Robust Decentralized Multi-Client Functional Encryption:

Motivation, Definition, and Inner-Product Constructions

Yamin Li, Jianghong Wei, Fuchun Guo, Willy Susilo and Xiaofeng Chen*

西安電子科技大學
XIDIAN UNIVERSITY

State Key Laboratory of Mathematical Engineering and Advanced Computing

UNIVERSITY OF WOLLONGONG AUSTRALIA

ASIACRYPT 2023

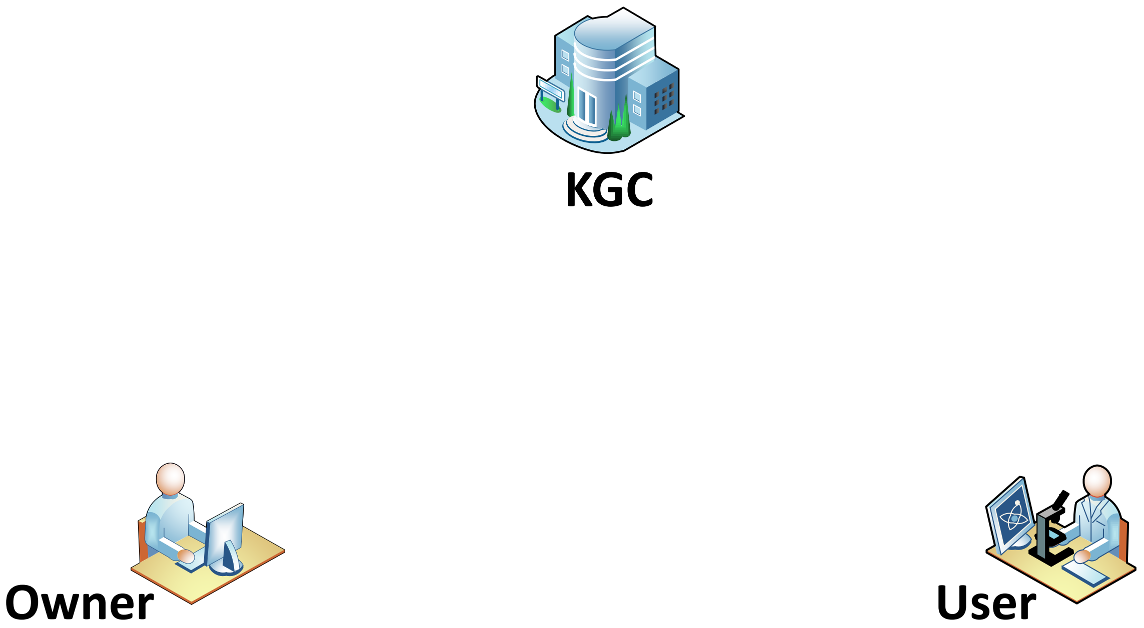
Outline

1. Introduction
2. Motivation
3. Definition (RDMCFE)
4. IP-RDMCFE Constructions
5. Conclusion

1. Introduction

Functional Encryption (FE) [BSW 11, O'N 10]

