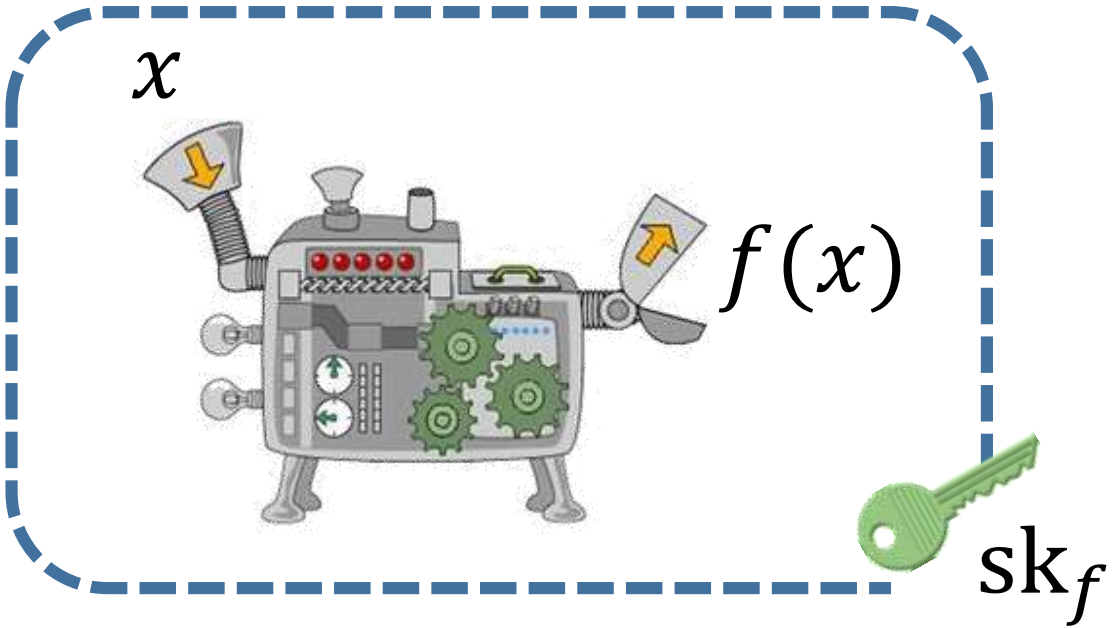
Functional Encryption: Deterministic to Randomized Functions from Simple Assumptions

Shashank Agrawal and David J. Wu

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



Keys are associated with <u>deterministic</u> functions f

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

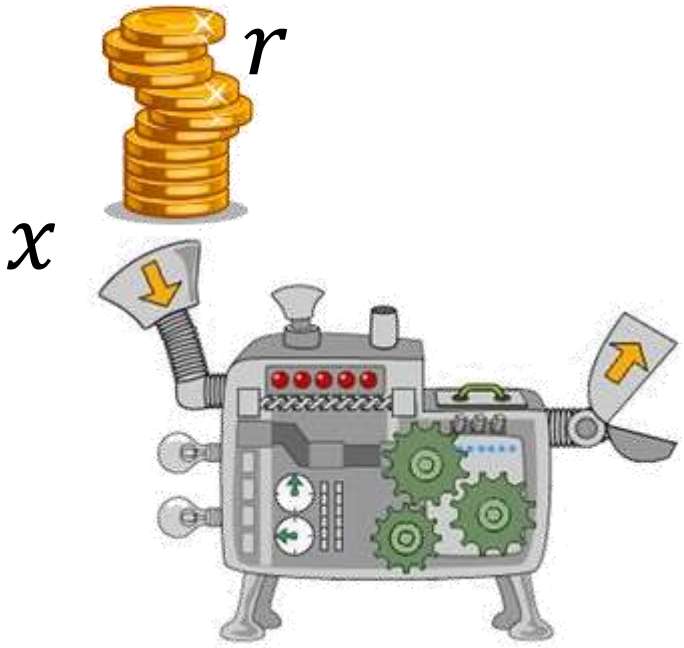
- $\mathrm{Setup}(1^\lambda)$: Outputs $(\mathrm{msk}, \mathrm{mpk})$
- $\mathrm{KeyGen}(\mathrm{msk}, f)$: Outputs decryption key sk_f
- $\mathrm{Encrypt}(\mathrm{mpk}, m)$: Outputs ciphertext ct_m
- $\mathrm{Decrypt}(\mathrm{sk}_f, \mathrm{ct}_m)$: Outputs $f(m)$

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

- $\mathrm{Setup}(1^\lambda)$: Outputs $(\mathrm{msk}, \mathrm{mpk})$
- $\mathrm{KeyGen}(\mathrm{msk}, f)$: Outputs decryption key sk_f
- Encrypt(mpk, rtext ct_m
- Decrypt(sk_f,

Deterministic function f

Functional Encryption for Randomized Functionalities (rFE) [GJKS15]

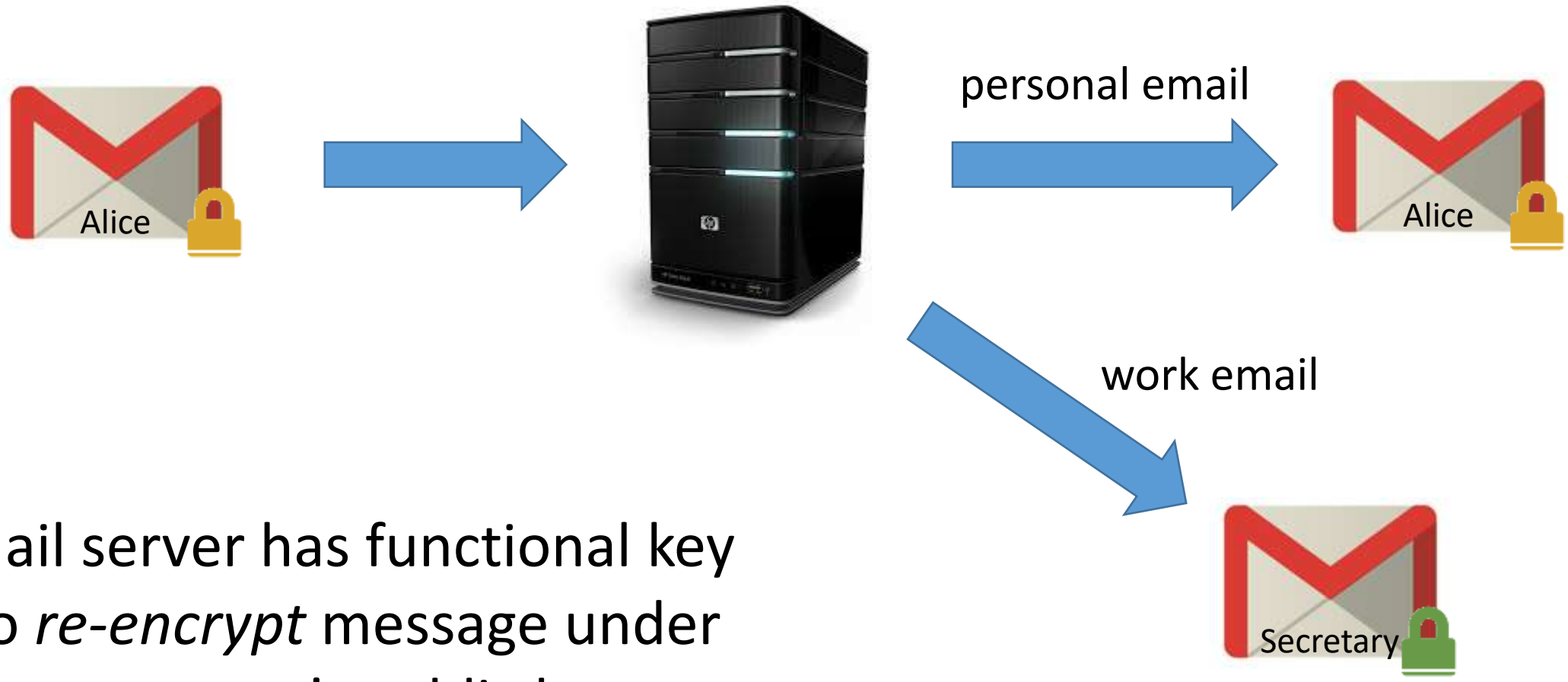


$f(x\,;r)$

But what if f is *randomized*?

