FE.*

Defining rFE

# Correctness for FE

Deterministic functions



# Correctness for rFE [ABFGGTW13, GJKS15]

Randomized functions



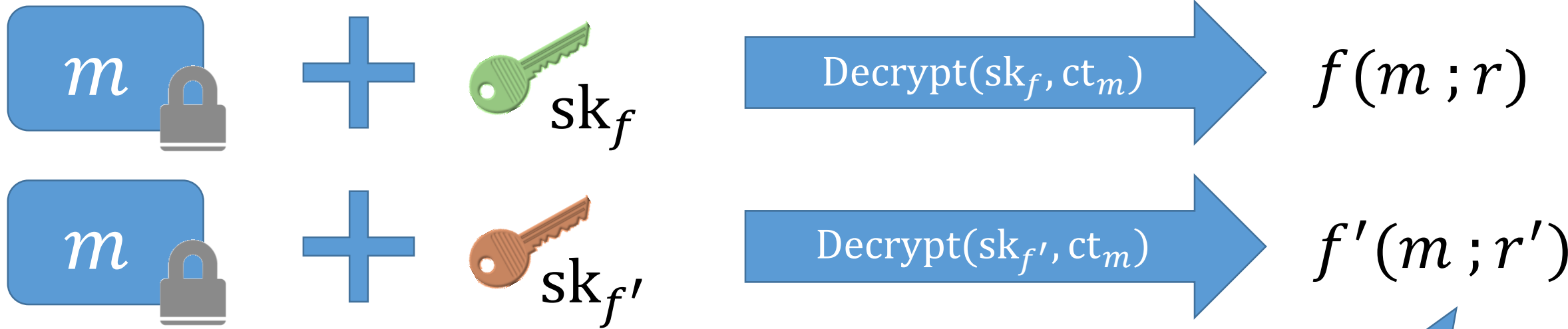
Different  
ciphertexts

Same function  
key

Independent draws  
from output  
distribution

# Correctness for rFE [ABFGGTW13, GJKS15]

Randomized functions



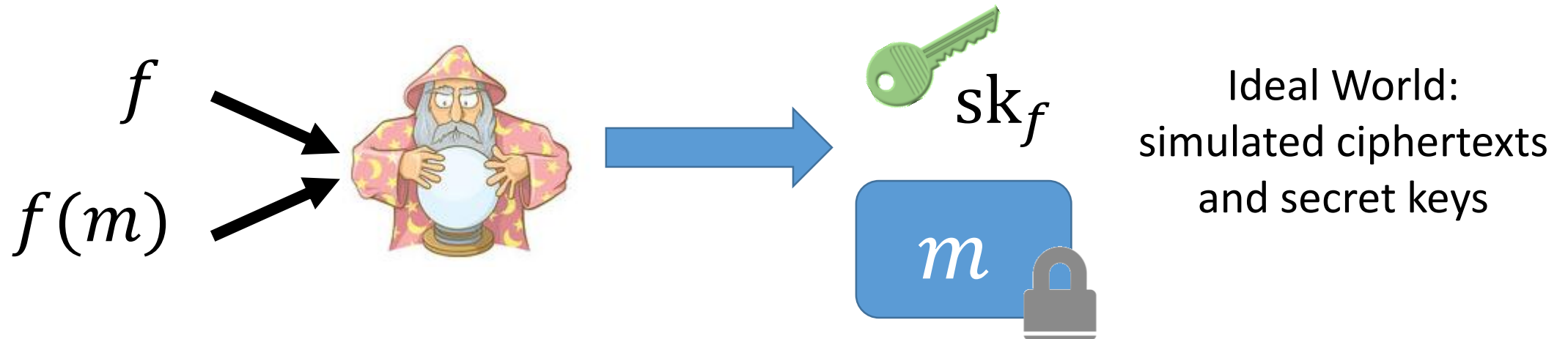
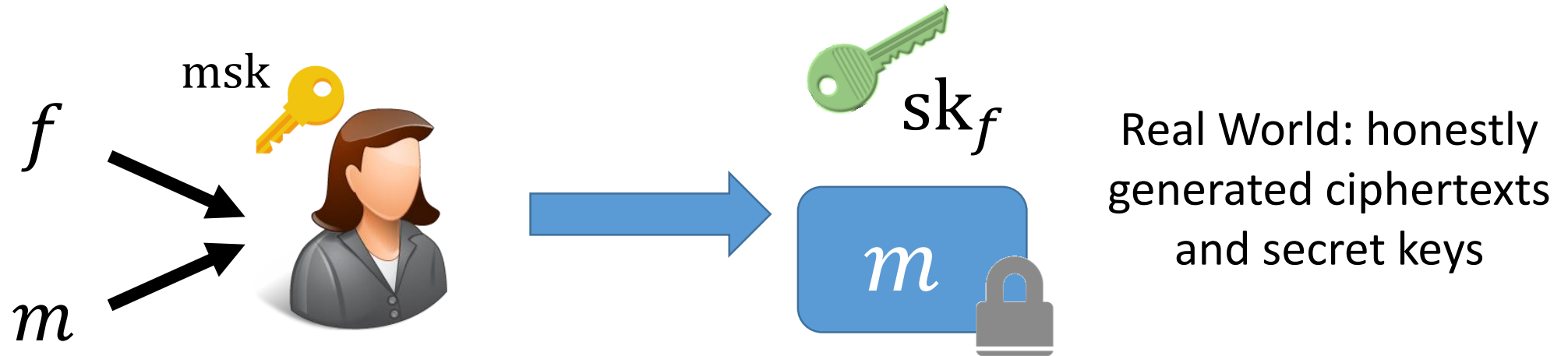
Same  
ciphertexts