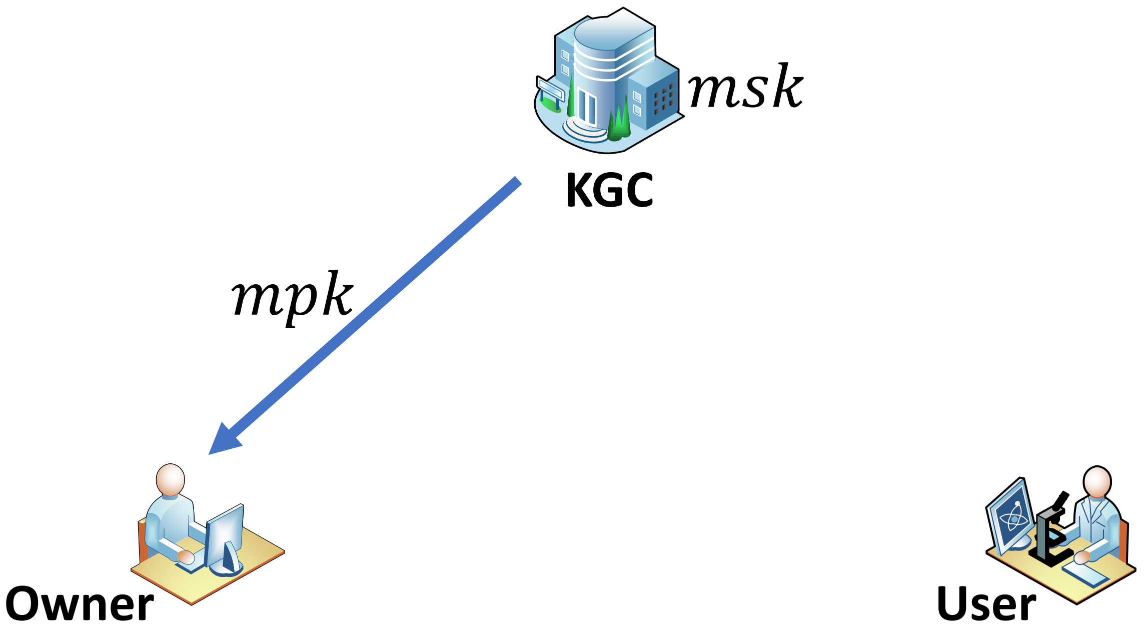
- $(mpk, msk) \leftarrow Setup(1^{\lambda}, \mathcal{F})$
- $Ct \leftarrow Enc(mpk, x)$
- $sk_f \leftarrow KeyGen(msk, f)$
- $f(x) \leftarrow Dec(sk_f, Ct)$



1. Introduction

Functional Encryption (FE) [BSW 11, O'N 10]

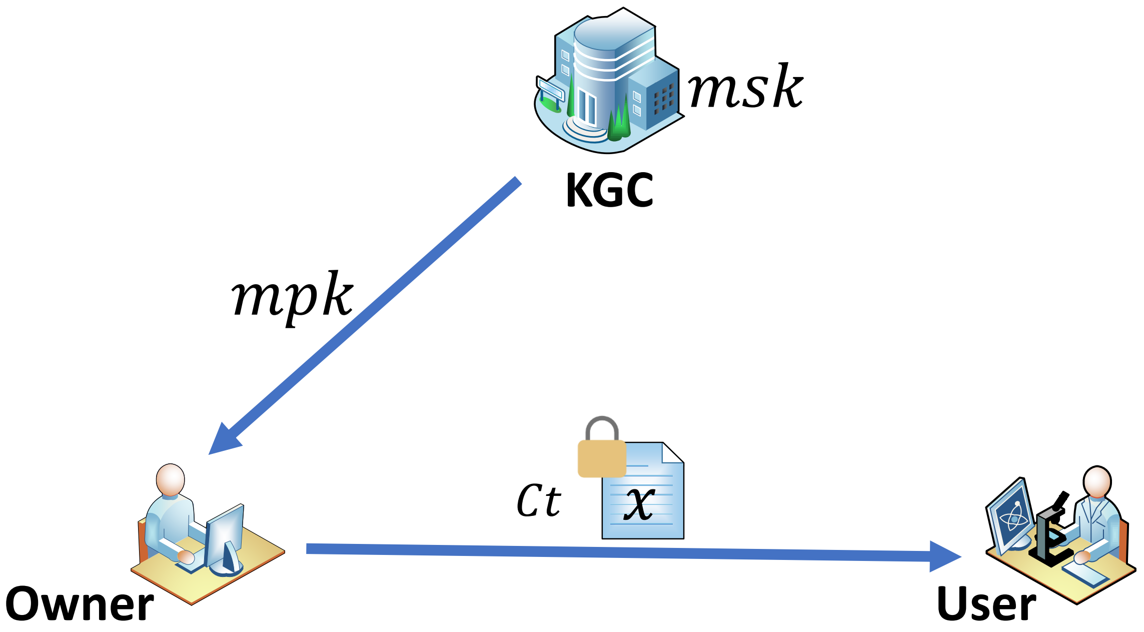
- $(mpk, msk) \leftarrow Setup(1^\lambda, \mathcal{F})$
- $Ct \leftarrow Enc(mpk, x)$
- $sk_f \leftarrow KeyGen(msk, f)$
- $f(x) \leftarrow Dec(sk_f, Ct)$