Many interesting functions are *randomized*

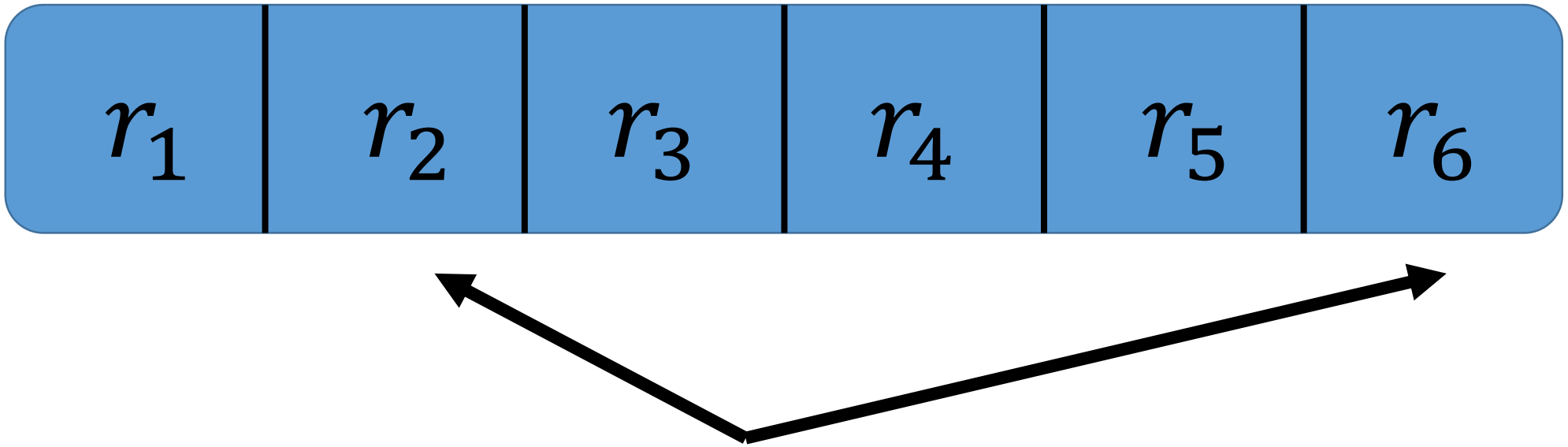
Application 1: Proxy Re-Encryption



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

Application 2: Auditing an Encrypted Database

Encrypted database of records



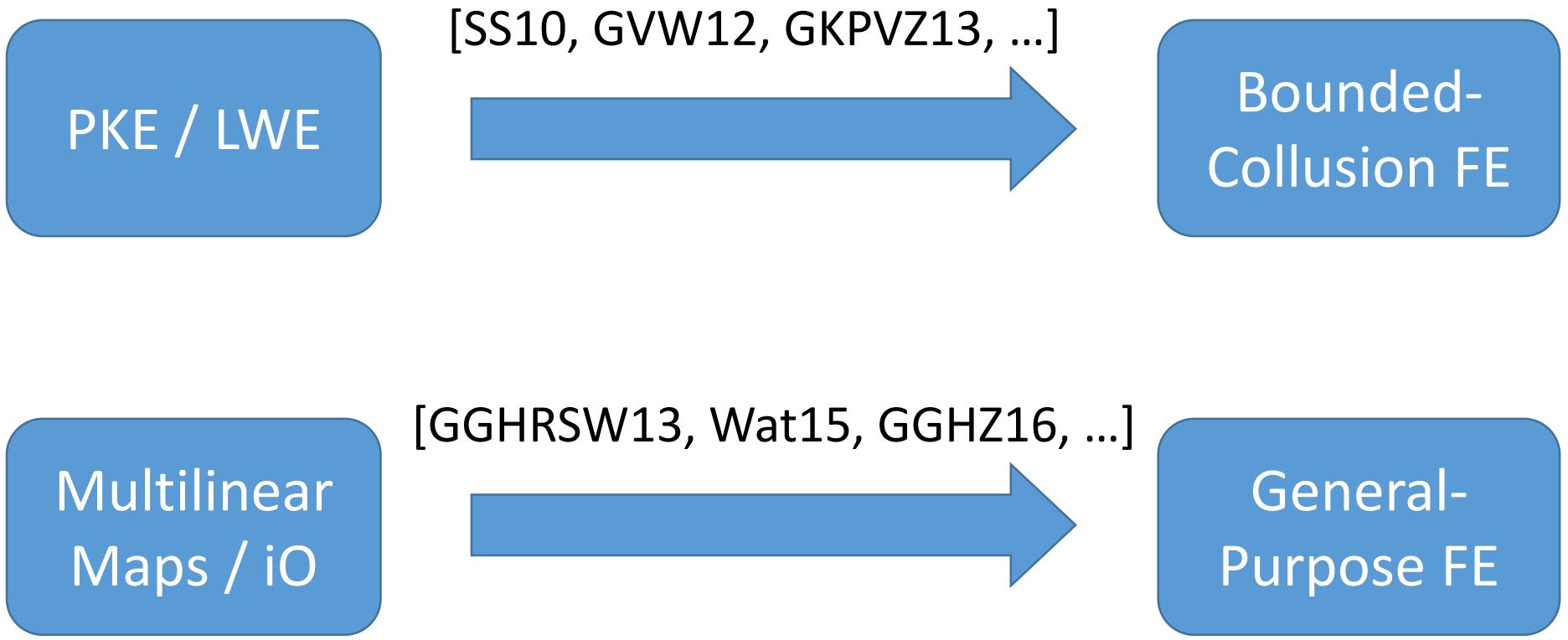
r_2 r_6

Sample a *random* subset to audit

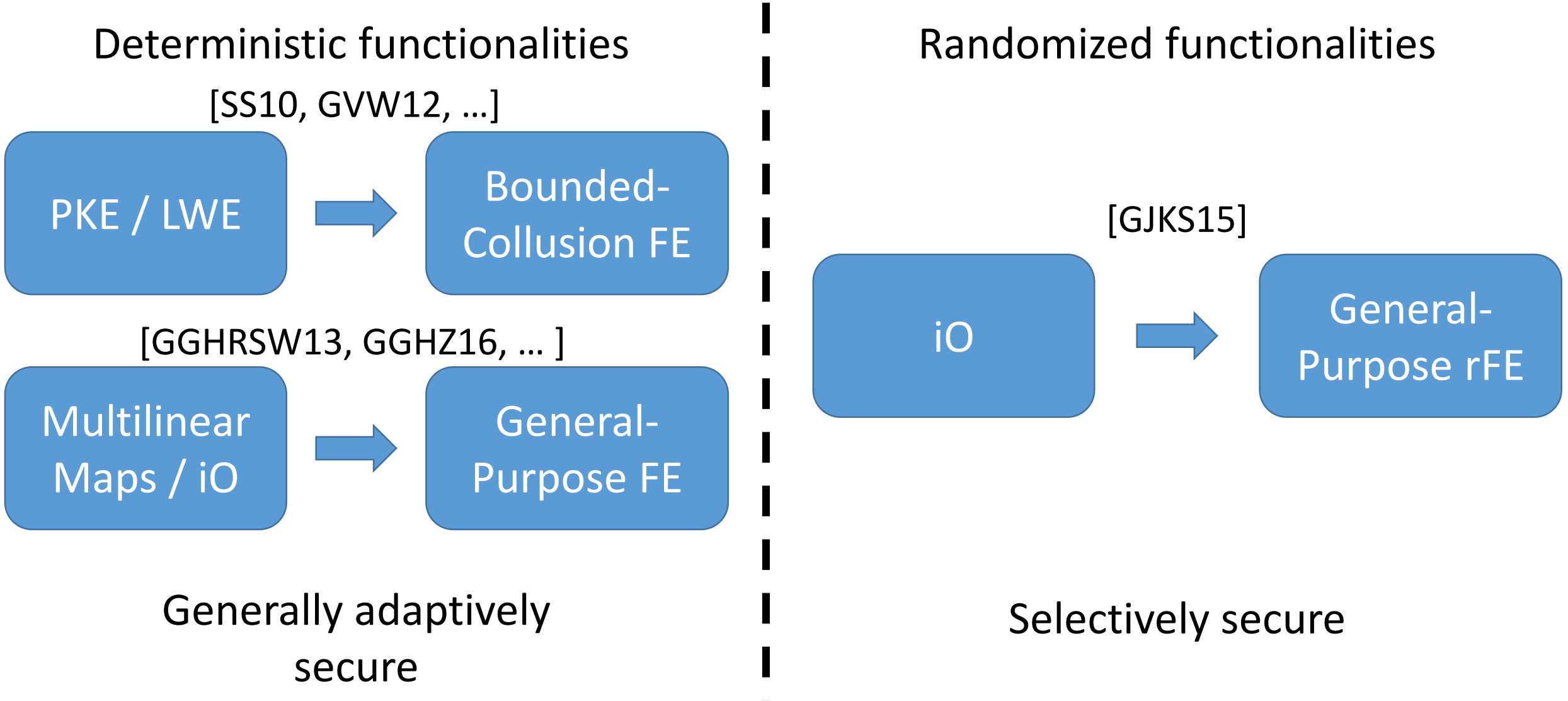
Does Public-Key rFE Exist?

