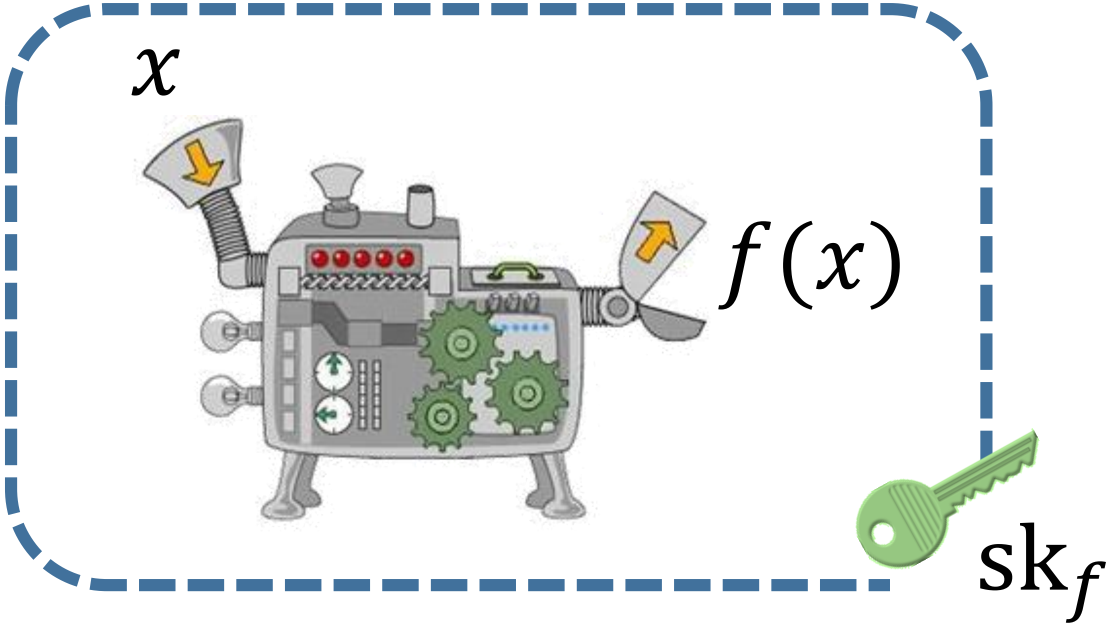
Functional Encryption: Deterministic to Randomized Functions from Simple Assumptions

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]

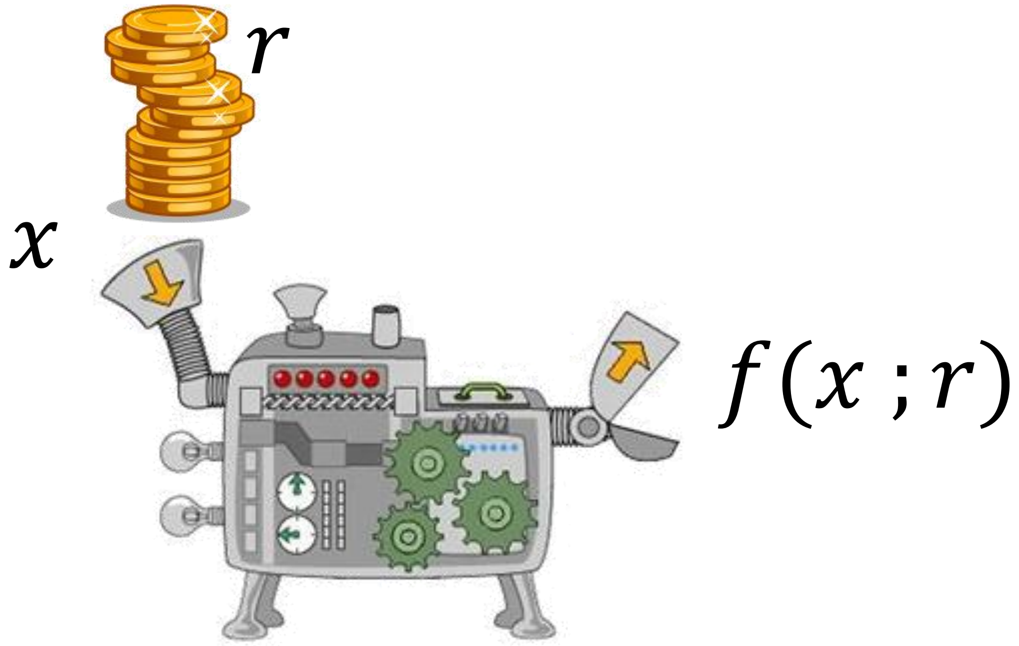
- $\mathrm{Setup}(1^\lambda)$: Outputs $(\mathrm{msk}, \mathrm{mpk})$
- $\mathrm{KeyGen}(\mathrm{msk}, f)$: Outputs decryption key sk_f
- $\mathrm{Encrypt}(\mathrm{mpk}, m)$: Outputs ciphertext ct_m
- $\mathrm{Decrypt}(\mathrm{sk}_f, \mathrm{ct}_m)$: Outputs $f(m)$

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

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

Deterministic function f

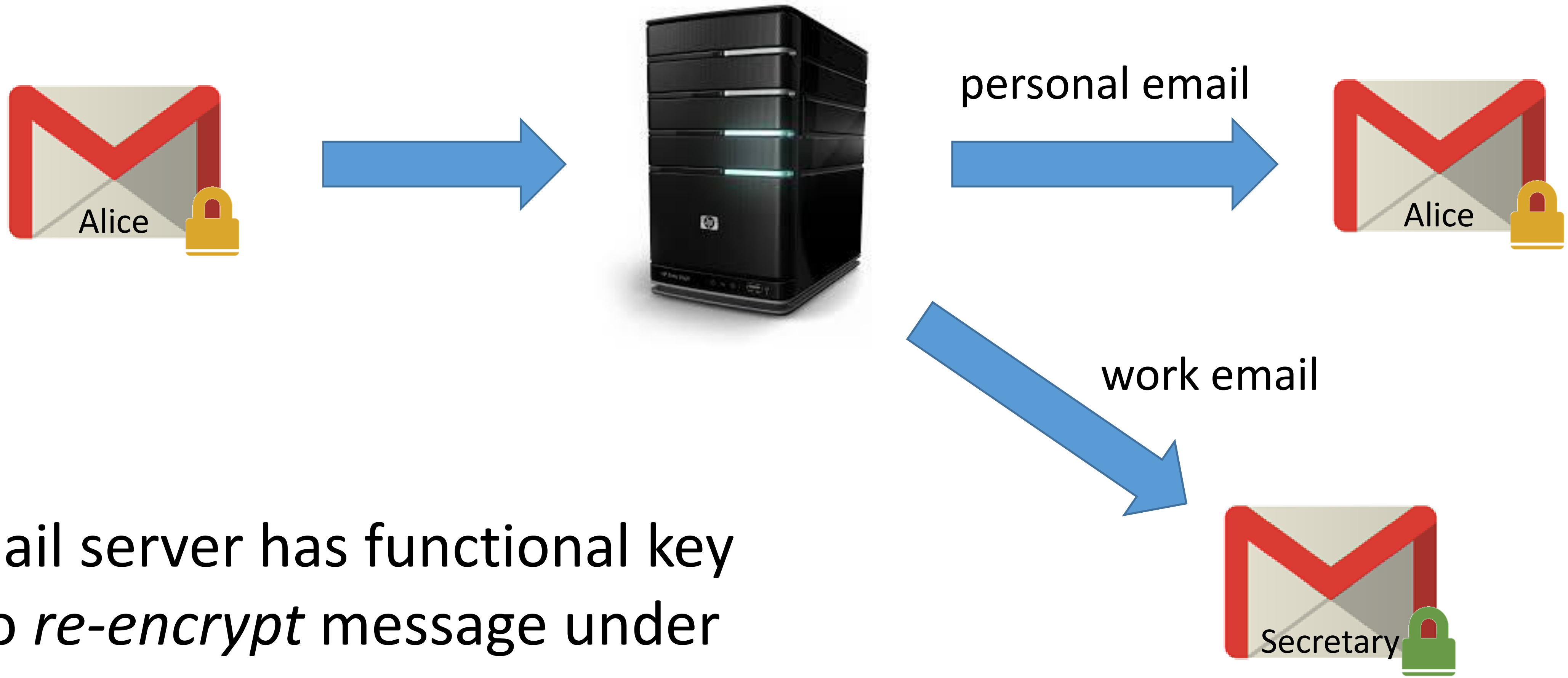
Functional Encryption for Randomized Functionalities (rFE) [GJKS15]



But what if f is *randomized*?