1. Introduction

Functional Encryption (FE) [BSW 11, O'N 10]

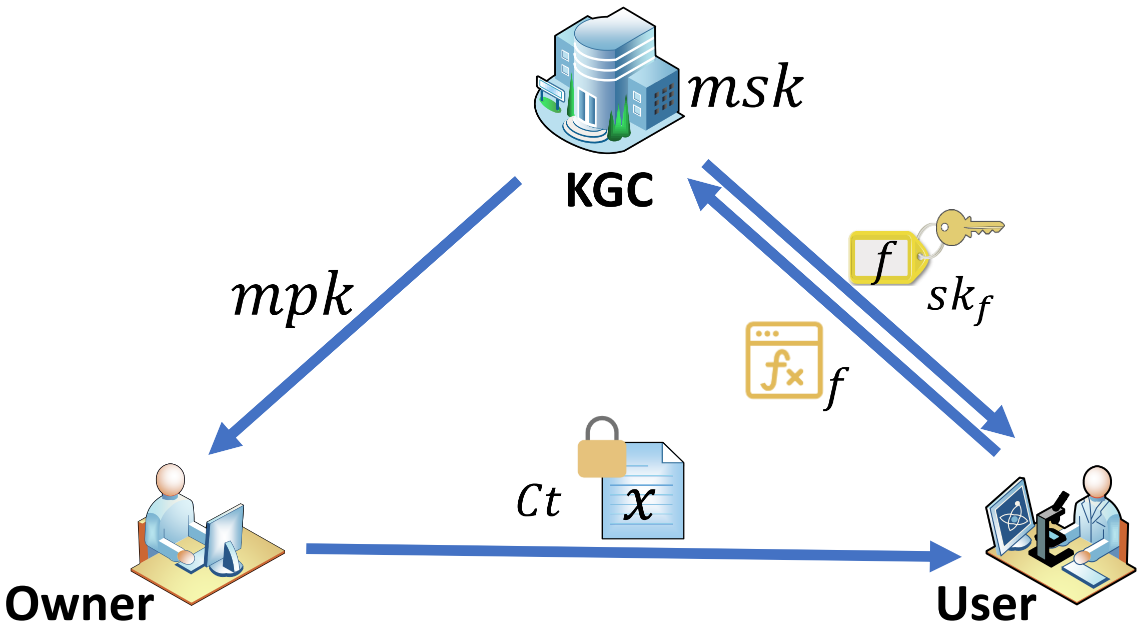
- $(mpk, msk) \leftarrow Setup(1^{\lambda}, \mathcal{F})$
- $Ct \leftarrow Enc(mpk, x)$
- $sk_f \leftarrow KeyGen(msk, f)$
- $f(x) \leftarrow Dec(sk_f, Ct)$



1. Introduction

Functional Encryption (FE) [BSW 11, O'N 10]