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

Can be instantiated from a wide range of assumptions



The Landscape of (Public-Key) Functional Encryption



The Landscape of (Public-Key) Functional Encryption

Deterministic functionalities

Randomized functionalities

PKE / LW

Multiline
Maps / i

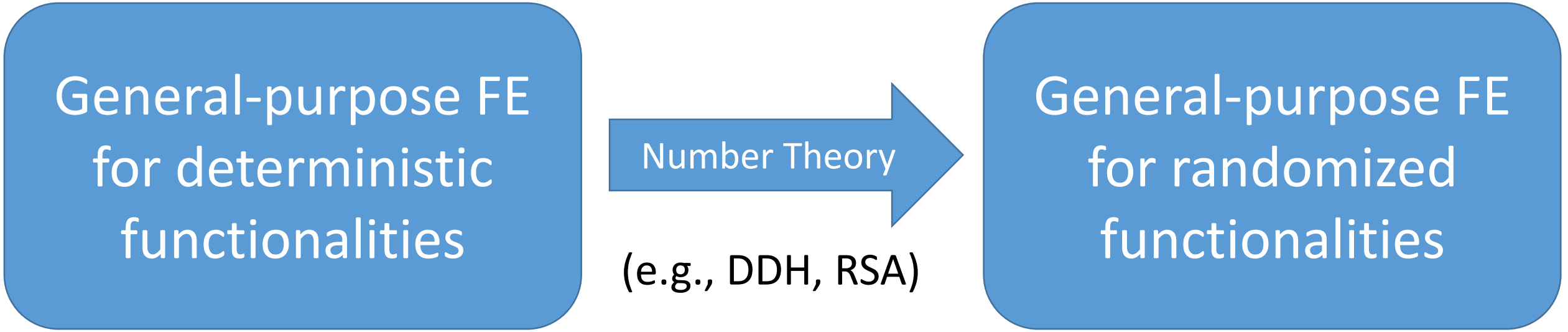
General-
urpose rFE

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

Generally adaptively secure

Selectively secure

Our Main Result



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

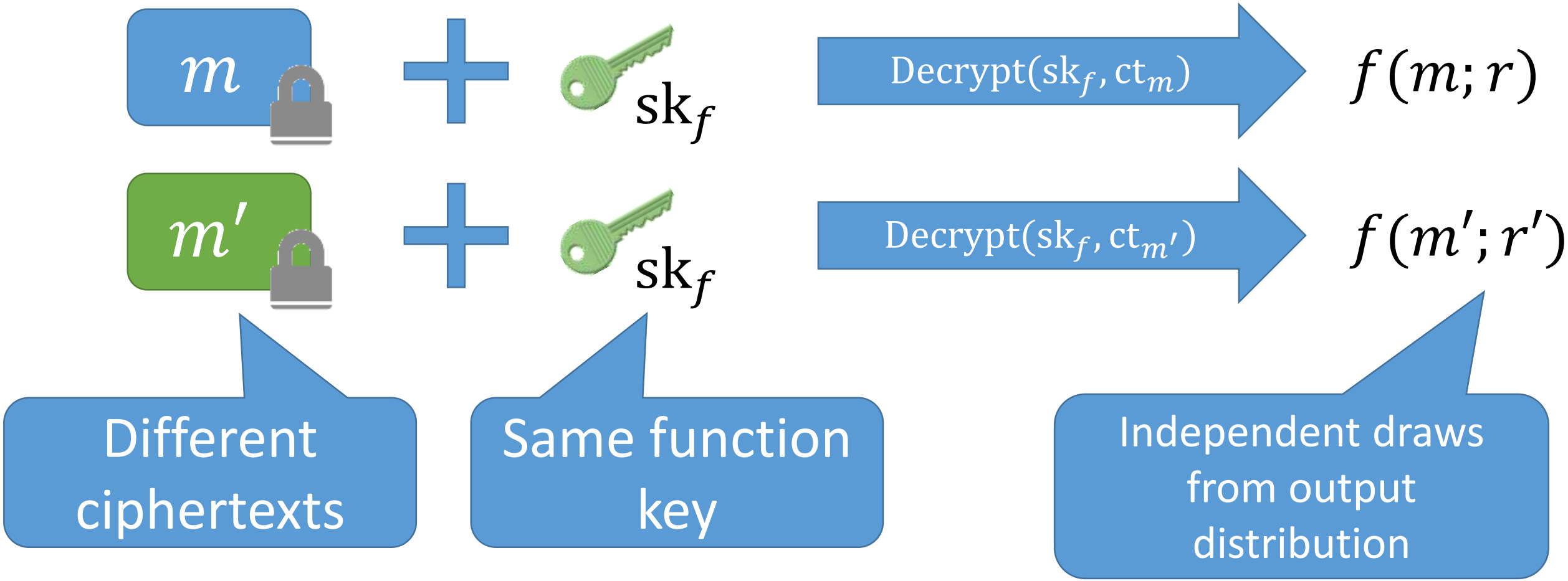
Defining rFE

Correctness for FE

Deterministic functions

Correctness for rFE [GJKS15]

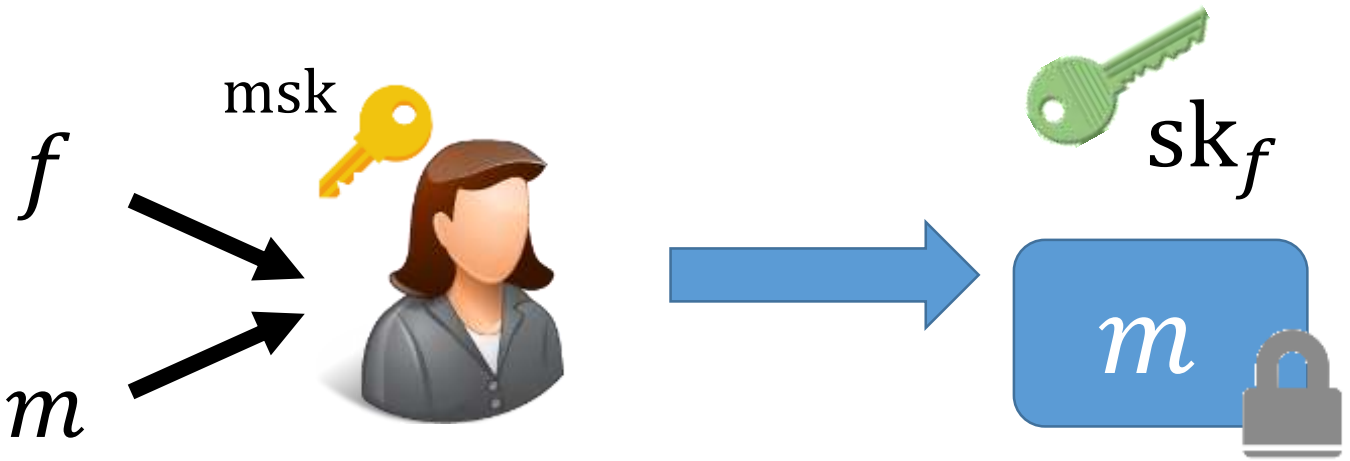
Randomized functions



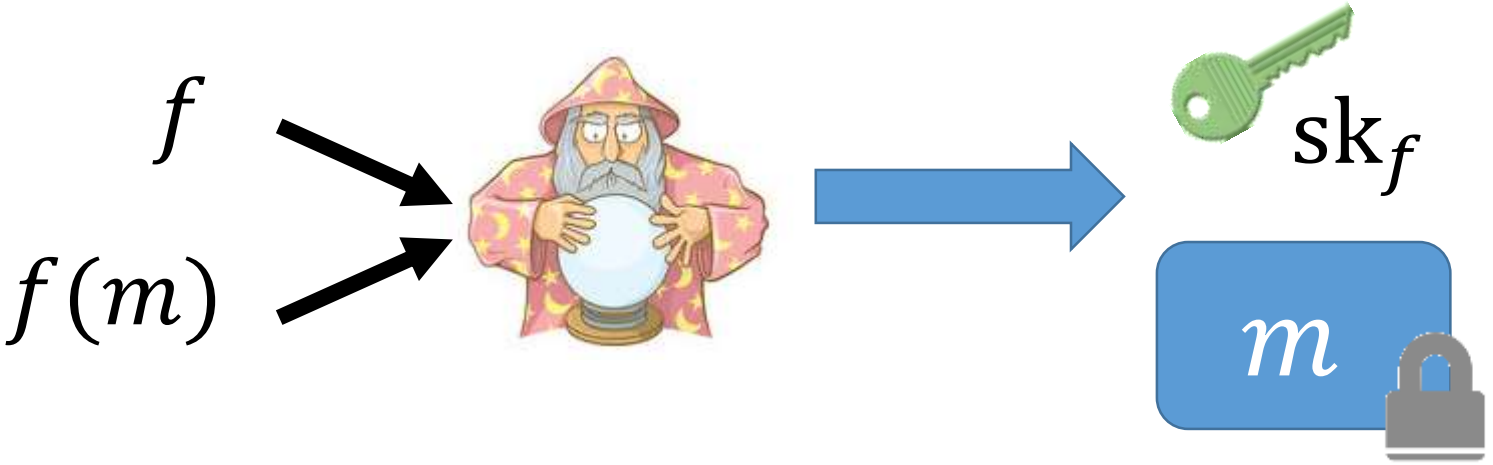
Correctness for rFE [GJKS15]

Randomized functionalities

Simulation-Based Security (Informally)