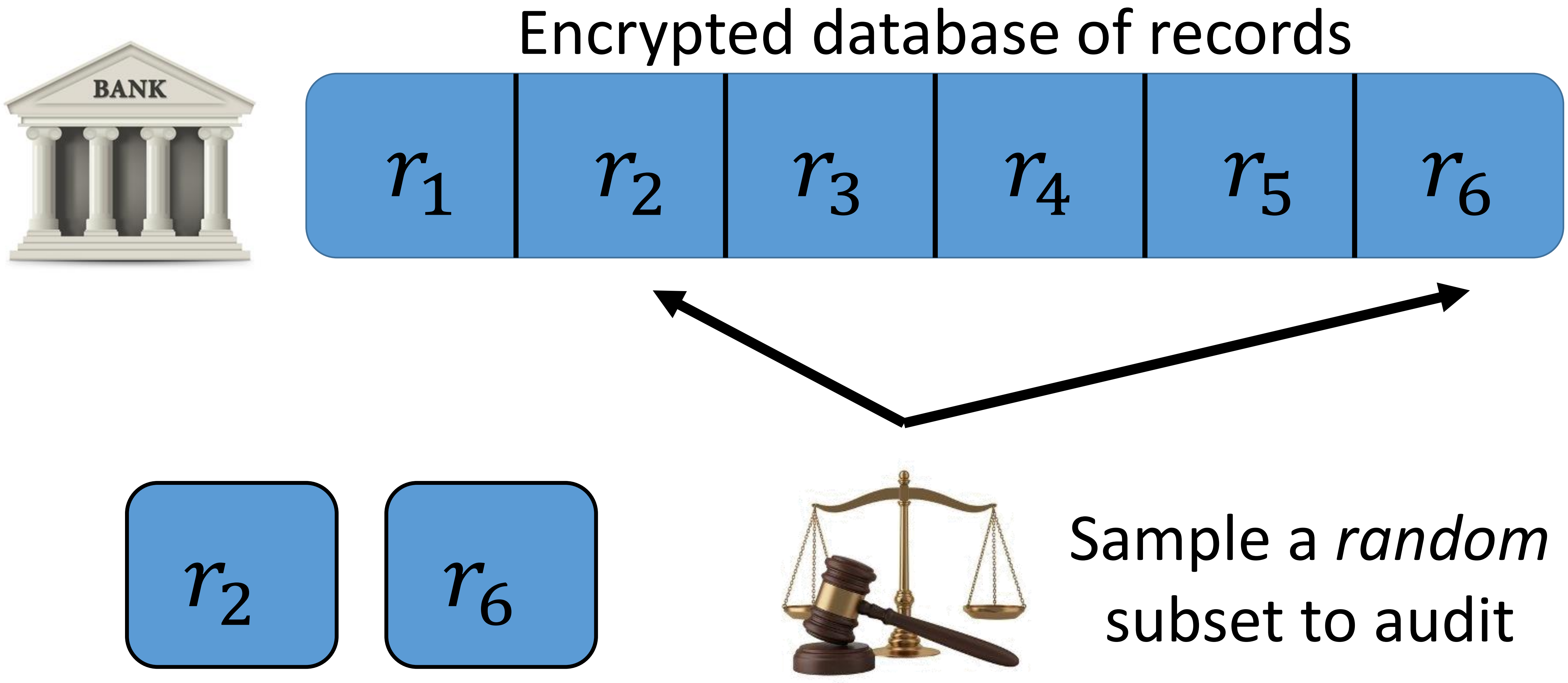
Many interesting functions are *randomized*

Application 1: Proxy Re-Encryption

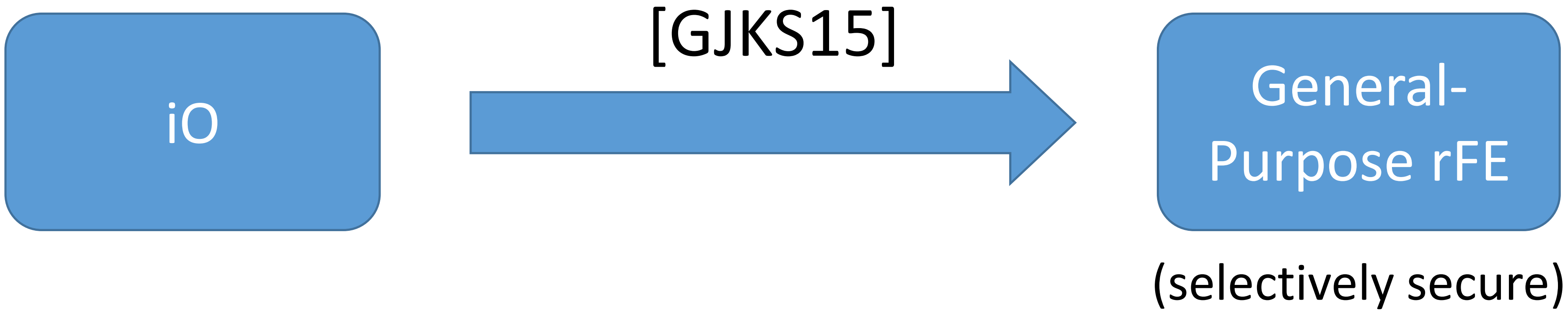


Mail server has functional key to *re-encrypt* message under secretary’s public key

Application 2: Auditing an Encrypted Database



Does Public-Key rFE Exist?

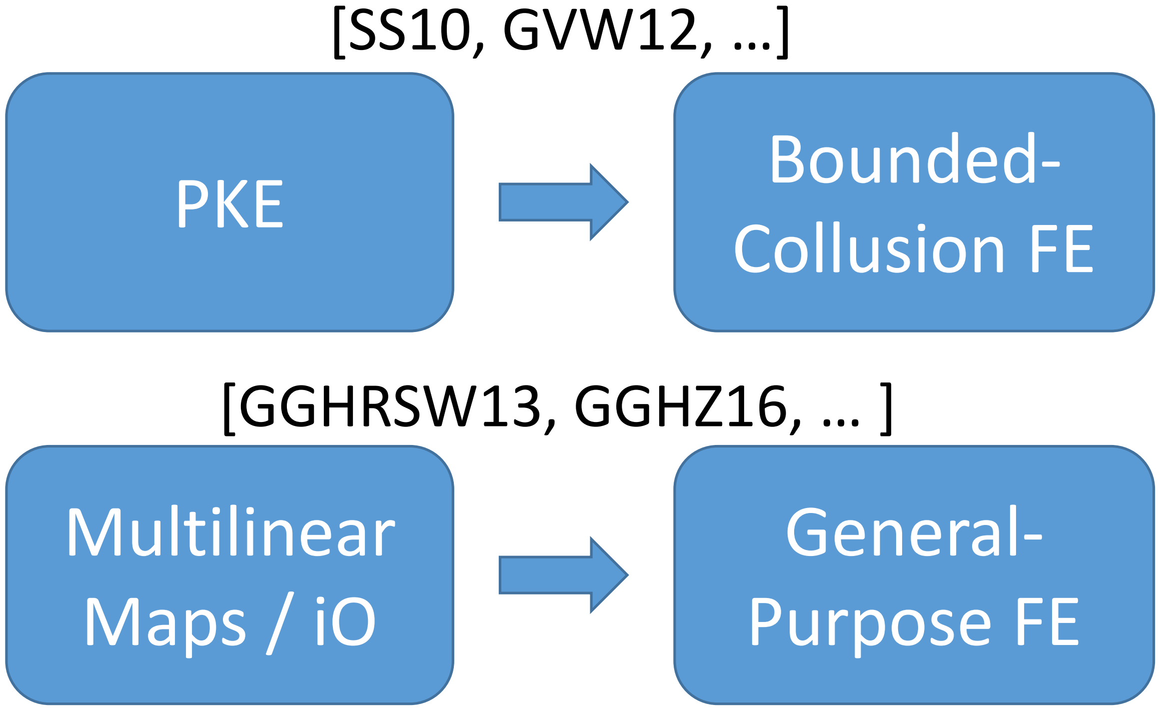


Public-Key Functional Encryption [BSW11, O’N10]

Can be instantiated from a wide range of assumptions

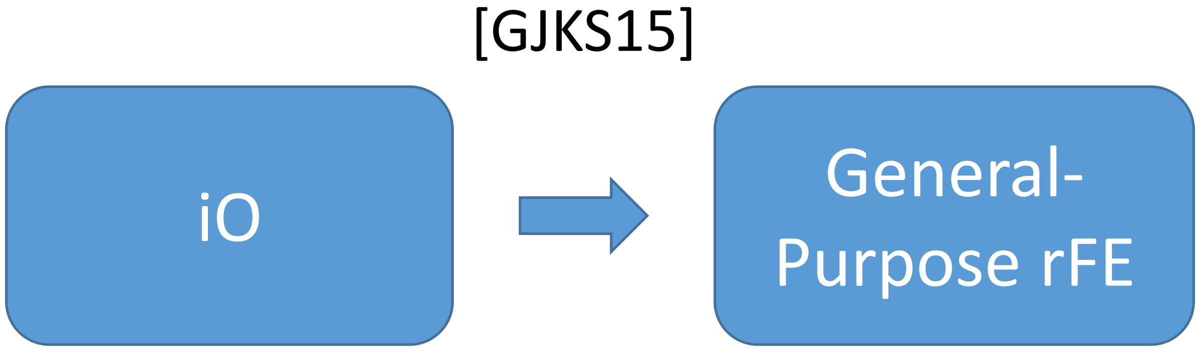
The State of (Public-Key) Functional Encryption