Different  
function keys

Independent draws  
from output  
distribution

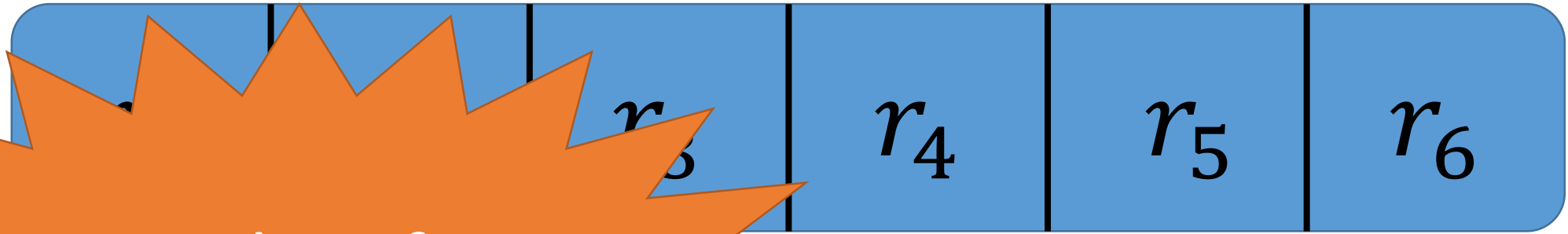


# Simulation-Based Security (Informally)

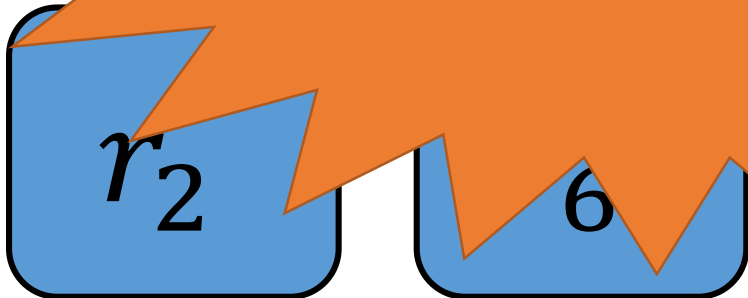


# The Case for Malicious Encrypters [GJKS15]

Encrypted database of records



What if  
encrypter (bank)  
is adversarial?