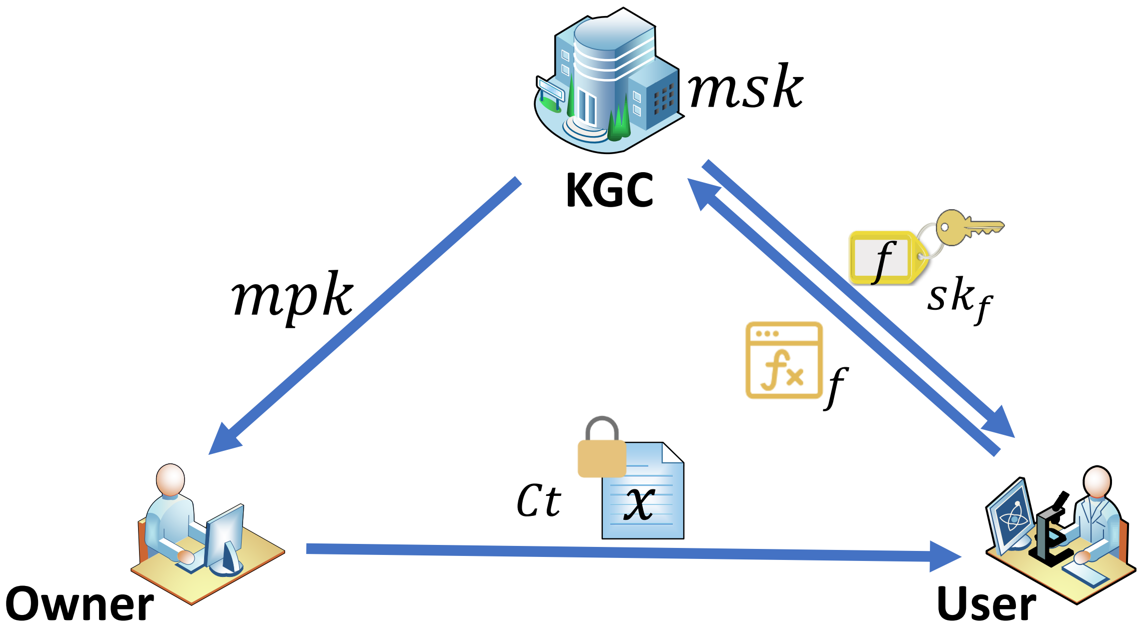
- $(mpk, msk) \leftarrow Setup(1^{\lambda}, \mathcal{F})$
- $Ct \leftarrow Enc(mpk, x)$
- $sk_f \leftarrow KeyGen(msk, f)$
- $f(x) \leftarrow Dec(sk_f, Ct)$



1. Introduction

Functional Encryption (FE) [BSW 11, O'N 10]

- $(mpk, msk) \leftarrow Setup(1^{\lambda}, \mathcal{F})$
- $Ct \leftarrow Enc(mpk, x)$
- $sk_f \leftarrow KeyGen(msk, f)$
- $f(x) \leftarrow Dec(sk_f, Ct)$