Deterministic functionalities



Generally adaptively secure

Randomized functionalities



Selectively secure

The State of (Public-Key) Functional Encryption

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

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.*

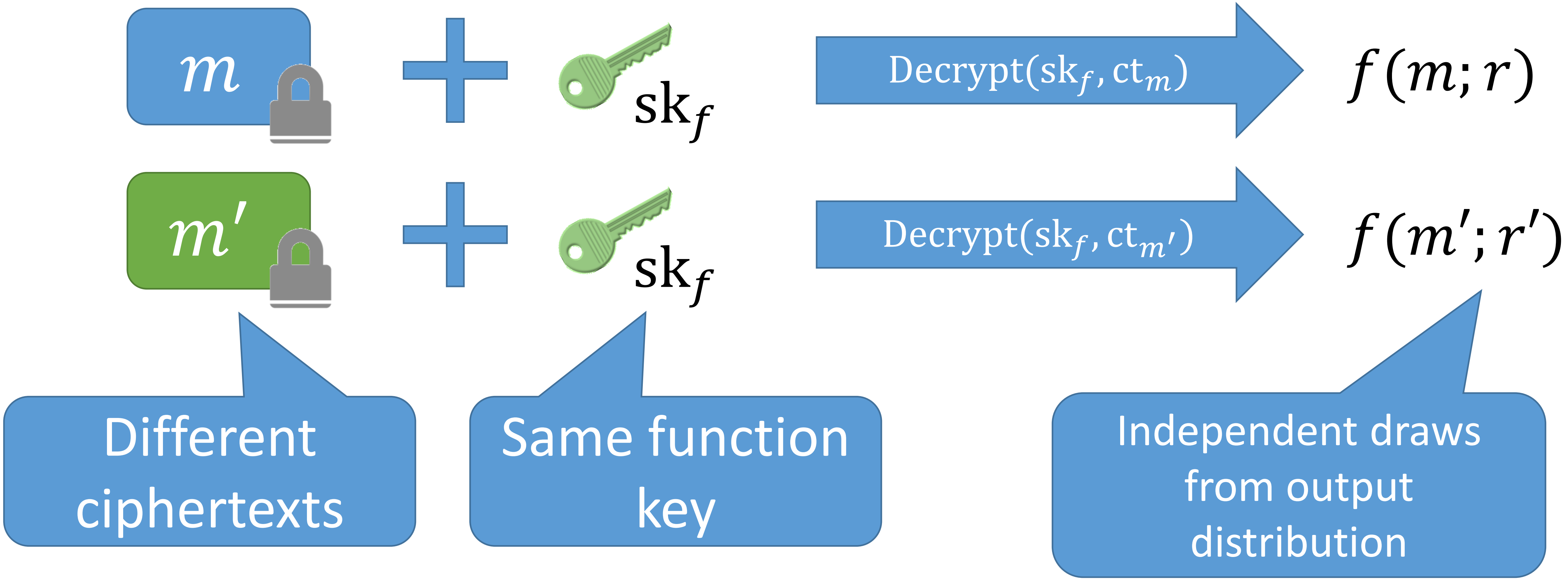
Defining rFE

Defining Correctness for FE

Deterministic functionalities

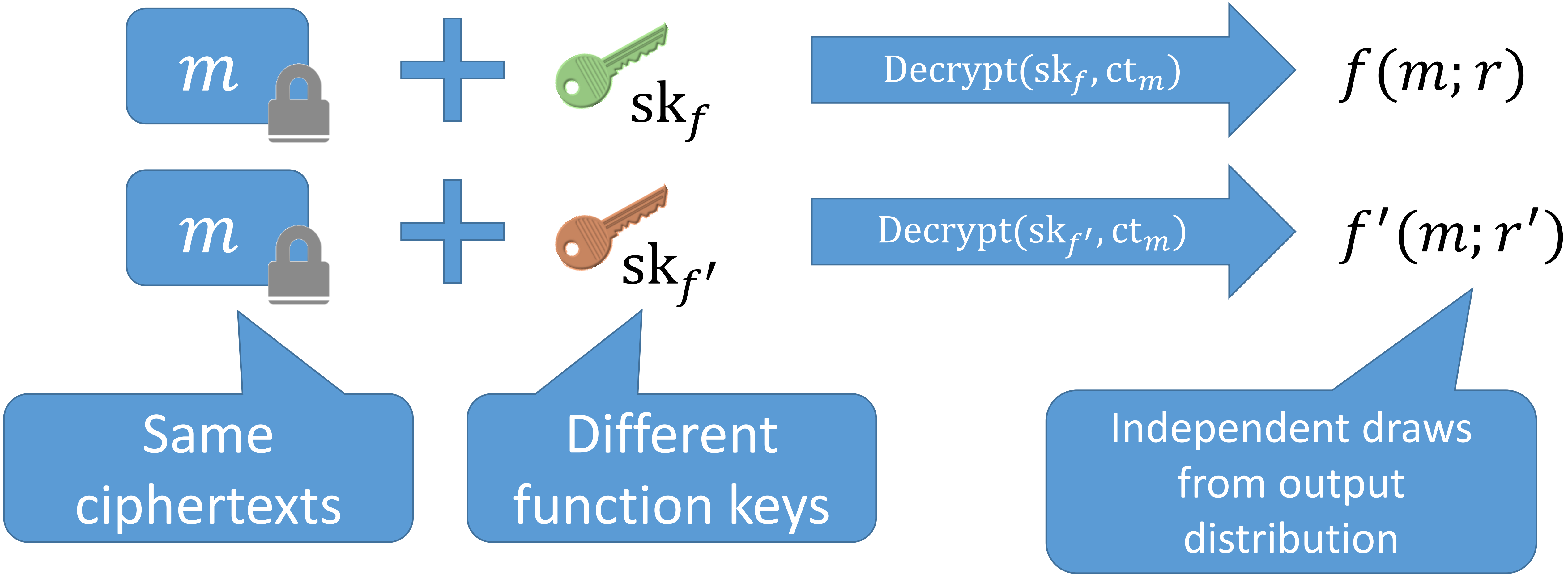
Defining Correctness for rFE [GJKS15]

Randomized functionalities

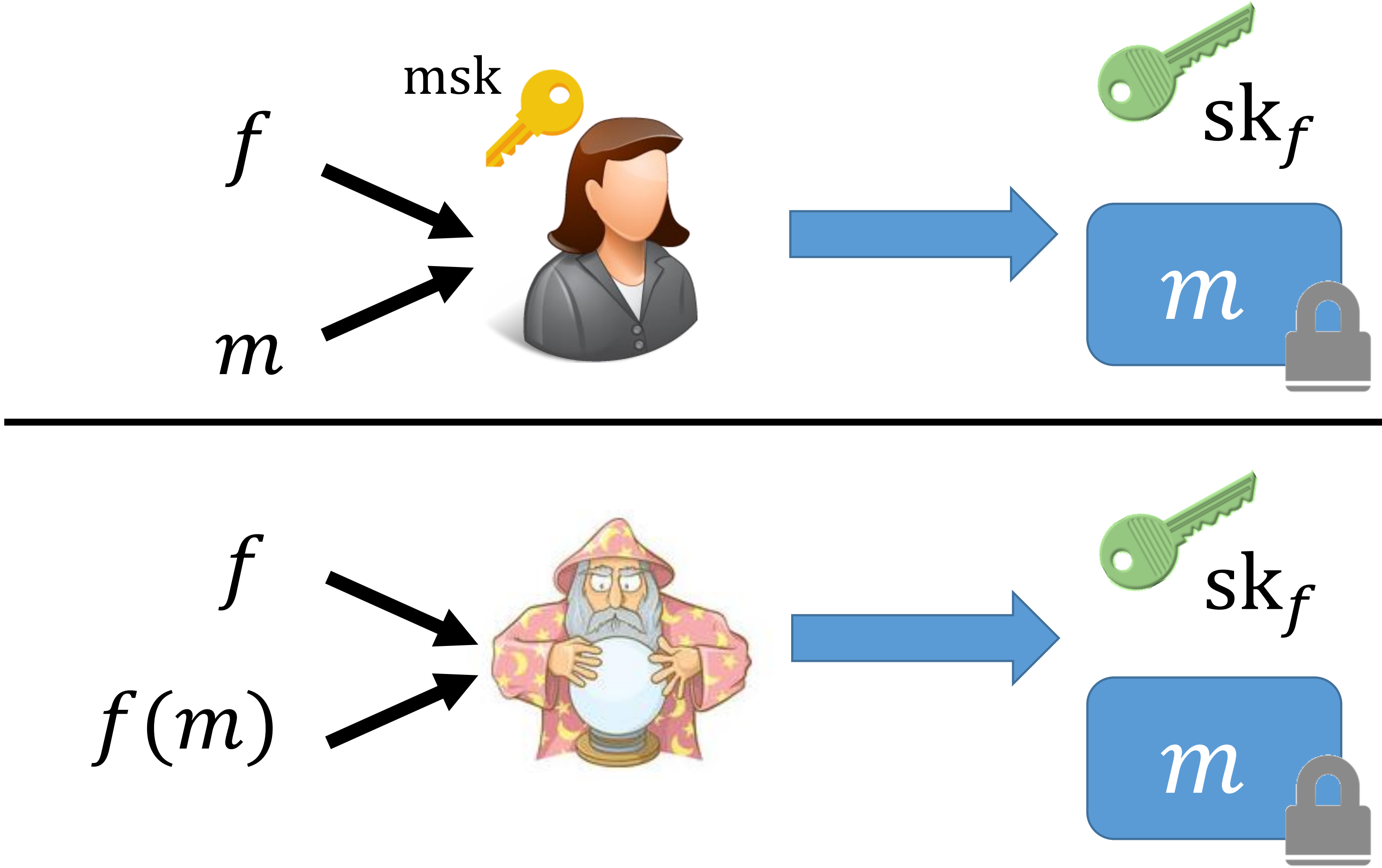


Defining Correctness for rFE [GJKS15]

Randomized functionalities



Simulation-Based Security (Informally)