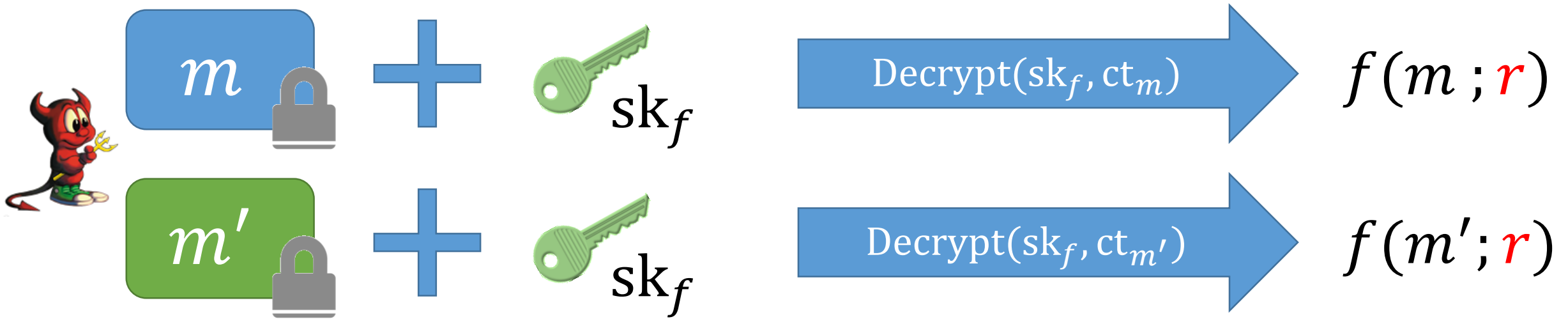


Sample a *random*  
subset to audit



# The Case for Malicious Encrypters [GJKS15]

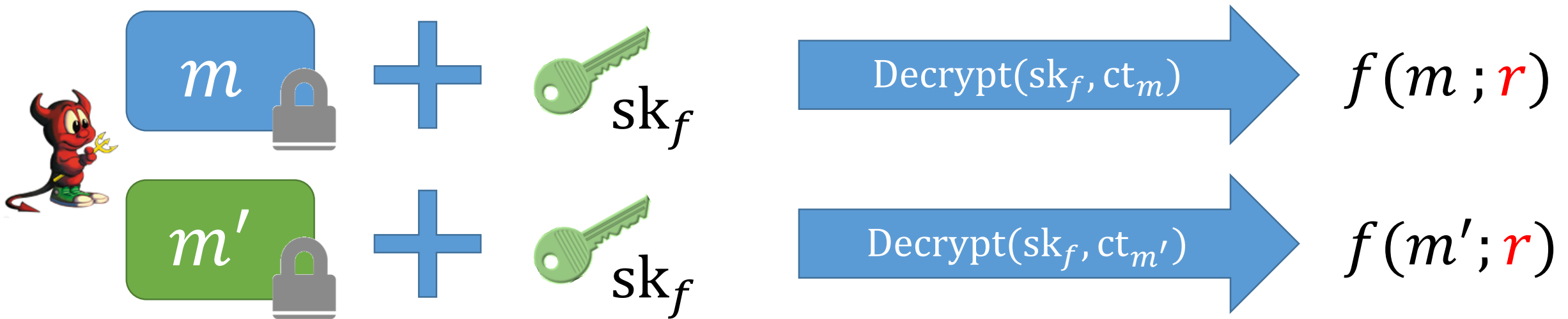
Randomized functionalities



Dishonest encrypters can construct “bad” ciphertexts such that decryption produces *correlated* outputs

# The Case for Malicious Encrypters [GJKS15]

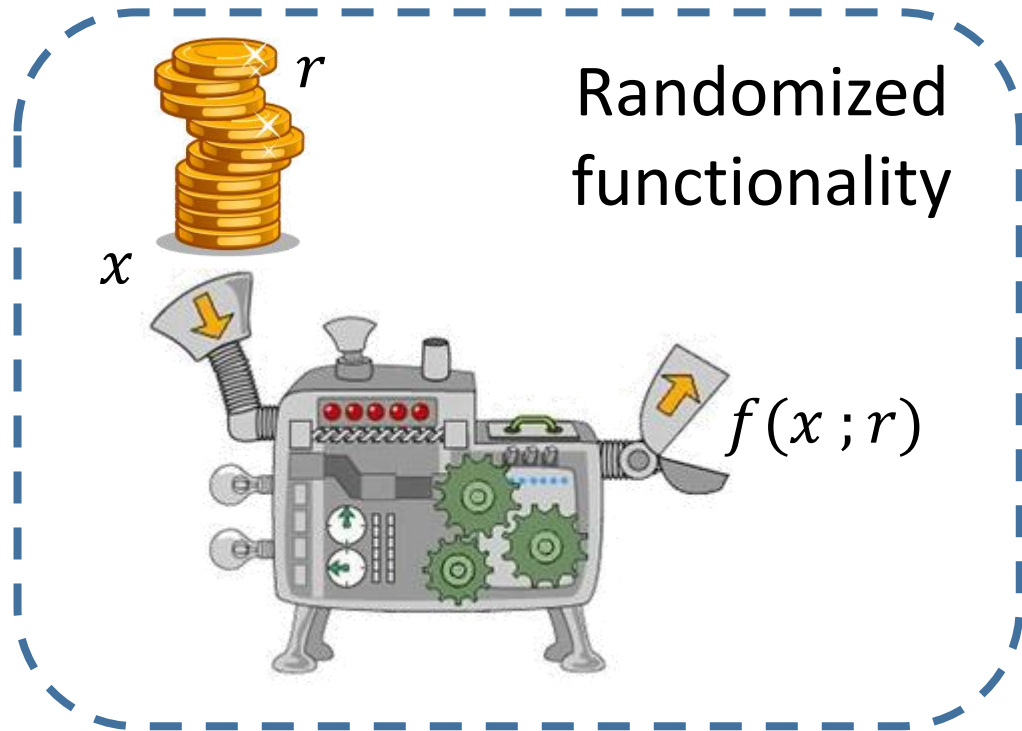
Randomized functionalities



Formally captured by giving adversary access to a decryption oracle (like in the CCA-security game). [See paper for details.]