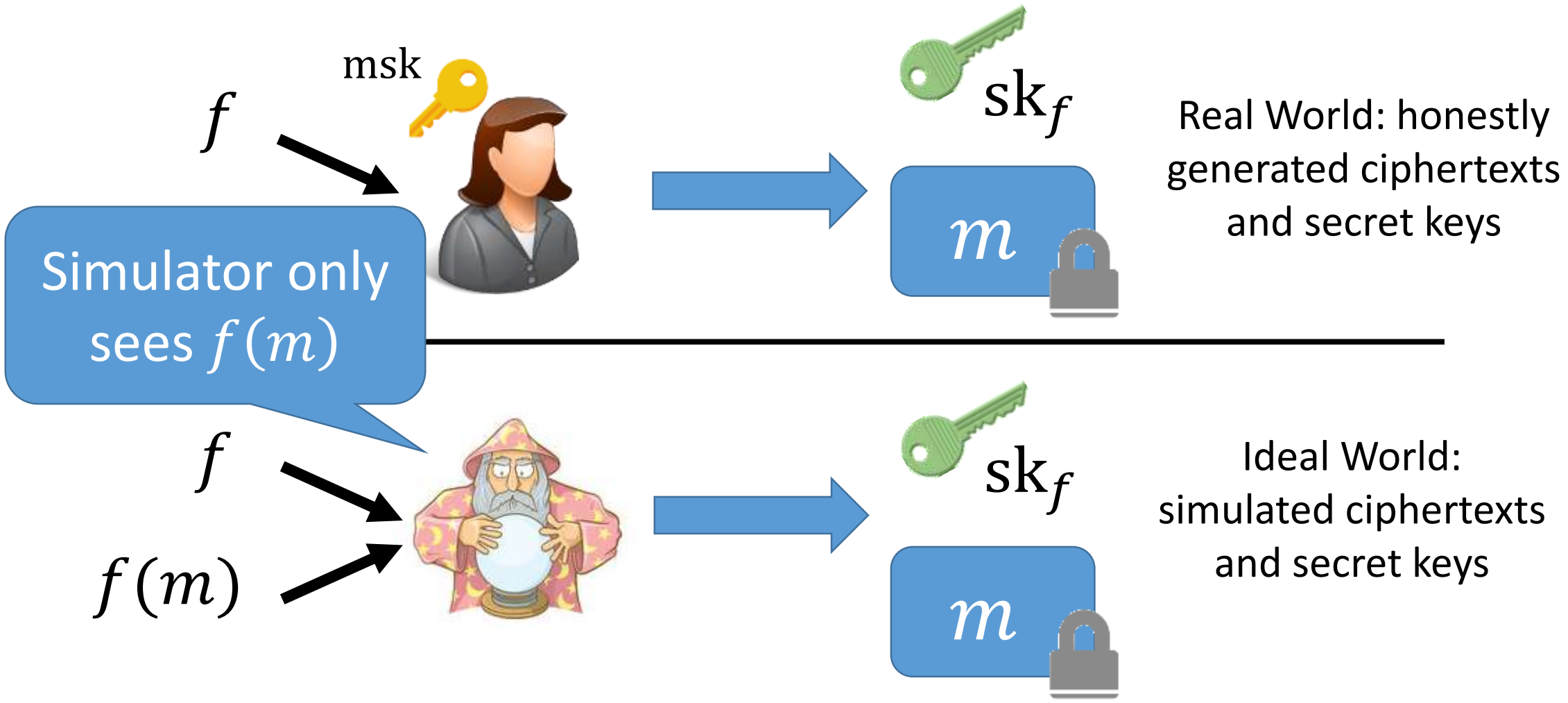


Real World: honestly generated ciphertexts and secret keys



Ideal World: simulated ciphertexts and secret keys

Simulation-Based Security (Informally)



The Case for Malicious Encrypters [GJKS15]

Encrypted database of records

r_3 r_4 r_5 r_6

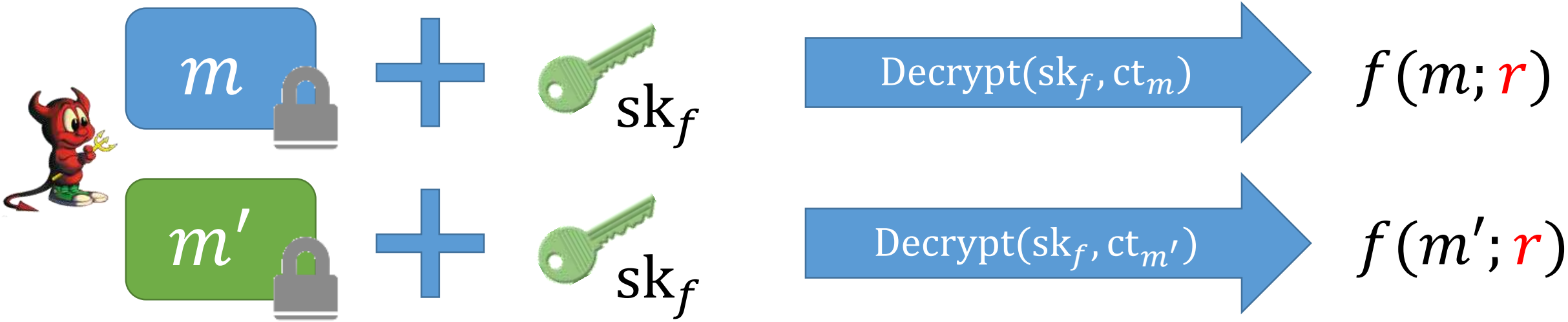
What if encrypter (bank) is adversarial?

r_2 6

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

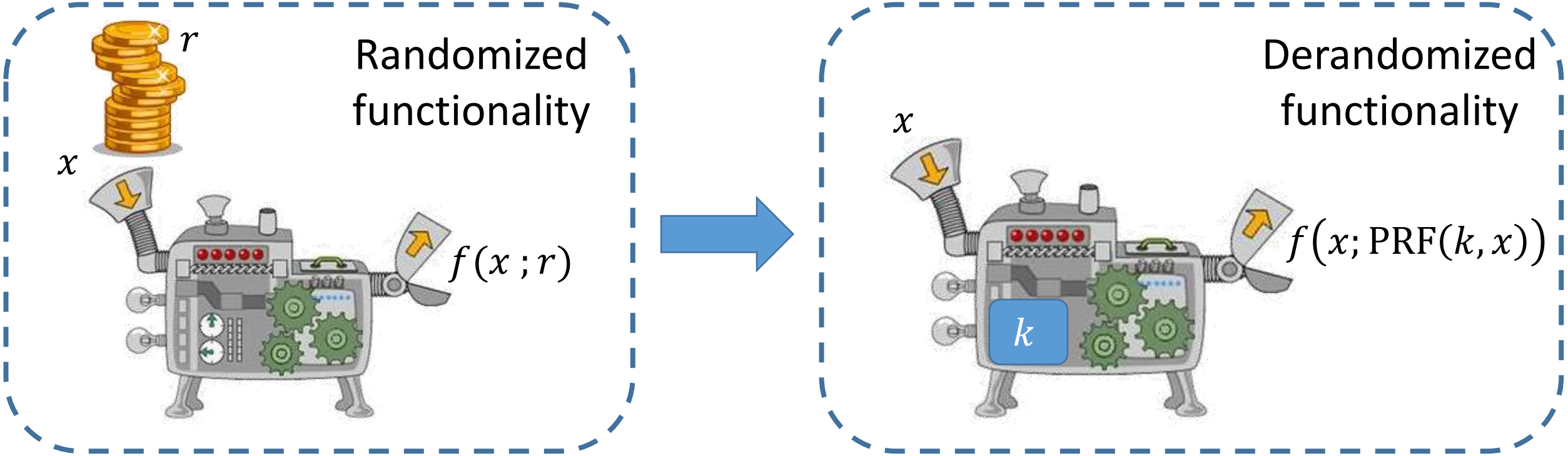
m + sk_f → $\mathrm{Decrypt}(\mathrm{sk}_f, \mathrm{ct}_m)$ → $f(m; r)$