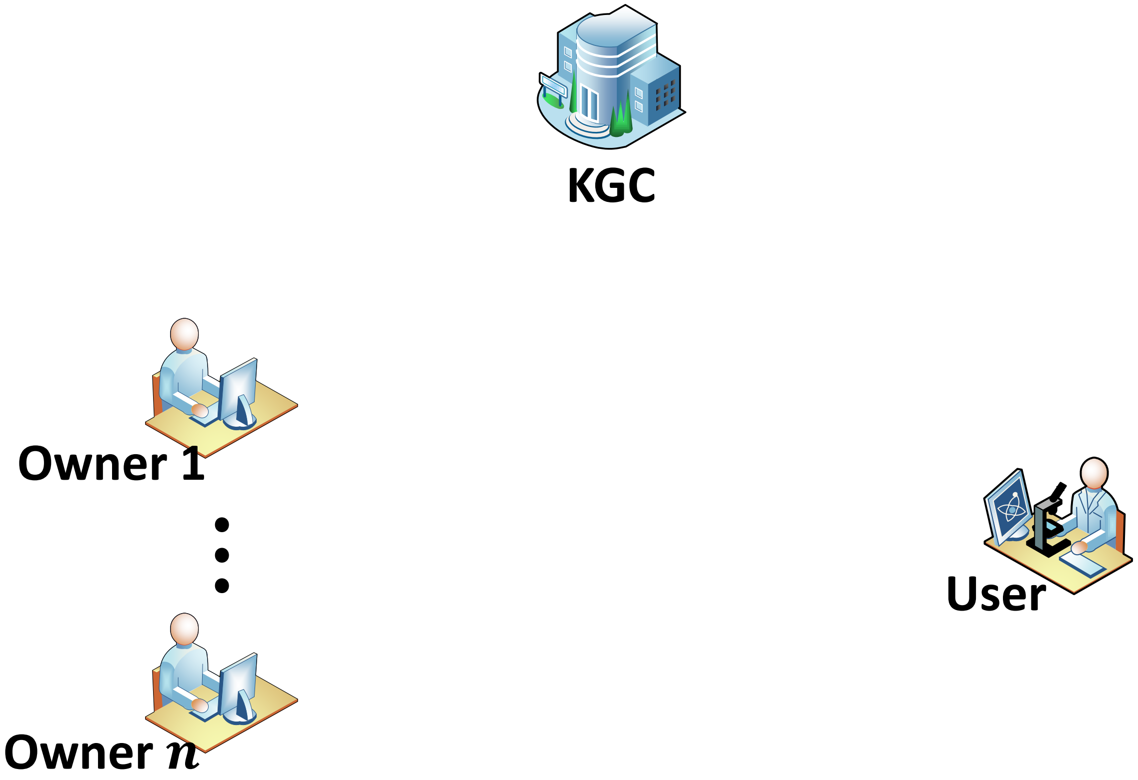


Data User can only learn $f(x)$ and nothing else about x.

1. Introduction

Multi-Client Functional Encryption (MCFE) [GGJS 14, GKLSZ 14]

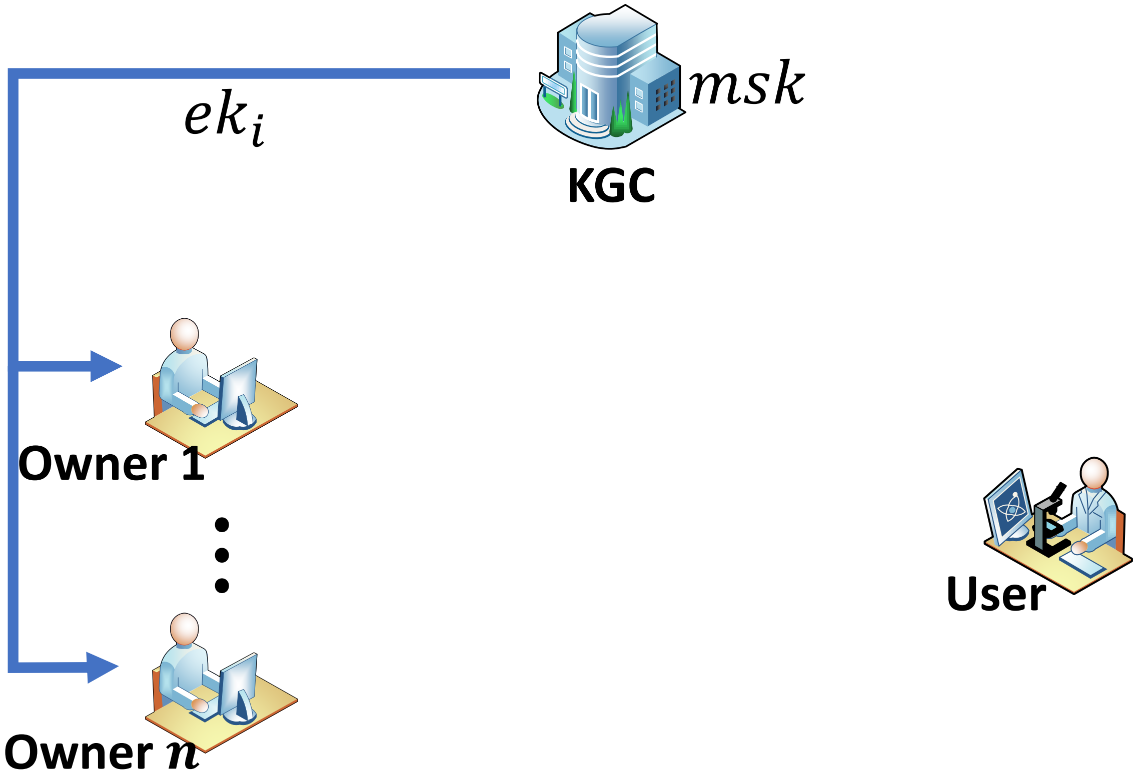
- $(pp, msk, \{ek_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n)$
- $Ct_{i,l} \leftarrow Enc(ek_i, x_i, l)$
- $sk_f \leftarrow KeyGen(msk, f)$
- $f(x_1, \cdots, x_n) \leftarrow Dec(sk_f, \{Ct_{i,l}\}_{i\in[n]})$