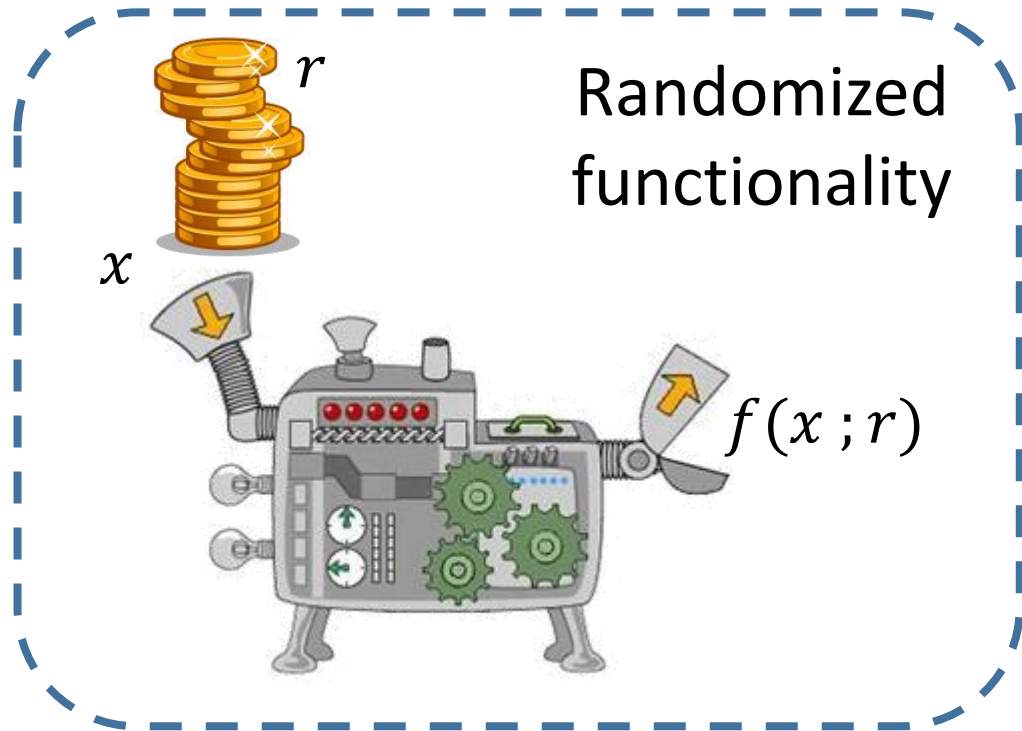
Our Generic Transformation

# Starting Point: Derandomization

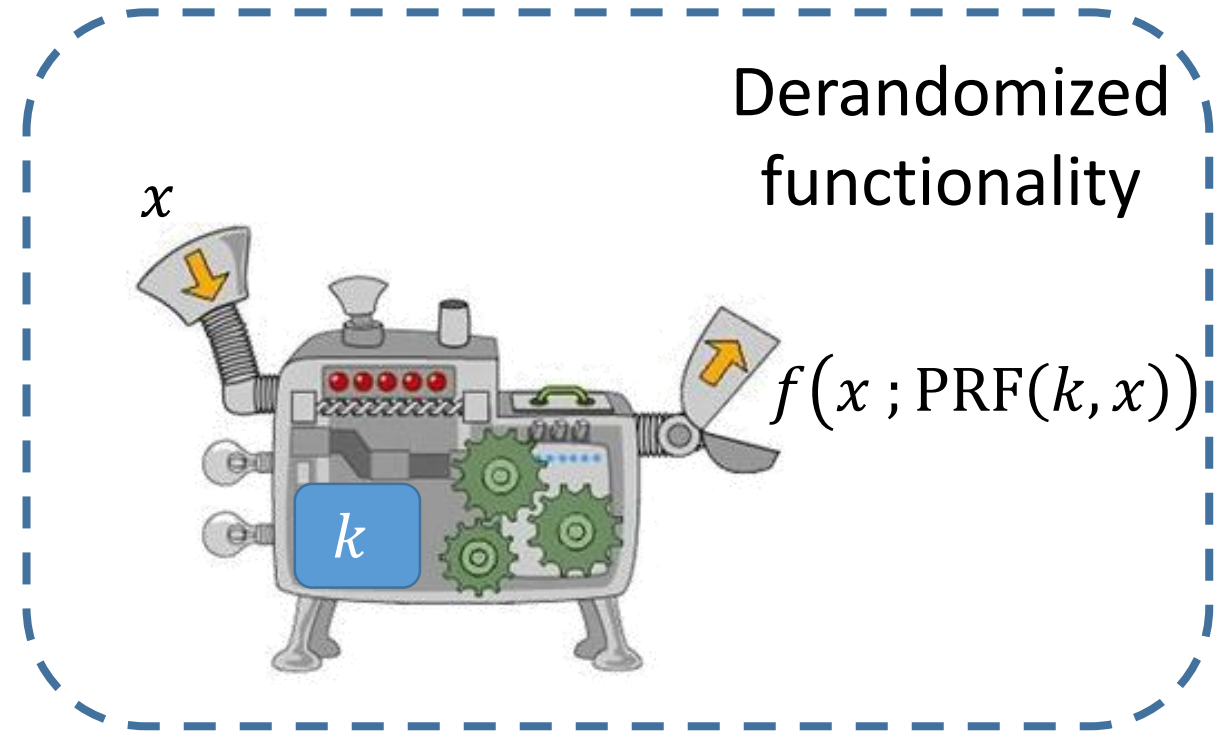


**Starting point:** construct “derandomized function” where randomness for  $f$  derived from outputs of a PRF

# Starting Point: Derandomization



Randomized function  $f$

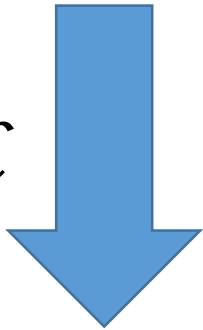


Derandomized function  $g_k$ :  
 $g_k(x) = f(x; \text{PRF}(k, x))$

# Starting Point: Derandomization

rFE. KeyGen(msk,  $f$ )

$k \stackrel{R}{\leftarrow} \mathcal{K}$



FE. KeyGen(msk,  $g_k$ )