m' + sk_f → $\mathrm{Decrypt}(\mathrm{sk}_f, \mathrm{ct}_{m'})$ → $f(m'; r)$

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

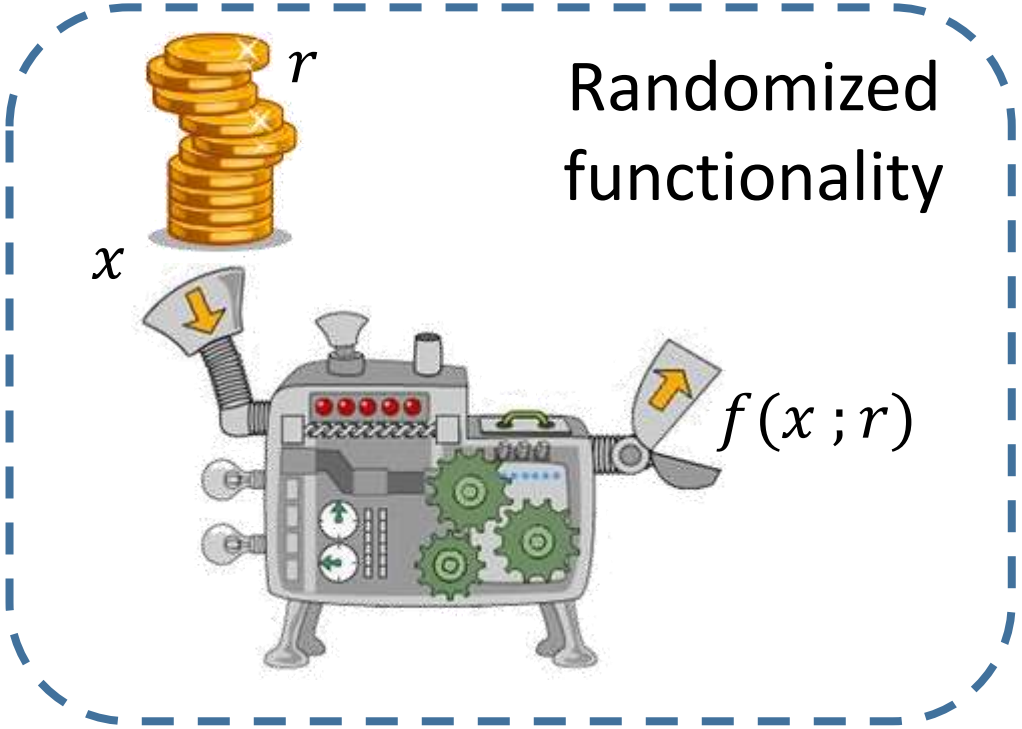
Our Generic Transformation

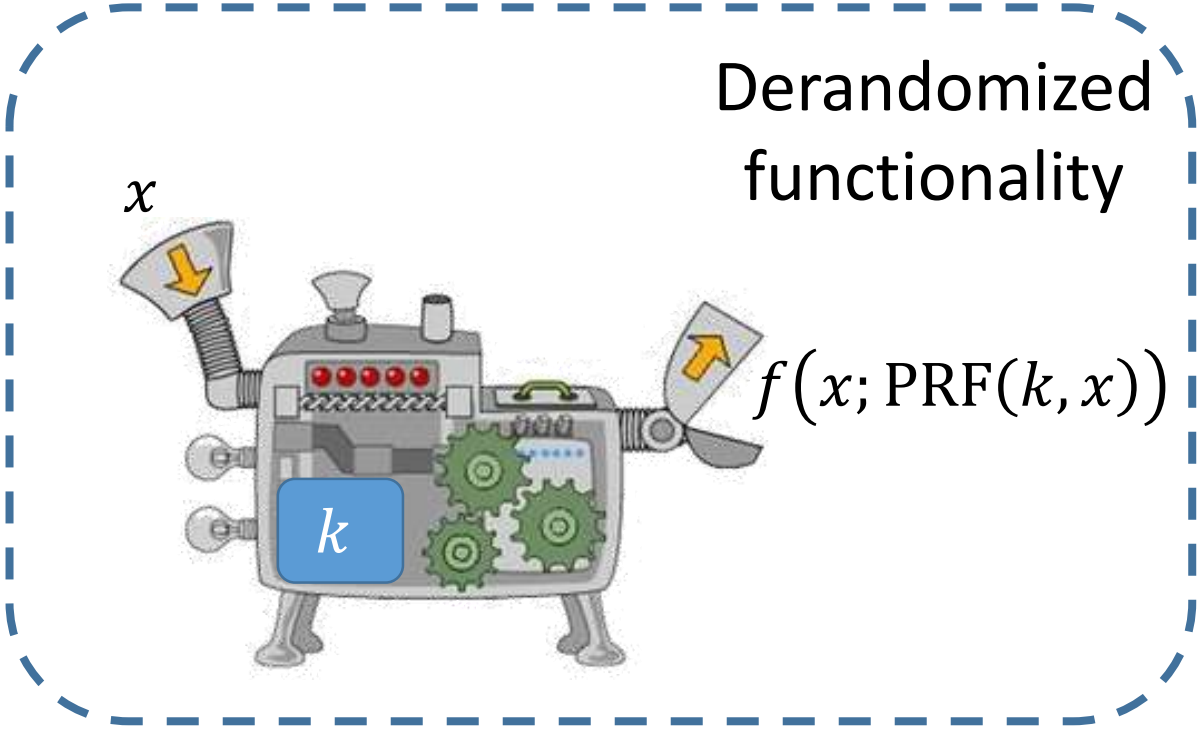
Starting Point: Derandomization



<u>Starting point:</u> construct "derandomized function" where randomness for f derived from outputs of a PRF

Starting Point: Derandomization



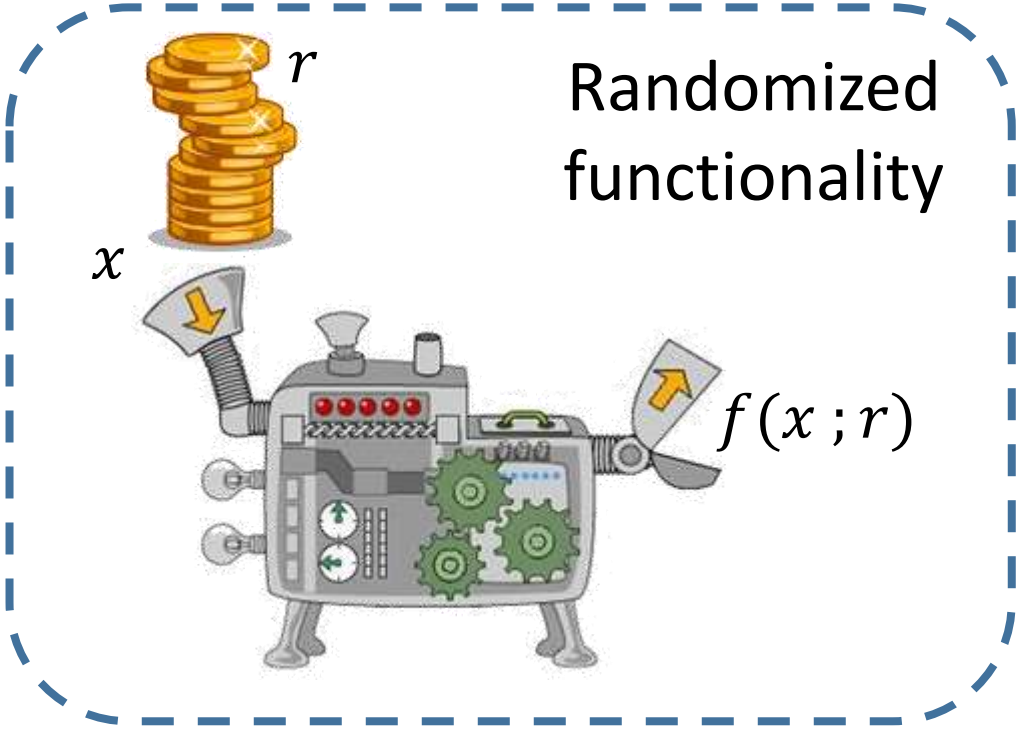


Randomized function f

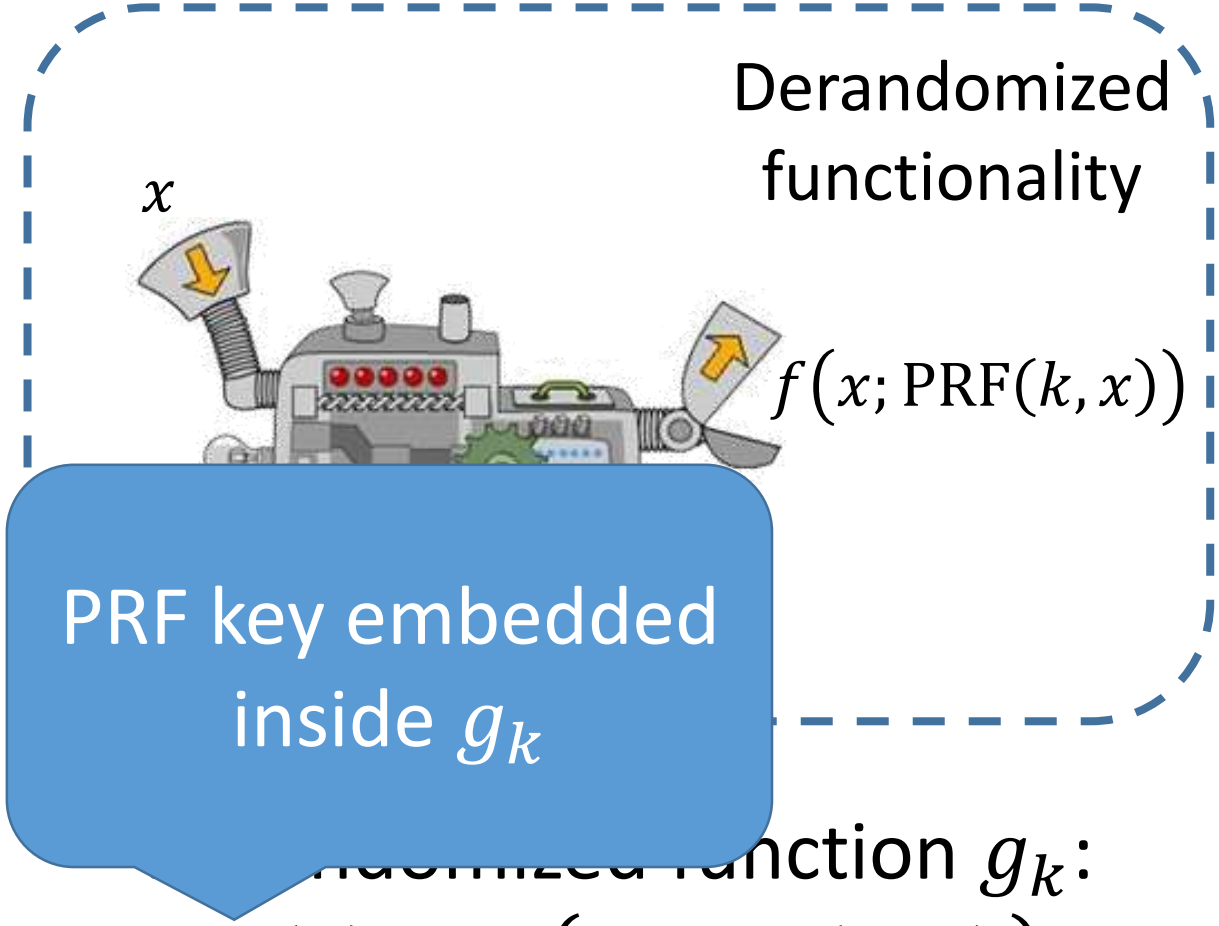
Derandomized function g_k:

$$g_k(x) = f\big(x, \mathrm{PRF}(k,x)\big)$$

Starting Point: Derandomization



Randomized function f



nction g_k:

$$g_k(x) = f\big(x, \mathrm{PRF}(k,x)\big)$$

Starting Point: Derandomization

rFE. KeyGen(msk, f)

$k \xleftarrow{R} \mathcal{K}$

FE. KeyGen(msk, g_k)

$g_k(x) = f\big(x, \mathrm{PRF}(k, x)\big)$

sk_{g_k}

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

Starting Point: Derandomization

rFE. KeyGen(msk, f)

$k \xleftarrow{R} \mathcal{K}$

FE. KeyGen(msk, g_k)

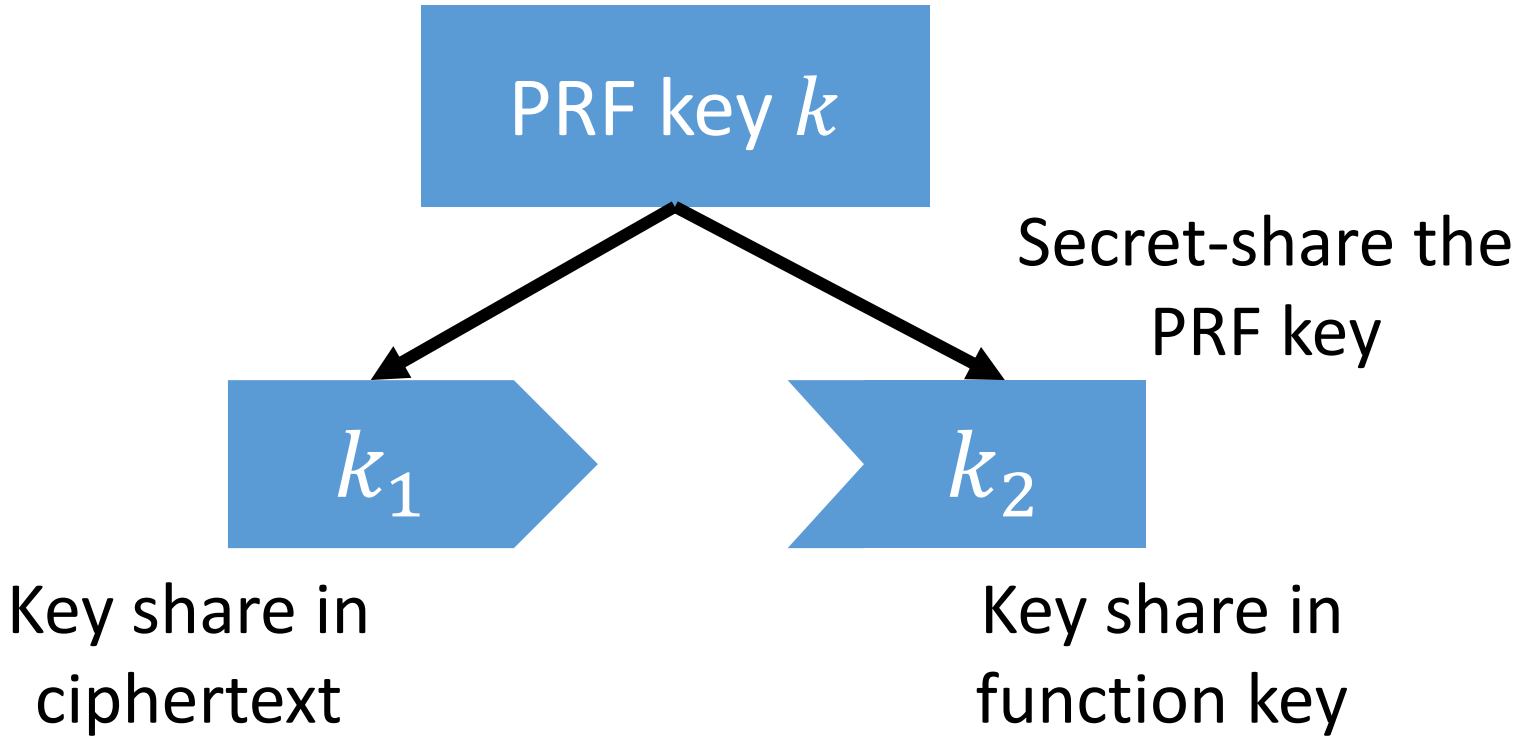
$g_k(x) = f\big(x, \mathrm{PRF}(k, x)\big)$

Given sk_{g_k}, adversary can learn the PRF key k

sk_{g_k}

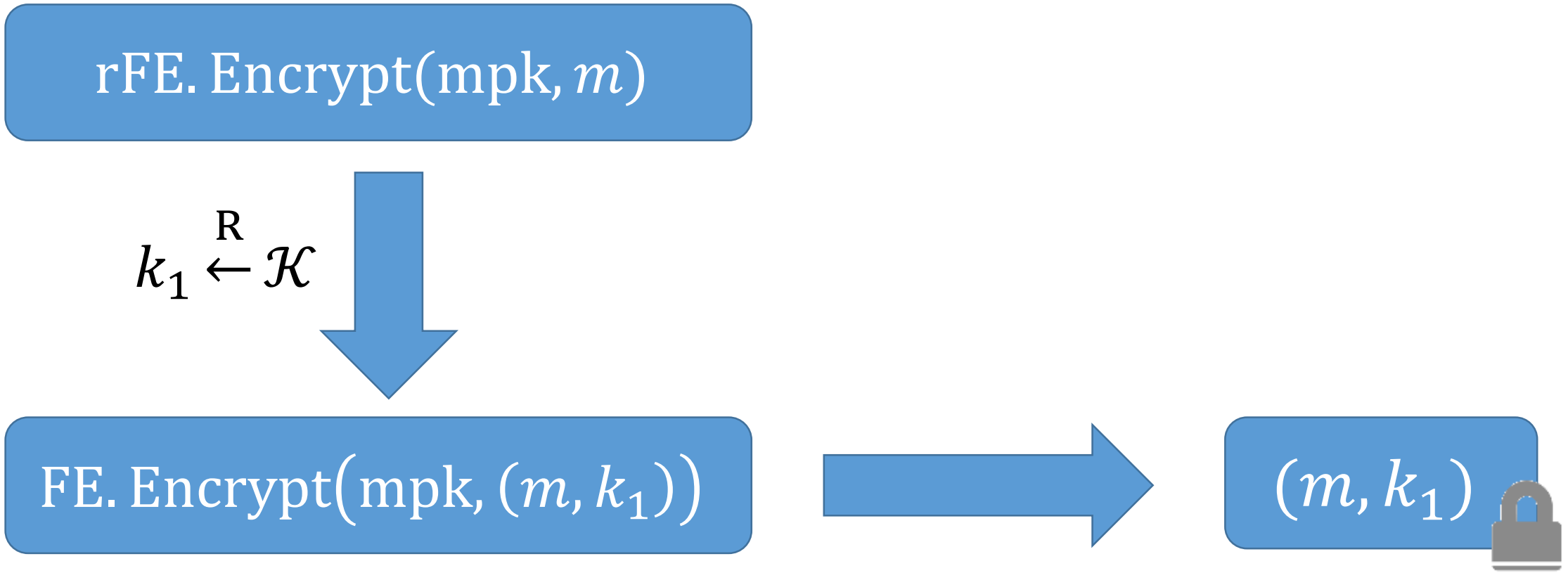
How to Hide the Key?

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



How to Hide the Key?