1. Introduction

Multi-Client Functional Encryption (MCFE) [GGJS 14, GKLSZ 14]

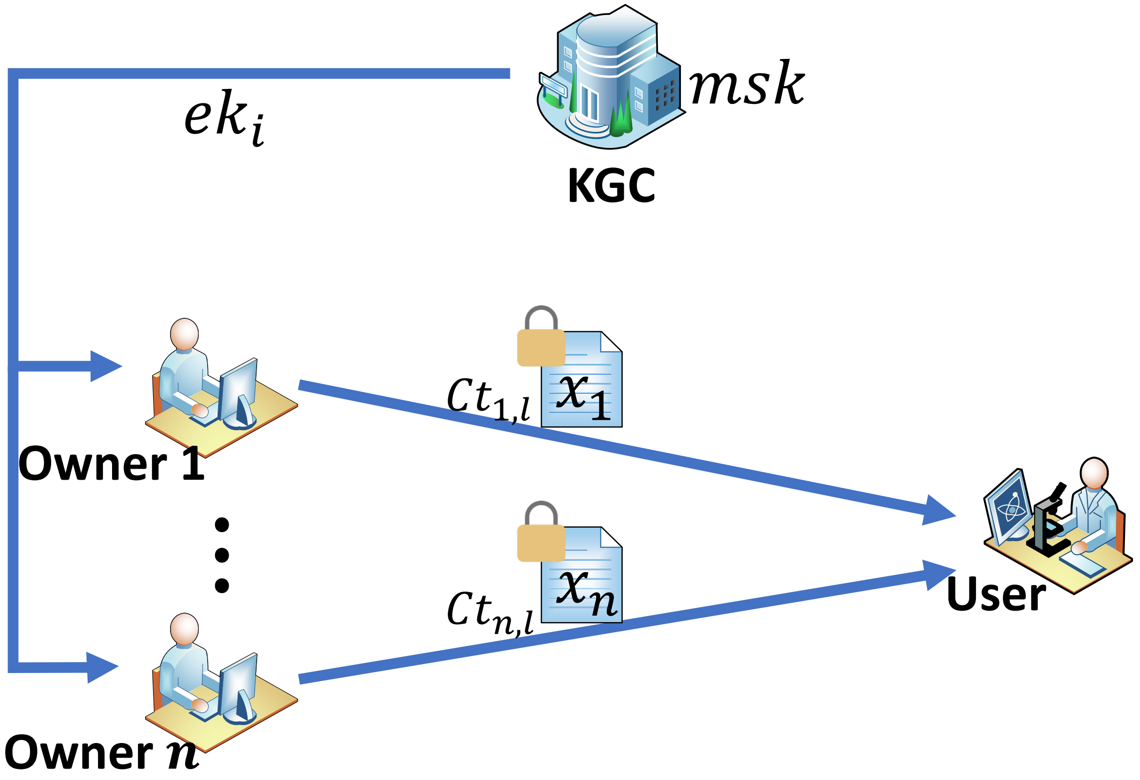
- $(pp, msk, \{ek_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n)$
- $Ct_{i,l} \leftarrow Enc(ek_i, x_i, l)$
- $sk_f \leftarrow KeyGen(msk, f)$
- $f(x_1, \cdots, x_n) \leftarrow Dec(sk_f, \{Ct_{i,l}\}_{i\in[n]})$