$g_k(x) = f(x; \text{PRF}(k, x))$



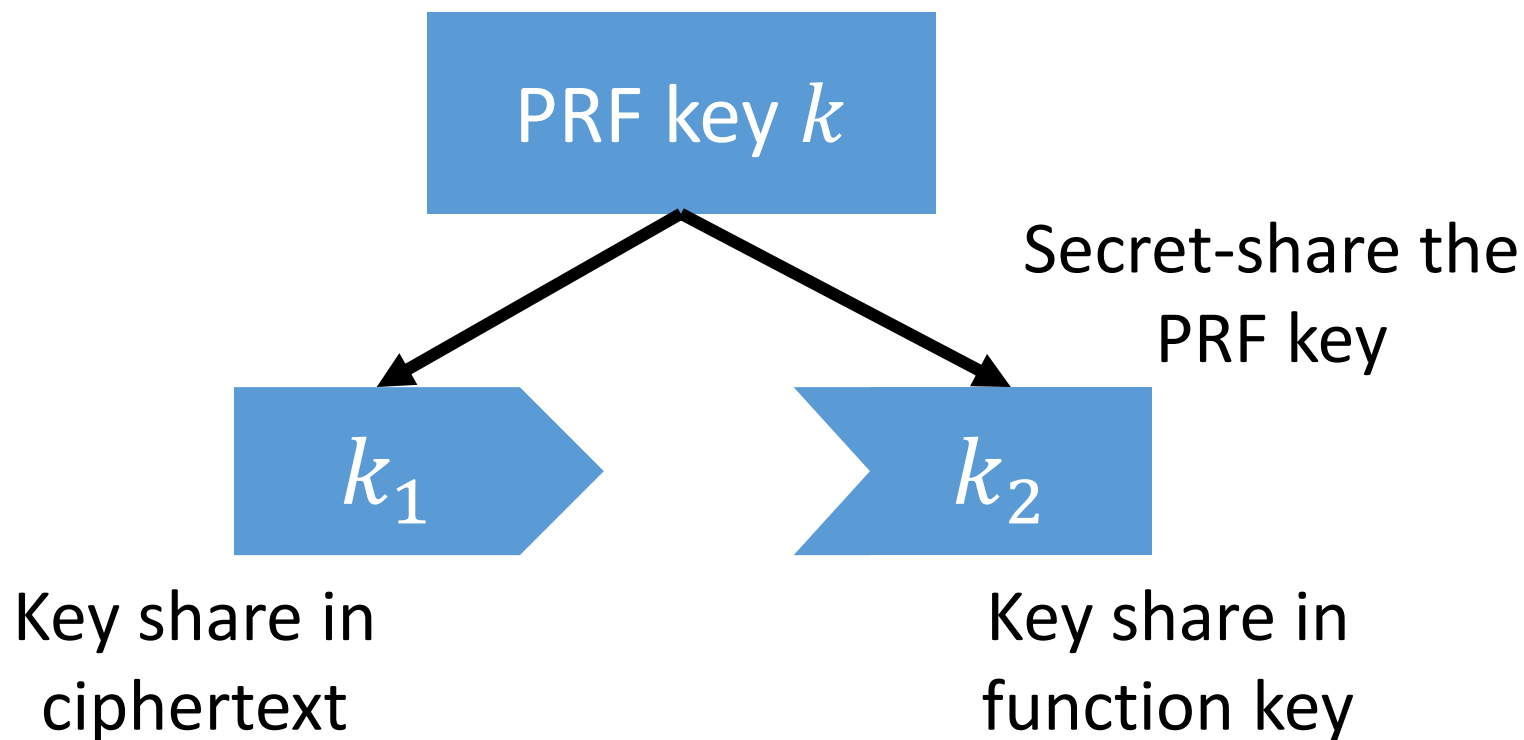
  $sk_{g_k}$

But in public-key setting, keys do not hide the function!



# How to Hide the Key?

**Key idea:** functional encryption provides message-hiding, so place part of the key in the ciphertext



# How to Hide the Key?

**Key idea:** functional encryption provides message-hiding, so place part of the key in the ciphertext

rFE. Encrypt(mp<sub>k</sub>,  $m$ )

$k_1 \stackrel{R}{\leftarrow} \mathcal{K}$

FE. Encrypt(mp<sub>k</sub>, ( $m, k_1$ ))

( $m, k_1$ )



# How to Hide the Key?

**Key idea:** functional encryption provides message-hiding, so place part of the key in the ciphertext

rFE. KeyGen(msk, f)

$$k_2 \stackrel{R}{\leftarrow} \mathcal{K}$$

Some operation to combine shares of key

$$g_{k_2}(m, k_1) = f(m; \text{PRF}(k_1 \diamond k_2, m))$$

FE. KeyGen(msk,  $g_{k_2}$ )



# How to Hide the Key?

**Key idea:** functional encryption provides message-hiding, so place part of the key in the ciphertext

rFE. KeyGen(msk, f)

$$k_2 \stackrel{R}{\leftarrow} \mathcal{K}$$

FE. KeyGen(msk,  $g_{k_2}$ )

Encrypter controls  $k_1$   
so we need related-key security

$$g_{k_2}(m, k_1) = f(m ; \text{PRF}(k_1 \diamond k_2, m))$$

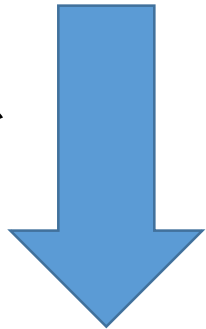


# Security Against Dishonest Encrypters

Encrypter has a lot of flexibility in constructing ciphertexts:

$\text{rFE. Encrypt}(\text{mpk}, m)$

$k_1 \stackrel{R}{\leftarrow} \mathcal{K}$



Encrypter can choose the key-share

Cannot influence output distribution due to RKA-security

$\text{FE. Encrypt}(\text{mpk}, (m, k_1))$

Encrypter can choose the randomness



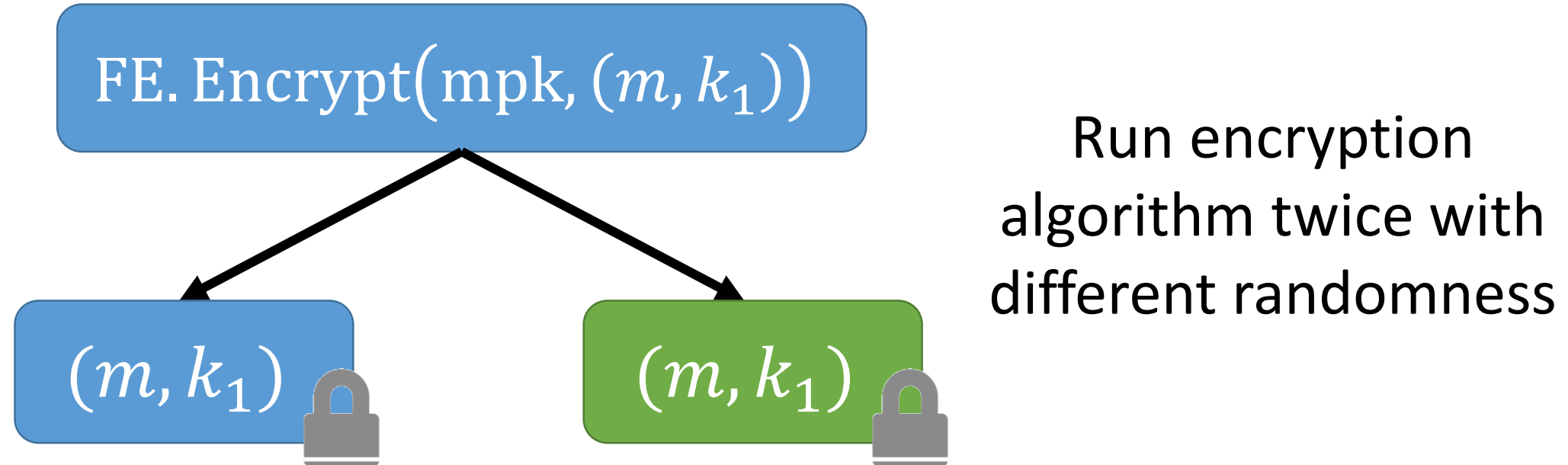
$(m, k_1)$



Potentially problematic

# Security Against Dishonest Encrypters

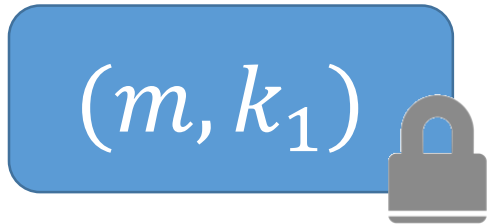
Encrypter has a lot of flexibility in constructing ciphertexts:



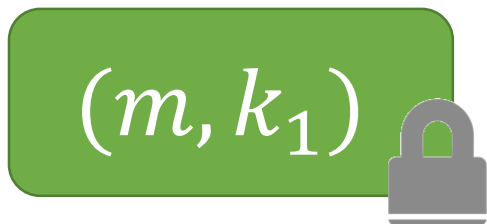
Two *distinct* FE ciphertexts encrypting the *same* message

# Security Against Dishonest Encrypters

Encrypter has a lot of flexibility in constructing ciphertexts:



**Reality:** Decryption always produces *same* output



**Desired:** Two different ciphertexts, so should produce independent outputs

Encrypter has too much freedom in constructing ciphertexts

# Applying Deterministic Encryption

**Key observation:** honestly generated ciphertexts have high entropy

Should be random PRF  
key – high entropy!

$(m, k_1)$



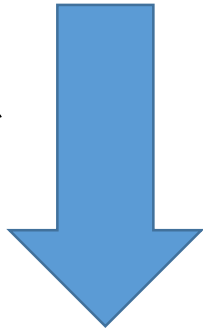
Derive encryption  
randomness from  $k_1$  and  
include a NIZK argument that  
ciphertext is well-formed



# Putting the Pieces Together

$\text{rFE. Encrypt}(\text{mpk}, m)$

$k_1 \stackrel{R}{\leftarrow} \mathcal{K}$



$\text{FE. Encrypt}(\text{mpk}, (m, k_1); h(k_1))$

Randomness for FE encryption derived from deterministic function on  $k_1$

NIZK argument of knowledge of  $(m, k_1)$  that explains ciphertext

+

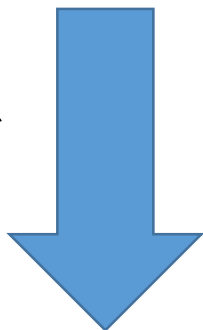
$\pi$

[See paper for full details.]