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

$\mathrm{rFE.KeyGen}(\mathrm{msk}, f)$

$k_2 \xleftarrow{R} \mathcal{K}$

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

Some operation to combine shares of key

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

$\mathrm{sk}_{g_{k_2}}$

How to Hide the Key?

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

$\text{rFE.KeyGen}(\text{msk}, f)$

$k_2 \xleftarrow{R} \mathcal{K}$

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

Encrypter controls k_1 so we need related-key security

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

$\text{sk}_{g_{k_2}}$

Security Against Dishonest Encrypters

Encrypter has a lot of flexibility in constructing ciphertexts:

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

$k_1 \xleftarrow{R} \mathcal{K}$

Encrypter can choose the key-share

Cannot influence output distribution due to RKA-security

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

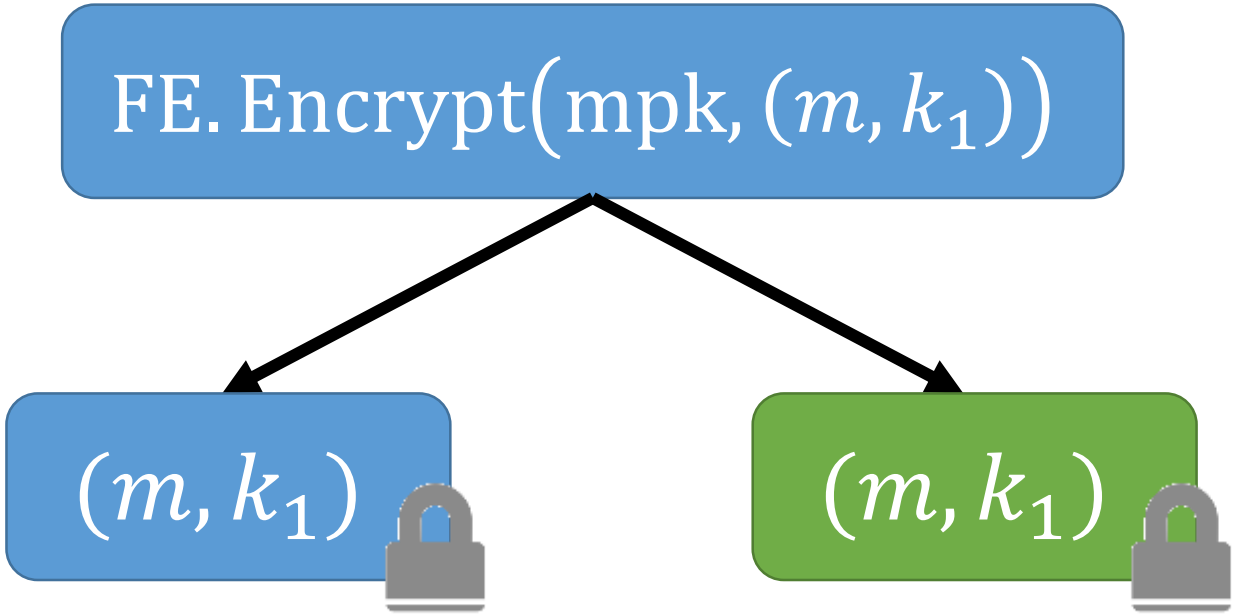
Encrypter can choose the randomness

(m, k_1)

Potentially problematic

Security Against Dishonest Encrypters

Encrypter has a lot of flexibility in constructing ciphertexts:

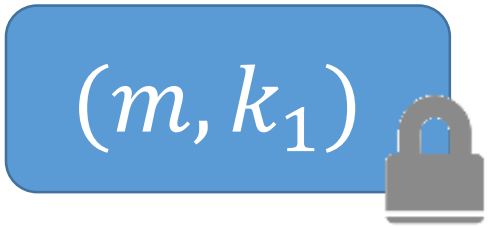


Run encryption algorithm twice with different randomness

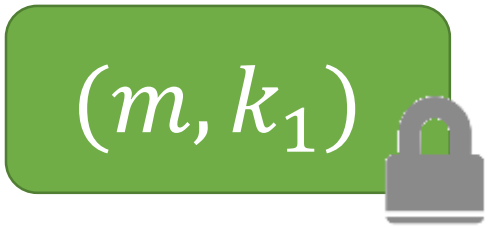
Two *distinct* FE ciphertexts encrypting the *same* message

Security Against Dishonest Encrypters

Encrypter has a lot of flexibility in constructing ciphertexts:



Reality: Decryption always produces *same* output

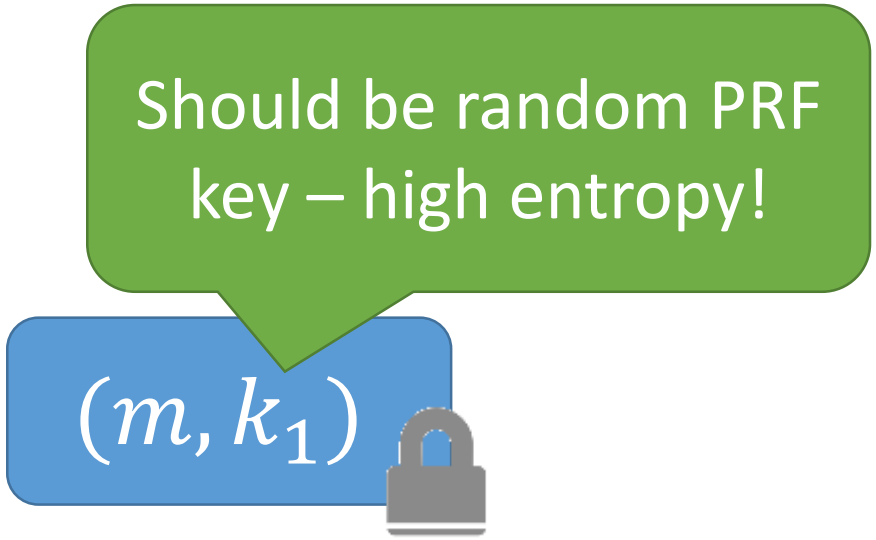


Desired: Two different ciphertexts, so should produces independent outputs

Encrypter has too much freedom in constructing ciphertexts

Applying Deterministic Encryption

Key observation: honestly generated ciphertexts have high entropy



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

Putting the Pieces Together

rFE. Encrypt(mpk, m)

$k_1 \xleftarrow{R} \mathcal{K}$

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

FE. Encrypt(mpk, (m, k_1) ; $h(k_1)$) + π

Randomness for FE encryption derived from deterministic function on k_1 (e.g., a PRG)

[See paper for full details.]

Putting the Pieces Together

rFE. Encrypt(mpk, m)