1. Introduction

Multi-Client Functional Encryption (MCFE) [GGJS 14, GKLSZ 14]

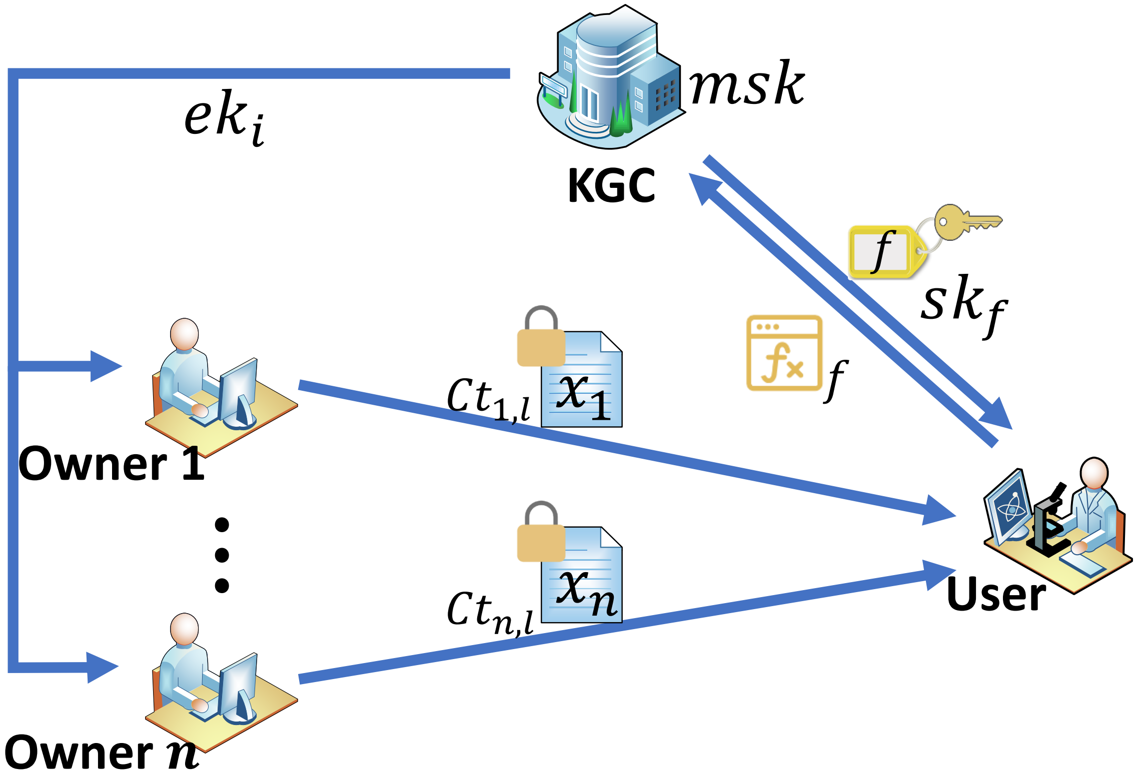
- $(pp, msk, \{ek_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n)$
- $Ct_{i,l} \leftarrow Enc(ek_i, x_i, l)$
- $sk_f \leftarrow KeyGen(msk, f)$
- $f(x_1, \cdots, x_n) \leftarrow Dec(sk_f, \{Ct_{i,l}\}_{i\in[n]})$