# Putting the Pieces Together

rFE. Encrypt(mp<sub>k</sub>,  $m$ )

$k_1 \stackrel{R}{\leftarrow} \mathcal{K}$



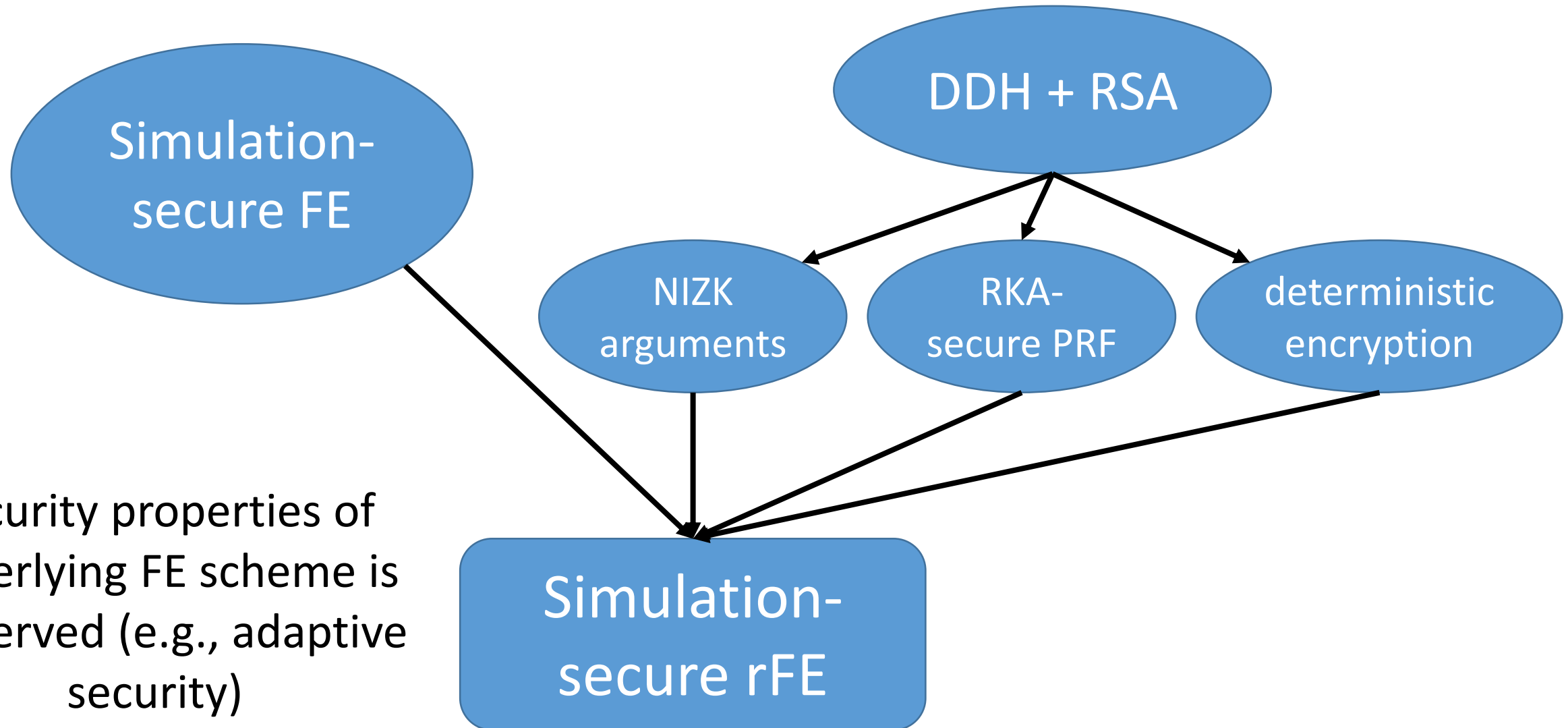
FE. Encrypt(mp<sub>k</sub>,  $(m, k_1) ; h(k_1)$ )

+

$\pi$

Ciphertext is a deterministic function of  $(m, k_1)$  so for *any* distinct pairs  $(m, k_1), (m', k'_1)$ , outputs of PRF uniform and independently distributed by RKA-security

# Our Transformation in a Nutshell

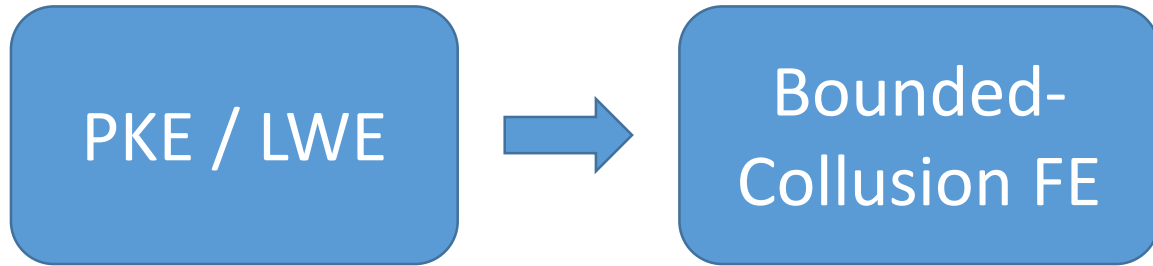


Security properties of underlying FE scheme is preserved (e.g., adaptive security)

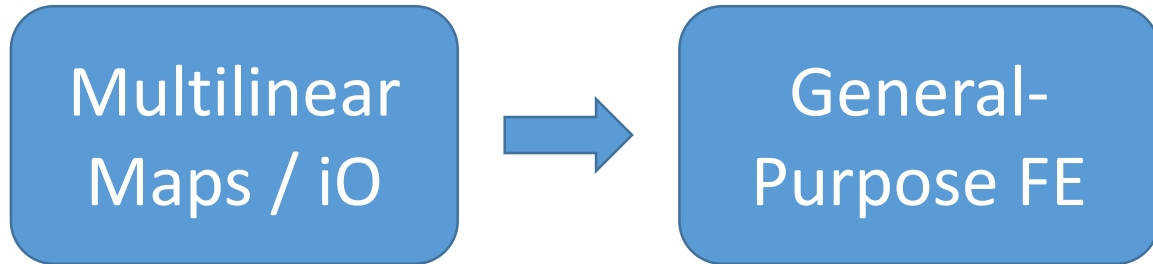
# The State of (Public-Key) Functional Encryption

Deterministic functionalities

[SS10, GVW12, ...]