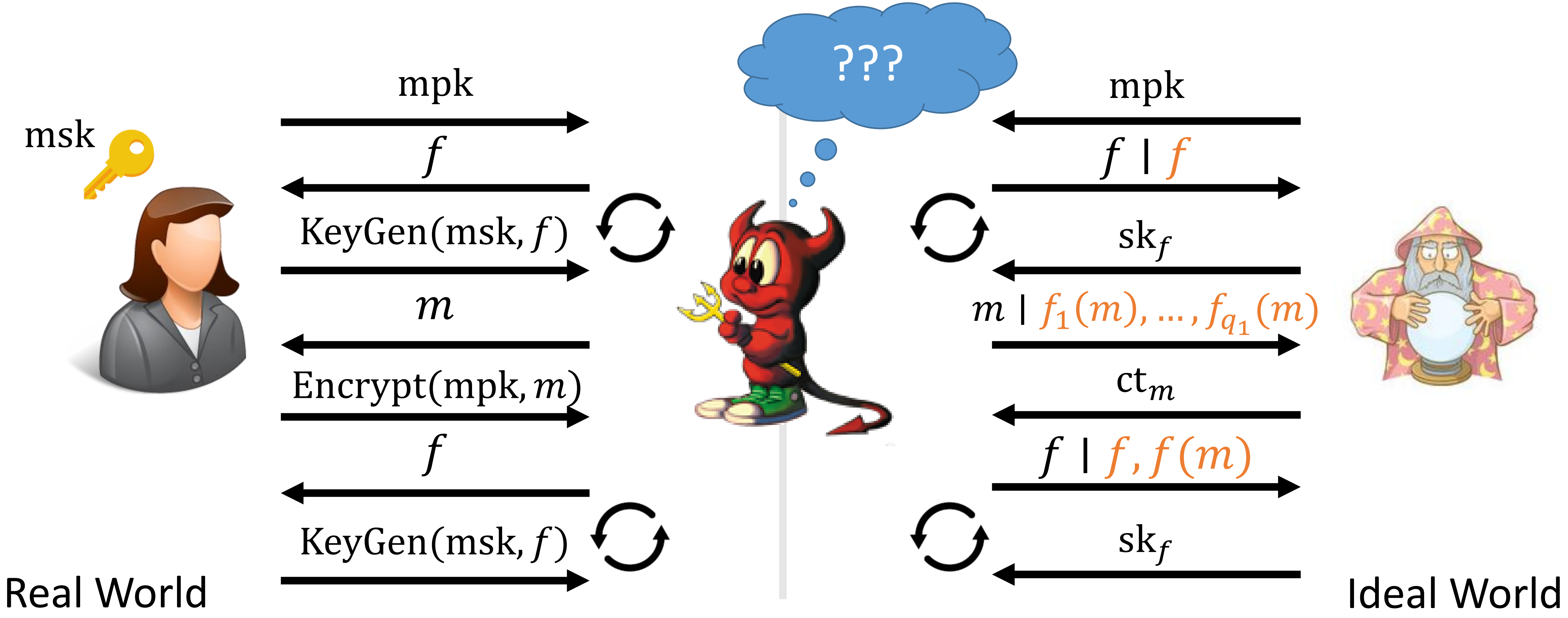


Real World: honestly generated ciphertexts and secret keys

Ideal World: simulated ciphertexts and secret keys

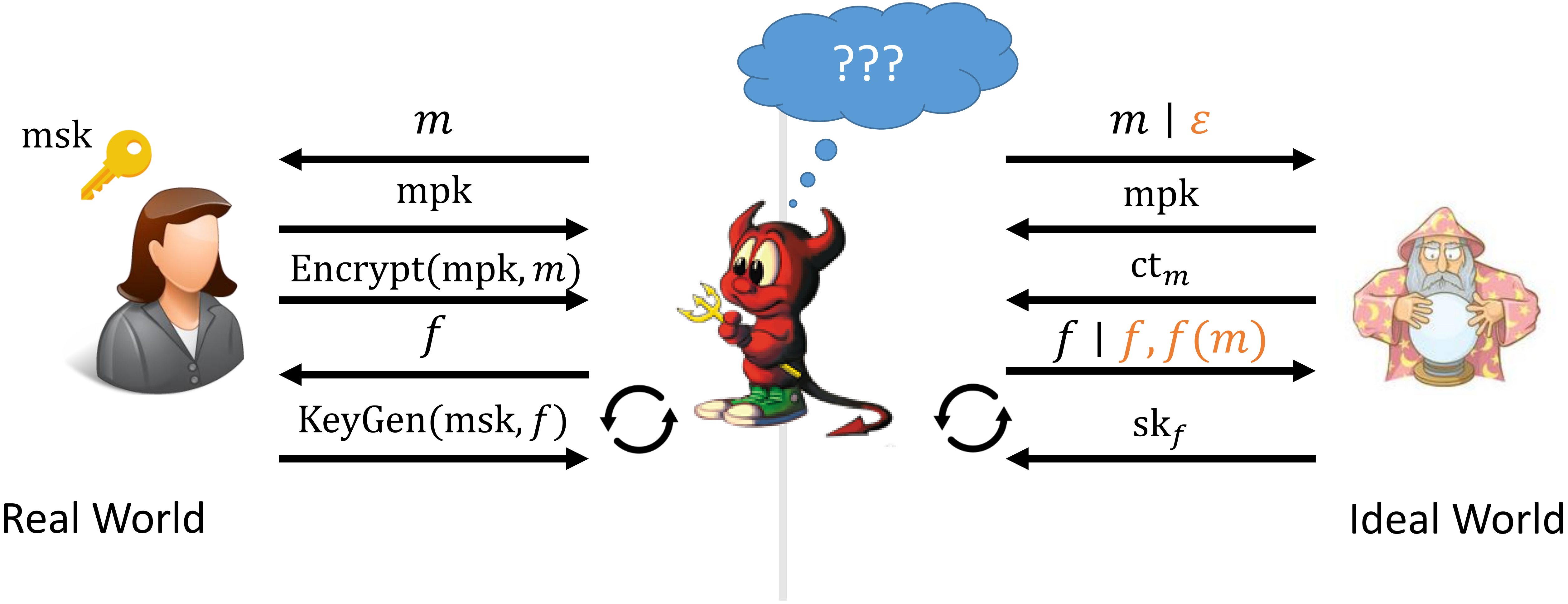
Public-Key Functional Encryption [BSW11, O'N10]

Simulation-based notion of security:



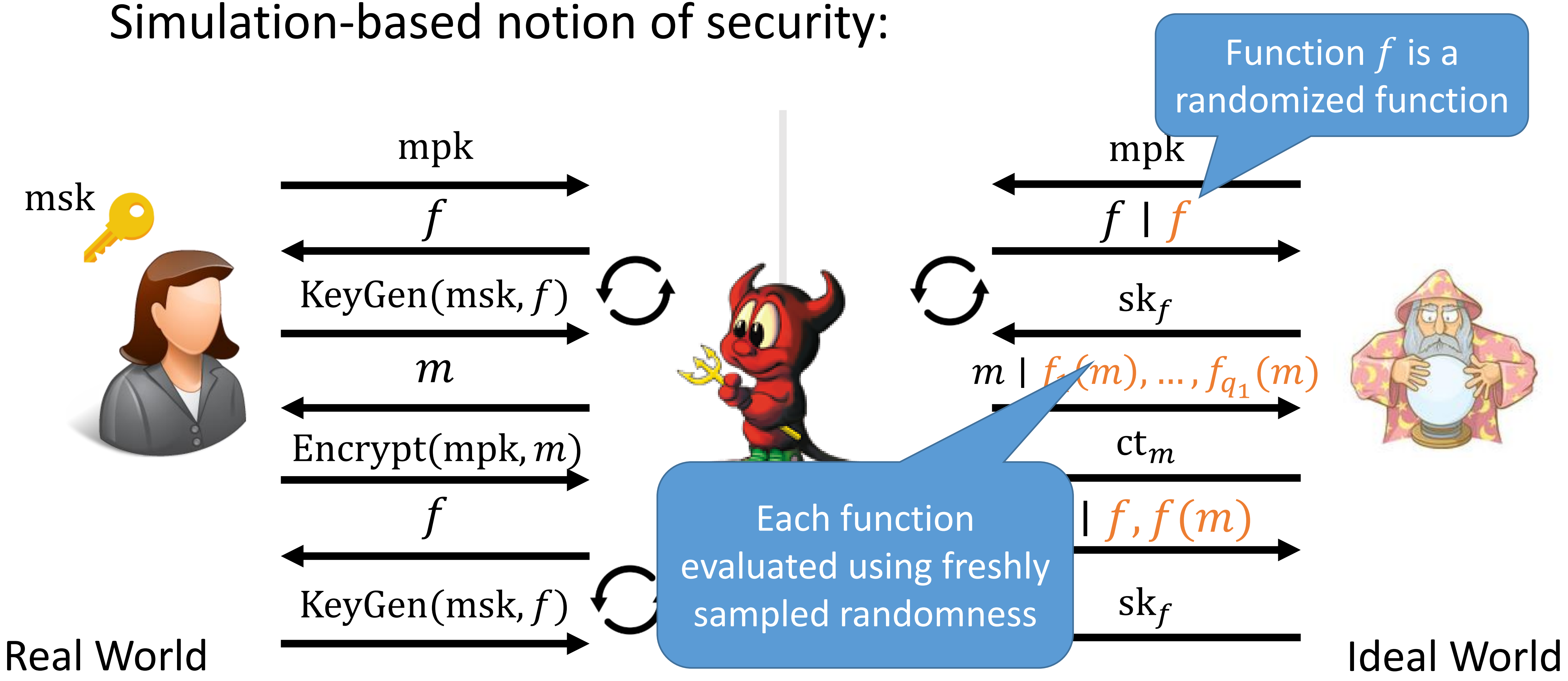
Public-Key Functional Encryption [BSW11, O'N10]

Selective security: adversary first commits to challenge



Security for rFE

Simulation-based notion of security:



The Case for Malicious Encrypters

Encrypted database of records

r_3 | r_4 | r_5 | r_6

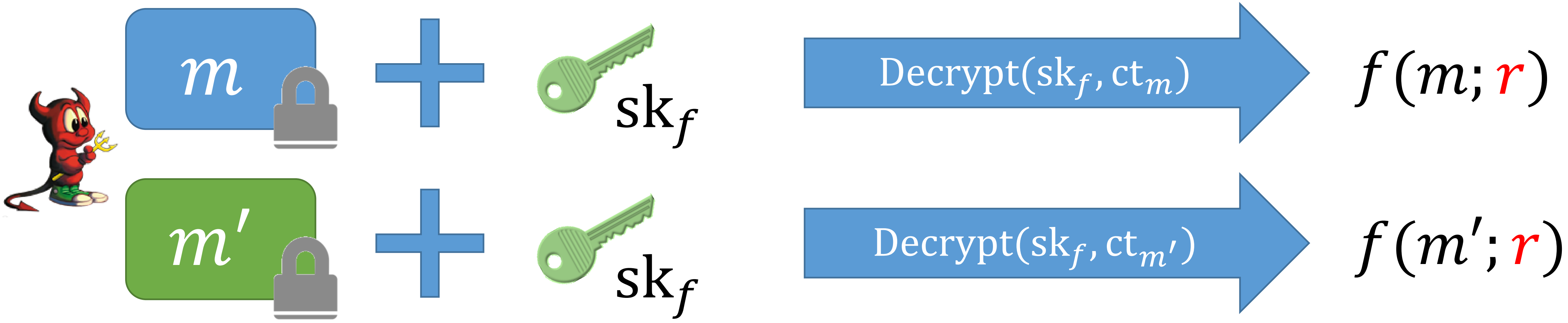
What if encrypter (bank) is adversarial?