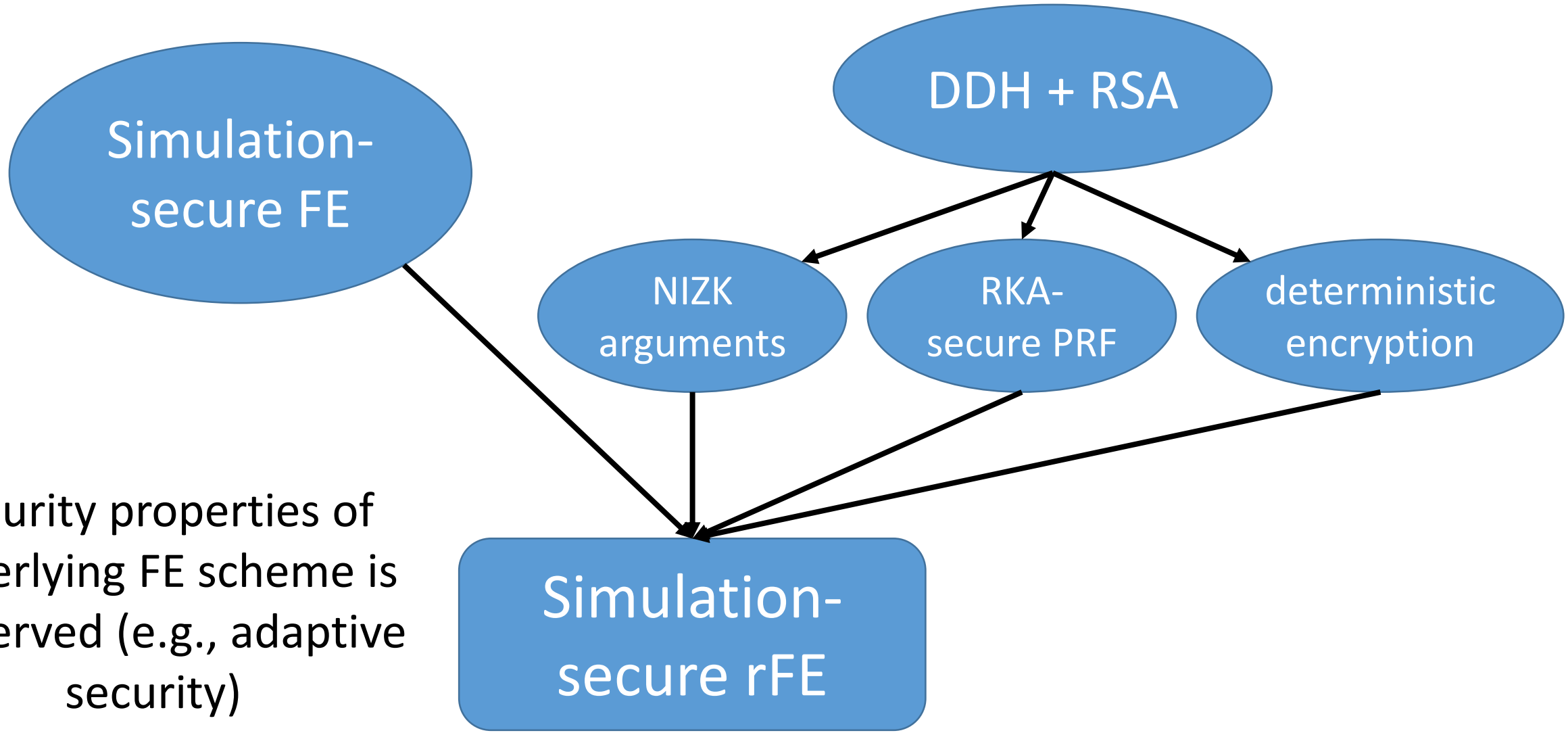
$k_1 \xleftarrow{R} \mathcal{K}$

FE. Encrypt(mpk, (m, k_1) ; $h(k_1)$) + π

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

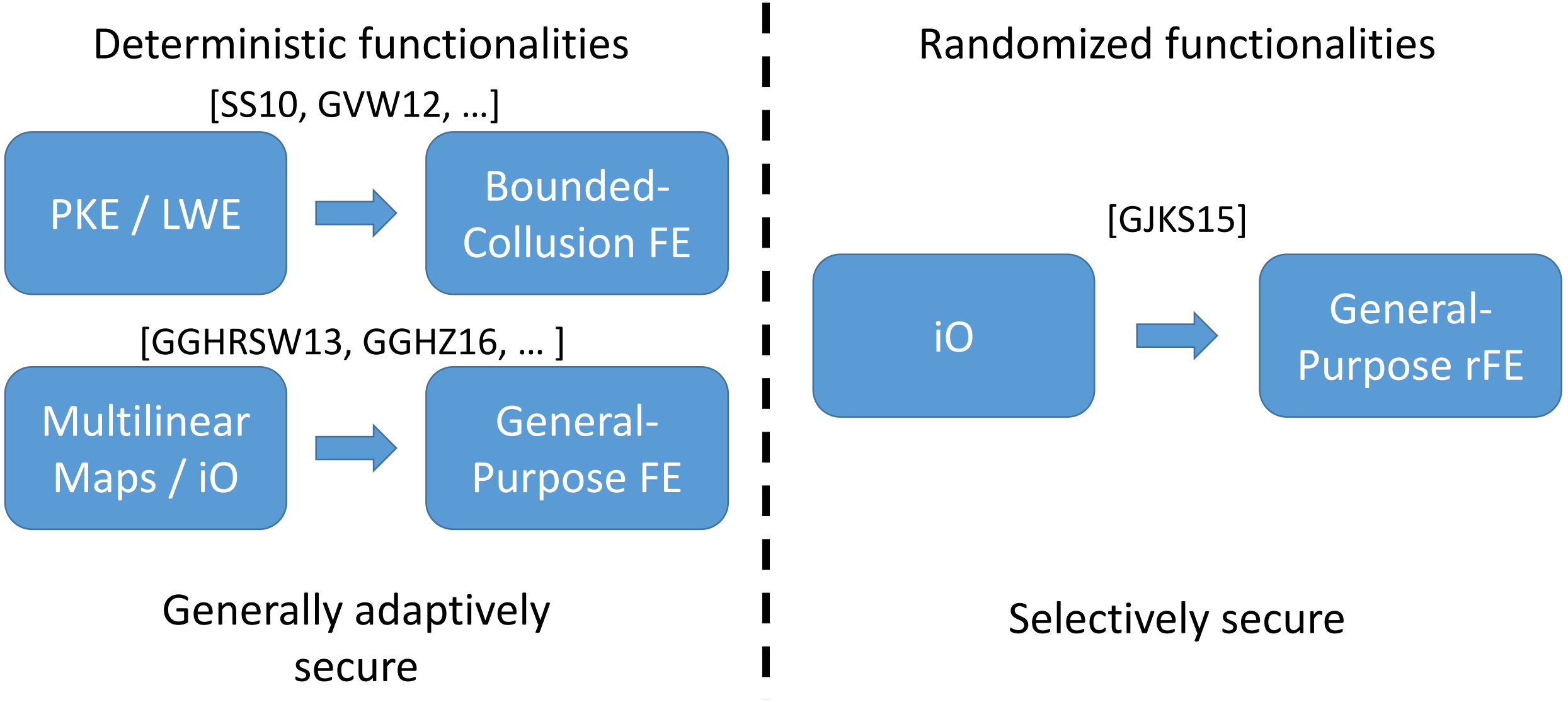
[See paper for full details.]

Our Transformation in a Nutshell

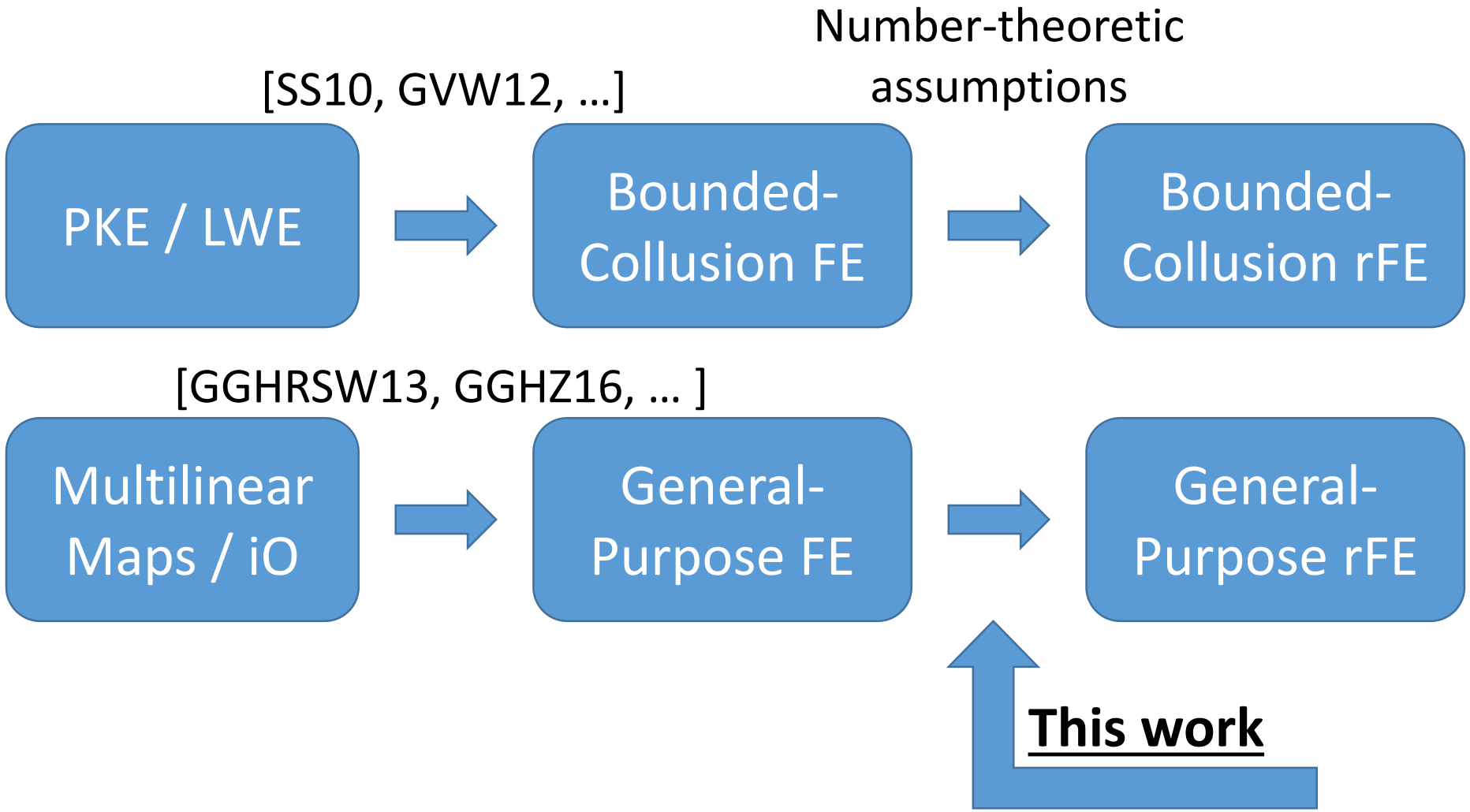


Security properties of underlying FE scheme is preserved (e.g., adaptive security)

The State of (Public-Key) Functional Encryption



The State of (Public-Key) Functional Encryption



Adaptively secure against malicious encrypters!

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`