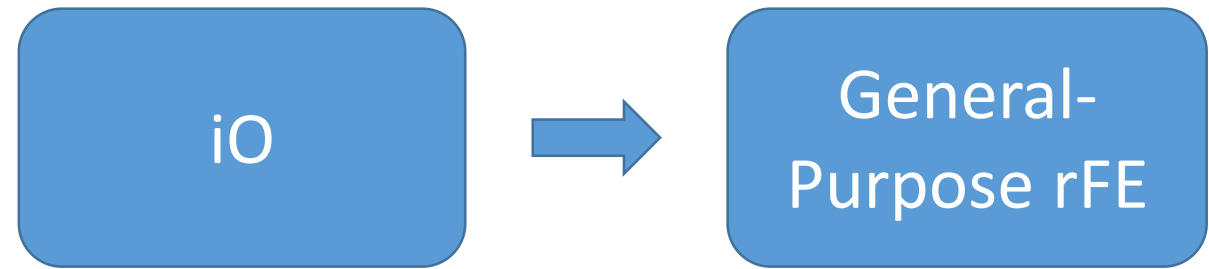
[GGHRSW13, GGHZ16, ...]



Generally adaptively  
secure

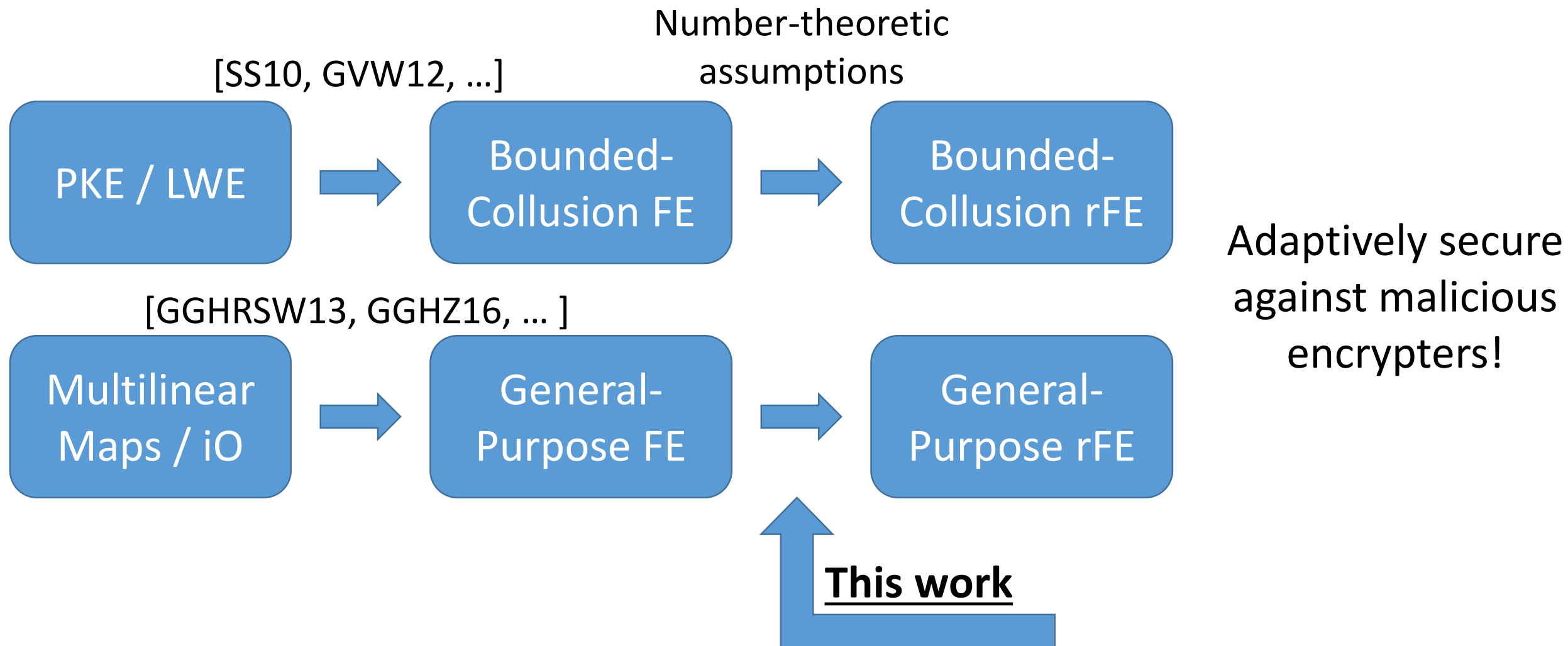
Randomized functionalities

[GJKS15]



Selectively secure

# The State of (Public-Key) Functional Encryption



# Open Questions

More direct / efficient constructions of rFE for simpler classes of functionalities (e.g., sampling random entries from a vector)?

Generic construction of rFE from FE *without* making additional assumptions?

Generic transformation for indistinguishability-based notions of security?

**Thank you!**

<http://eprint.iacr.org/2016/482>