r_2 | 6

Sample a *random* subset to audit

The Case for Malicious Encrypters

Randomized functionalities



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

Capturing Dishonest Encrypters

Give the adversary access to a decryption oracle (a “CCA” like definition) [GJKS15]

msk

ct, f

$\mathrm{sk}_f \leftarrow \mathrm{KeyGen}(\mathrm{msk}, f)$

$m \leftarrow \mathrm{Decrypt}(\mathrm{sk}_f, \mathrm{ct})$

m

ct, f | ct

Capturing Dishonest

Give the adversary access to a decryption (
definition) [GJKS15]

> Simulator sees ciphertext and can make a *single* query to an ideal evaluation oracle $\mathcal{O}_f$

msk

ct, f

$$\mathrm{sk}_f \leftarrow \mathrm{KeyGen}(\mathrm{msk}, f)$$
$$m \leftarrow \mathrm{Decrypt}(\mathrm{sk}_f, \mathrm{ct})$$

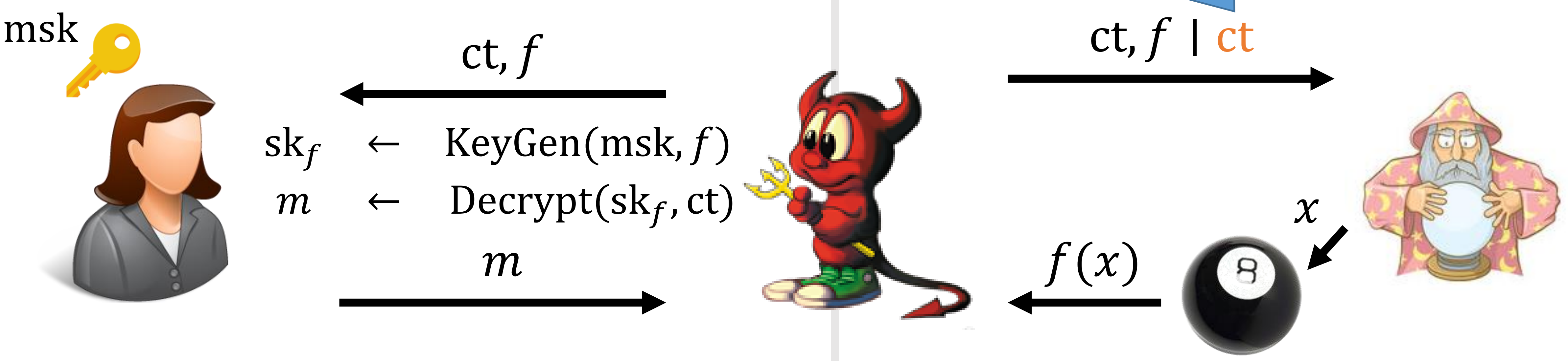
m

ct, f | ct

Capturing Dishonest

Give the adversary access to a decryption
definition) [GJKS15]

Simulator sees ciphertext and can make a *single* query to an ideal evaluation oracle $\mathcal{O}_f$



Capturing Dishonest

Give the adversary access to a decryption
definition) [GJKS15]

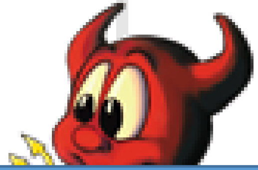
Simulator sees ciphertext and can make a *single* query to an ideal evaluation oracle $\mathcal{O}_f$

msk

ct, f

$\mathrm{sk}_f \leftarrow \mathrm{KeyGen}(\mathrm{msk}, f)$

m



ct, f | ct

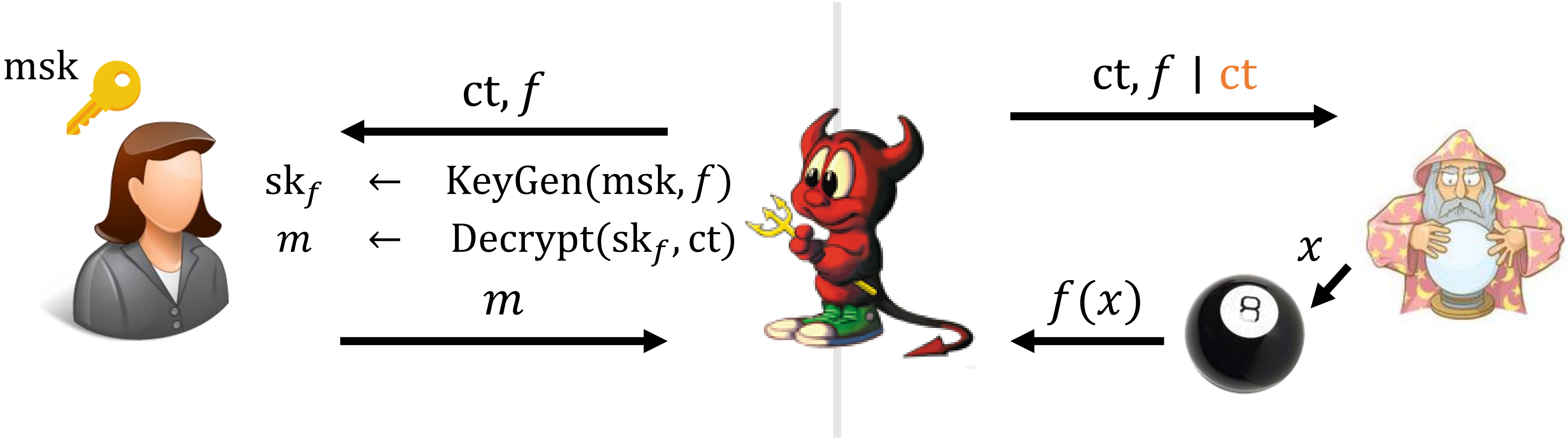
x

$f(x)$

Ideal evaluation oracle $\mathcal{O}_f$ takes an input x and outputs random draw from output distribution $f(x)$

Capturing Dishonest Encrypters

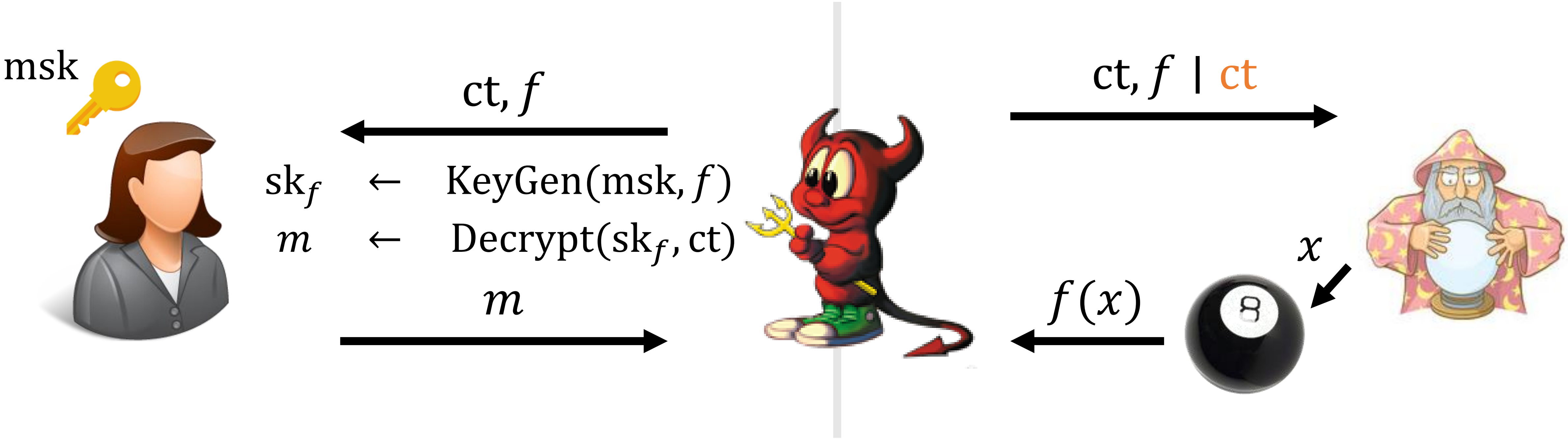
Give the adversary access to a decryption oracle (a "CCA" like definition) [GJKS15]



Note: in ideal world, distinguisher *always* sees a function evaluation using uniform randomness

Capturing Dishonest Encrypters

Give the adversary access to a decryption oracle (a “CCA” like definition) [GJKS15]



Notion also well-defined in deterministic setting and is easily achieved by attaching a NIZK to ciphertext

Capturing Dishonest Encrypters

This work: Extend security model to allow adversary to submit *multiple* ciphertexts (rules out adversary's ability to construct *correlated* ciphertexts)

msk

$$\xleftarrow{\{\mathrm{ct}_i\}_{i\in[n]},\, f}$$

$$\mathrm{sk}_f \leftarrow \mathrm{KeyGen}(\mathrm{msk}, f)$$

$$m_i \leftarrow \mathrm{Decrypt}(\mathrm{sk}_f, \mathrm{ct}_i)$$

$$\xrightarrow{m_1, \dots, m_n}$$

Capturing Dishonest Encrypters

<u>**This work**</u>: Extend security model to allow adversary to submit *multiple* ciphertexts (rules out adversary's ability to construct *correlated* ciphertexts)

msk

$$\xleftarrow{\{\mathrm{ct}_i\}_{i\in[n]},\, f}$$

$$\mathrm{sk}_f \leftarrow \mathrm{KeyGen}(\mathrm{msk}, f)$$

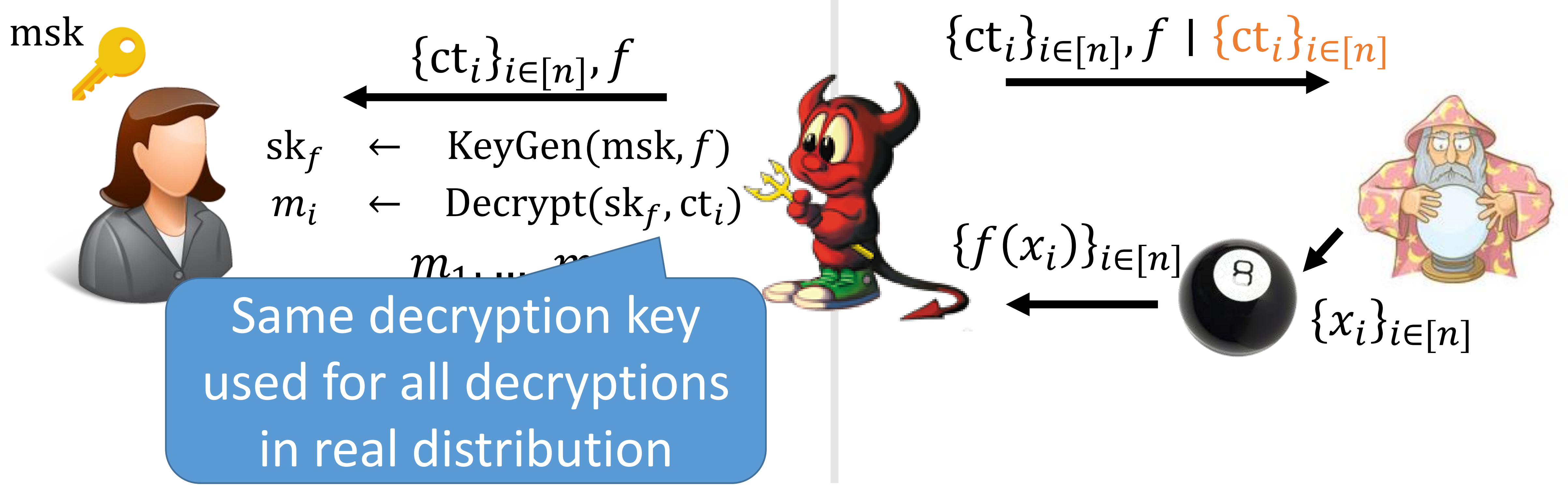
$$m_i \leftarrow \mathrm{Decrypt}(\mathrm{sk}_f, \mathrm{ct}_i)$$

$$m_1, \dots, m_n$$

Same decryption key used for all decryptions in real distribution

Capturing Dishonest Encrypters

<u>**This work**</u>: Extend security model to allow adversary to submit *multiple* ciphertexts (rules out adversary's ability to construct *correlated* ciphertexts)



Capturing Dishonest Encrypters

This work: Extend security model to allow adversary to submit *multiple* ciphertexts (rules out adversary's ab

ciphertexts)

Ideal evaluation oracle $\mathcal{O}_f$ takes vector of inputs x_i and for each input, outputs random draw from $f(x_i)$

msk

$\{\mathrm{ct}_i\}_{i \in [n]}, f$

$\mathrm{sk}_f \leftarrow \mathrm{KeyGen}(\mathrm{msk}, f)$

$m_i \leftarrow \mathrm{Decrypt}(\mathrm{sk}_f, \mathrm{ct}_i)$

$m_1, \ldots$

$\{f(x_i)\}_{i \in [n]}$

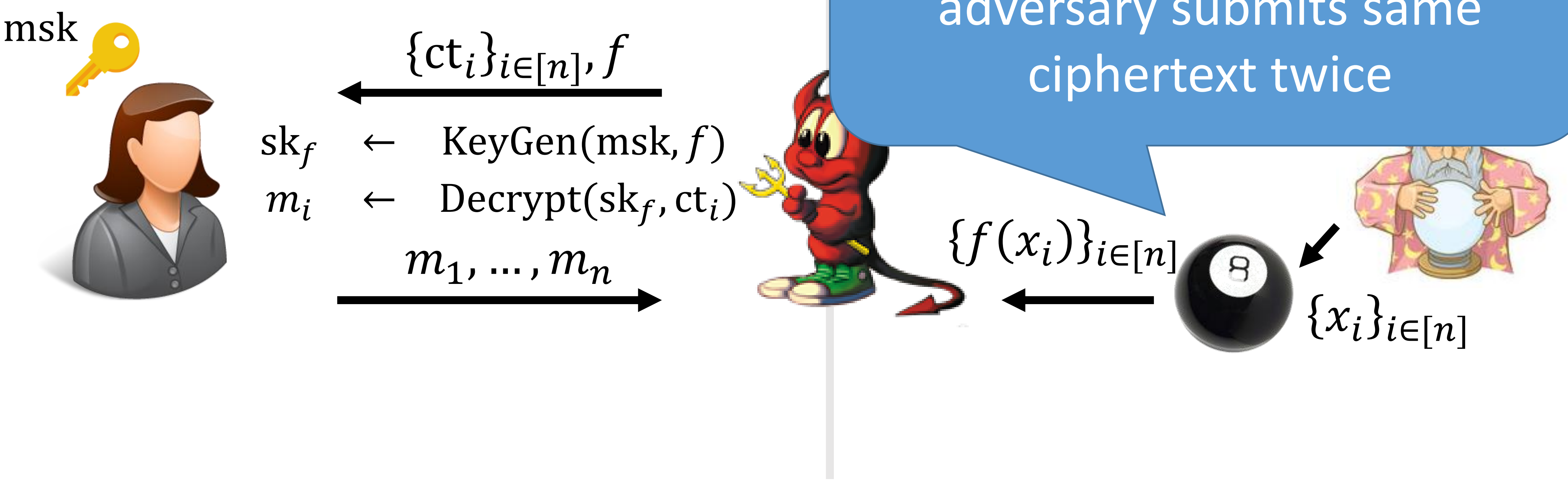
$\{x_i\}_{i \in [n]}$

Same decryption key used for all decryptions in real distribution

Capturing Dishonest Encrypters

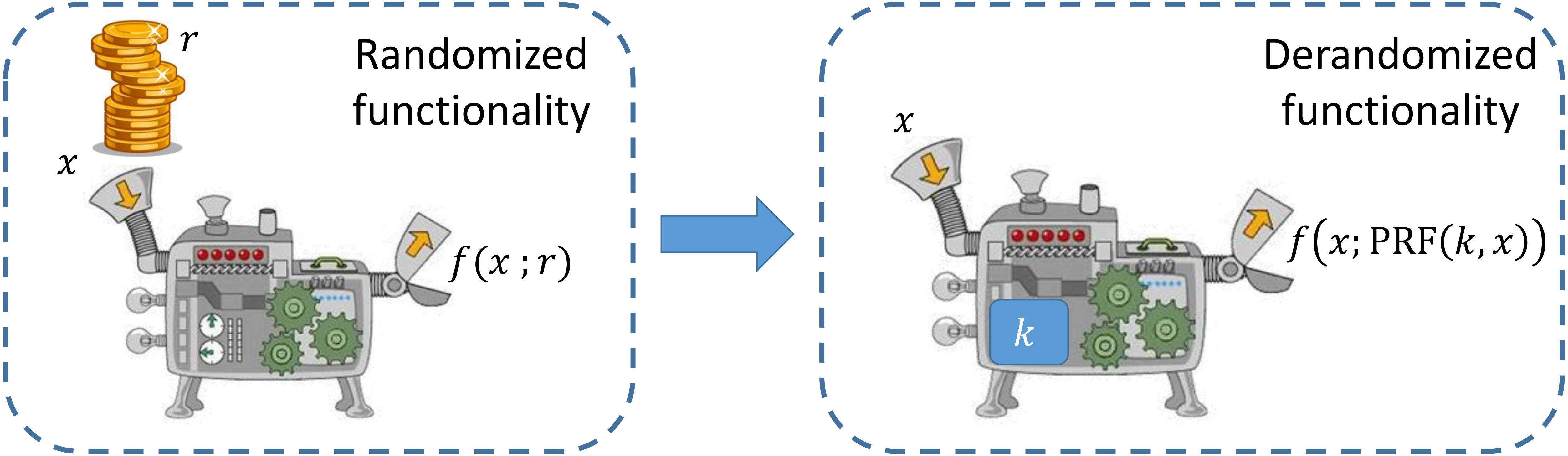
This work: Extend security model to allow adversary to submit *multiple* ciphertexts (rules out adversary's ab ciphertexts)

Impose admissibility criterion to rule out cases where adversary submits same ciphertext twice



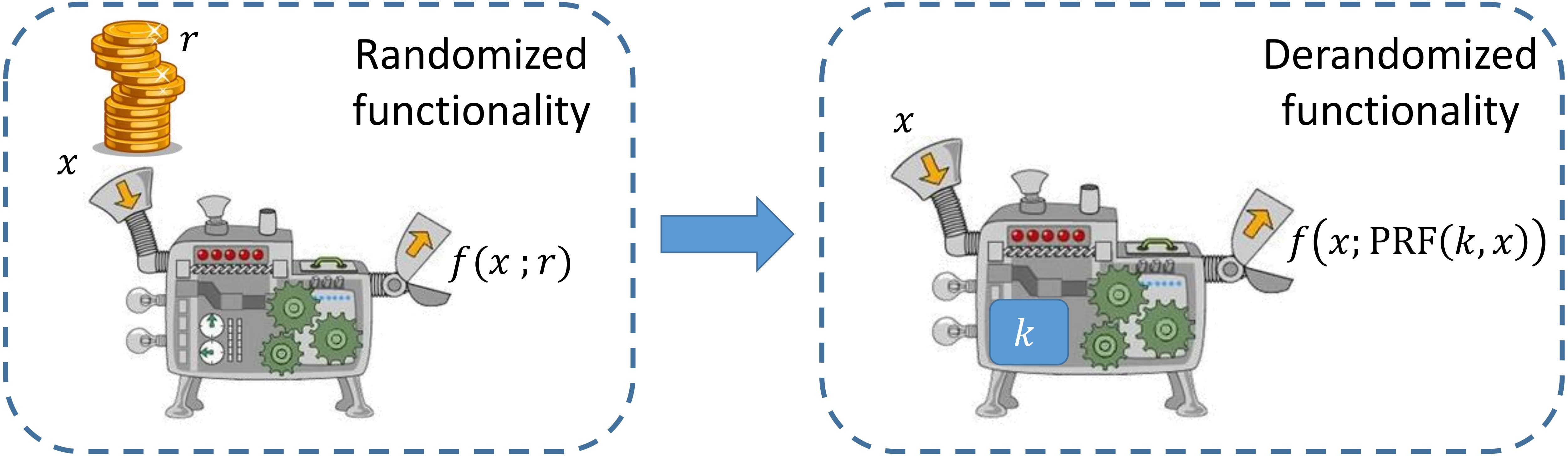
Our Generic Transformation

Starting Point: Derandomization



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

Starting Point: Derandomization

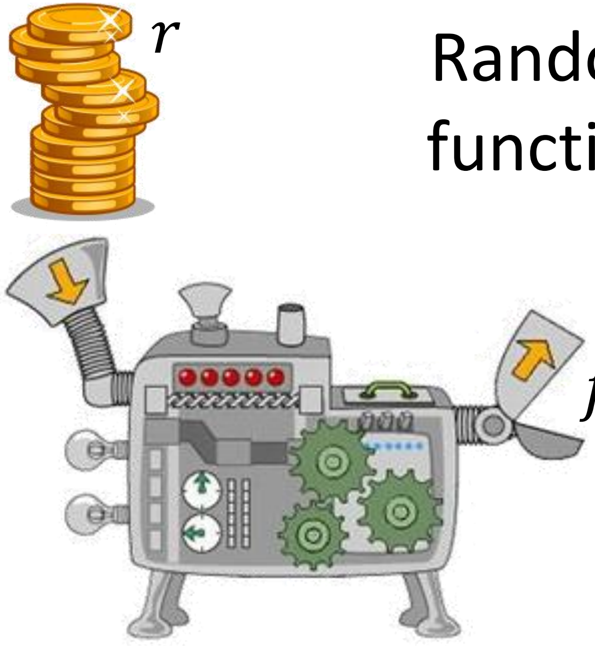


Randomized function f

Derandomized function g_k:

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

Starting Point: Derandomization



Randomized function f

PRF key embedded inside g_k

ndomized function g_k:

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

Starting Point: Derandomization

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

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

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

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

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

sk_{g_k}

Starting Point: Derandomization

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

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

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

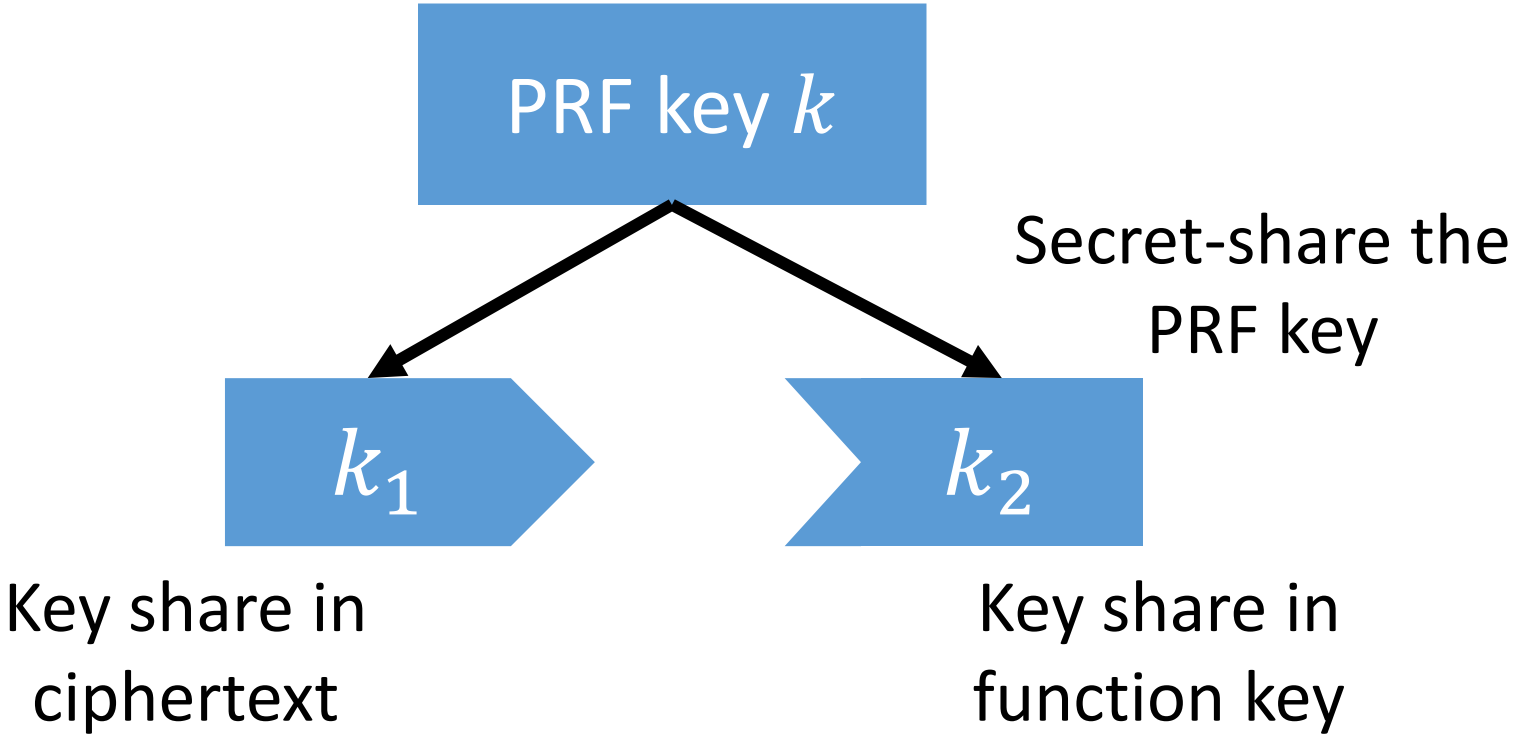
$g_k(x) = f\big(x, \mathrm{PRF}(k, x)\big)$

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

sk_{g_k}

How to Hide the Key?

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



How to Hide the Key?

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

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

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

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

(m, k_1)

How to Hide the Key?

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

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

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

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

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 <u>ciphertext</u>

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

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

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

Security now relies on <u>related-key security</u> for PRFs

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

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

Why Related-Key Security?

Challenge ciphertext:

(m, k_1)

Chosen by adversary

Chosen by challenger

Adversary knows m, k_2 and k_2' and outputs still look random!

Secret key queries:

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

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

Adversary sees

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

and

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

Security Against Dishonest Encrypters

Encrypter has a lot of flexibility in constructing ciphertexts:

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

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

Encrypter can choose the key-share

Cannot influence output distribution due to RKA-security

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

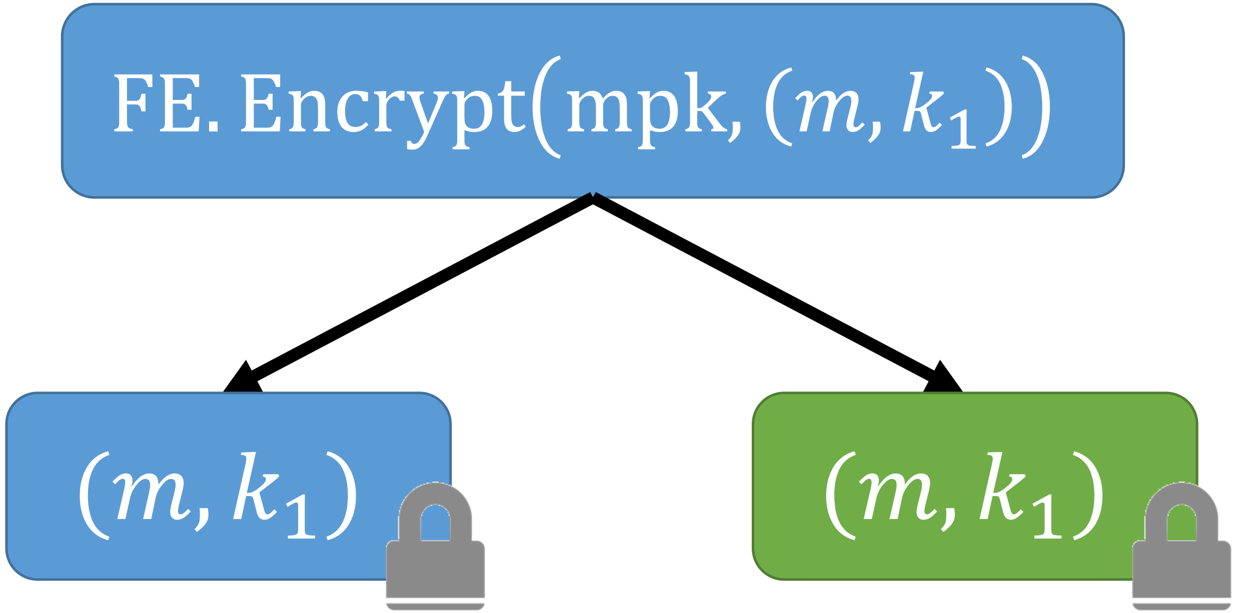
Encrypter can choose the randomness

(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:

(m, k_1)

Decryption in real world: always produces *same* output

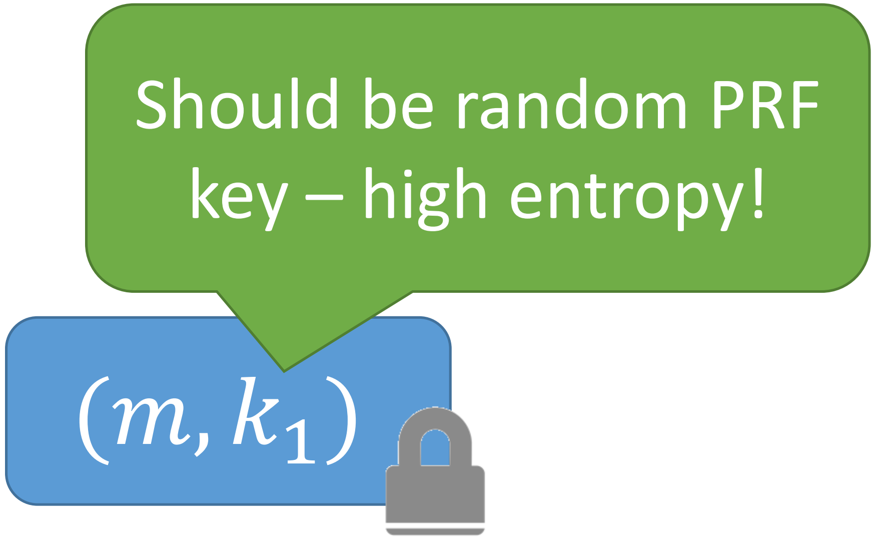
(m, k_1)

Decryption in ideal world: always produces independent outputs

Encrypter has <u>too much</u> freedom in constructing ciphertexts

Applying Deterministic Encryption

<u>Key observation:</u> 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

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

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

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

$\mathrm{FE.Encrypt}(\mathrm{mpk}, (m, k_1)\,; h(k_1))$ $+$ π

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

Putting the Pieces Together

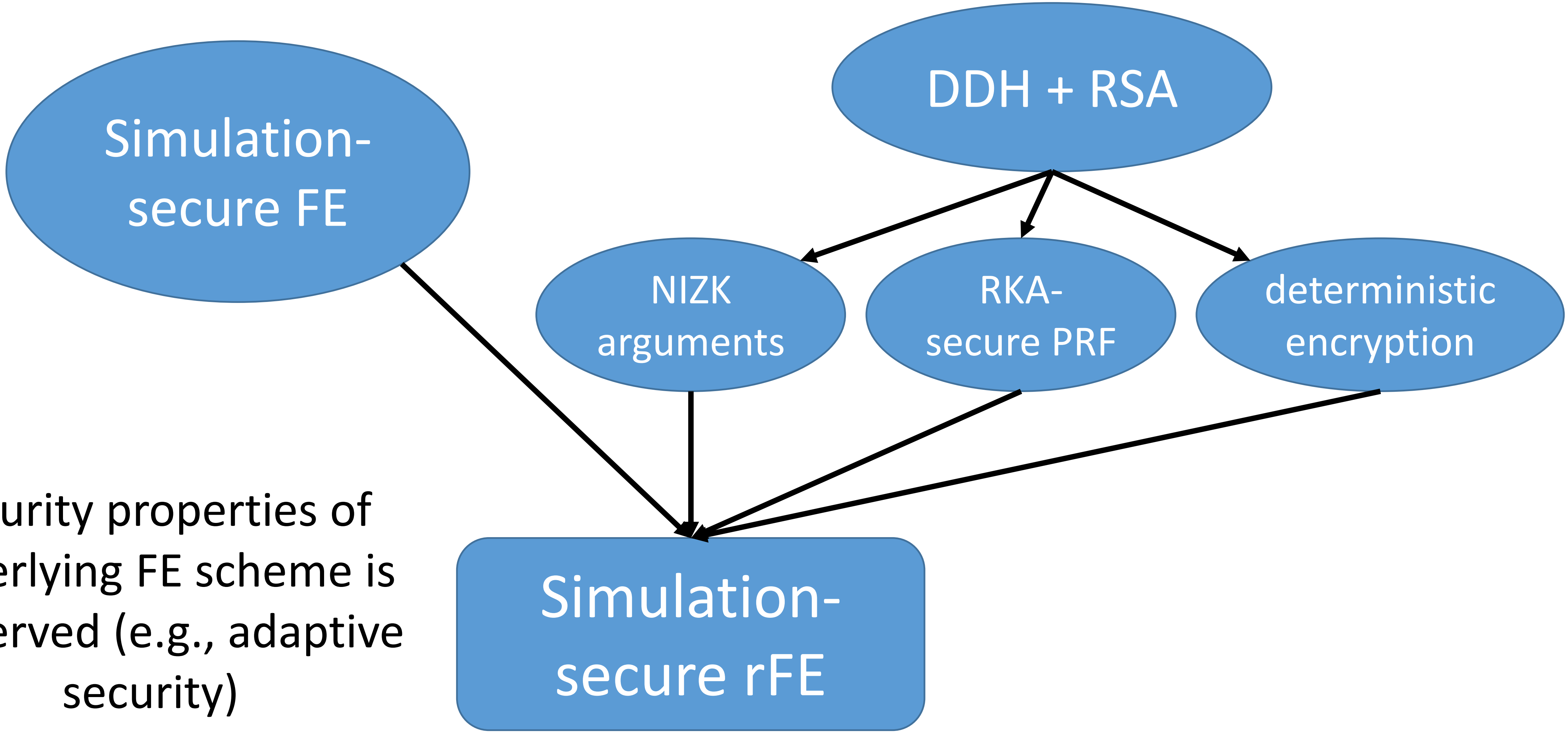
$\mathrm{rFE.Encrypt}(\mathrm{mpk}, m)$

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

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

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, …]

PKE → Bounded-Collusion FE

[GGHRSW13, GGHZ16, …]

Multilinear Maps / iO → General-Purpose FE

Generally adaptively secure

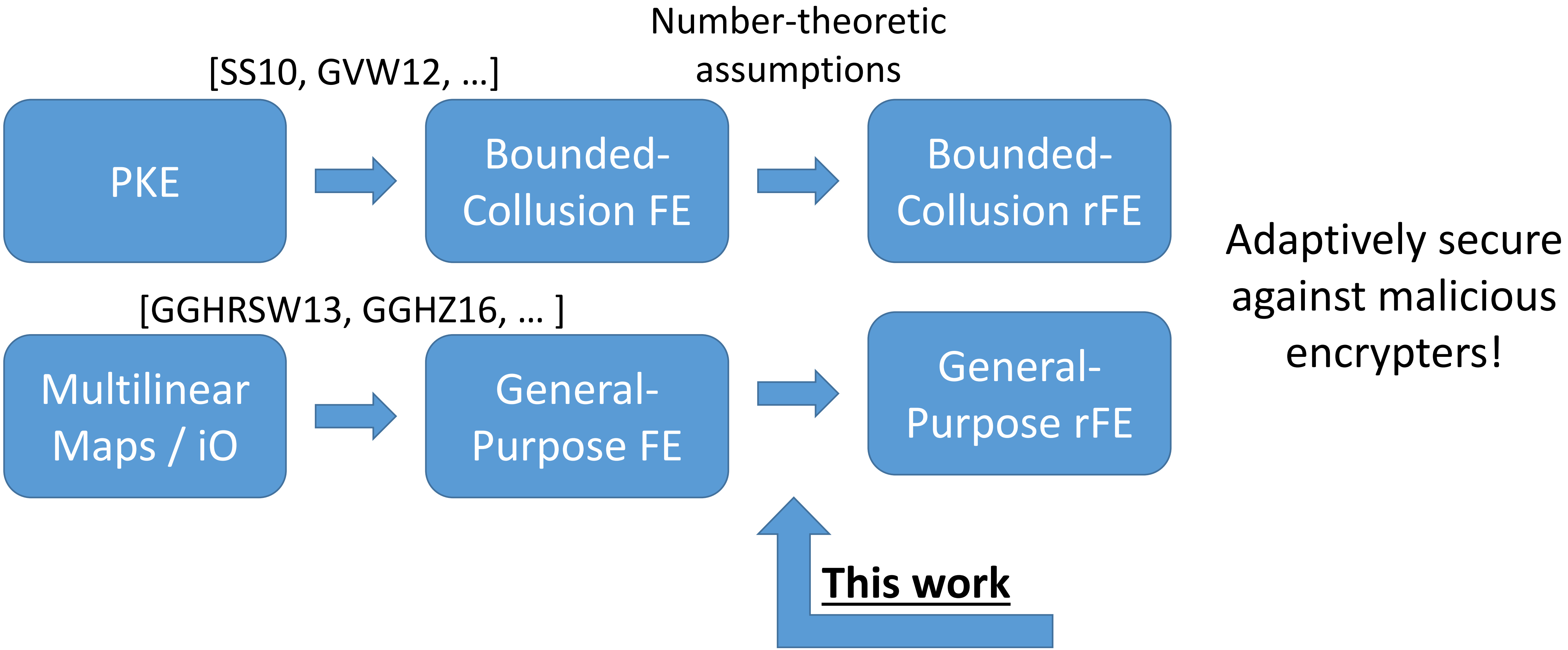
Randomized functionalities

[GJKS15]

iO → General-Purpose rFE

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 from a database)?
- Generic construction of rFE from FE *without* making additional assumptions?
- Connections between rFE and other primitives (e.g., various flavors of obfuscation)?

Questions?

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