1. Introduction

Multi-Client Functional Encryption (MCFE) [GGJS 14, GKLSZ 14]

- $(pp, msk, \{ek_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n)$
- $Ct_{i,l} \leftarrow Enc(ek_i, x_i, l)$
- $sk_f \leftarrow KeyGen(msk, f)$
- $f(x_1, \cdots, x_n) \leftarrow Dec(sk_f, \{Ct_{i,l}\}_{i\in[n]})$

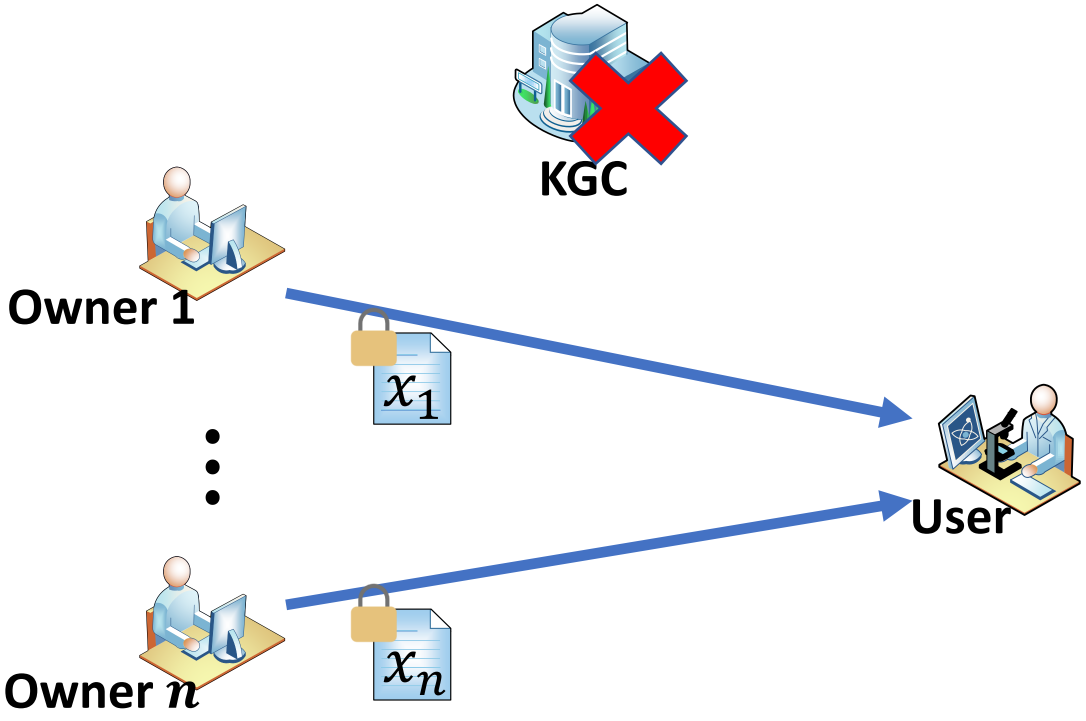


The user learns aggregate information from several different data owners.

1. Introduction

Decentralized Multi-Client Functional Encryption (DMCFE) [CSG+ 18]

- $(pp, \{ek_i\}_{i\in[n]}, \{sk_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n)$
- $Ct_{i,l} \leftarrow Enc(ek_i, x_i, l)$
- $dk_{f,i} \leftarrow KGShare(sk_i, f)$
- $sk_f \leftarrow KGComb(f, \{dk_{f,i}\}_{i\in[n]})$
- $f(x_1, \cdots, x_n) \leftarrow Dec(sk_f, \{Ct_{i,l}\}_{i\in[n]})$

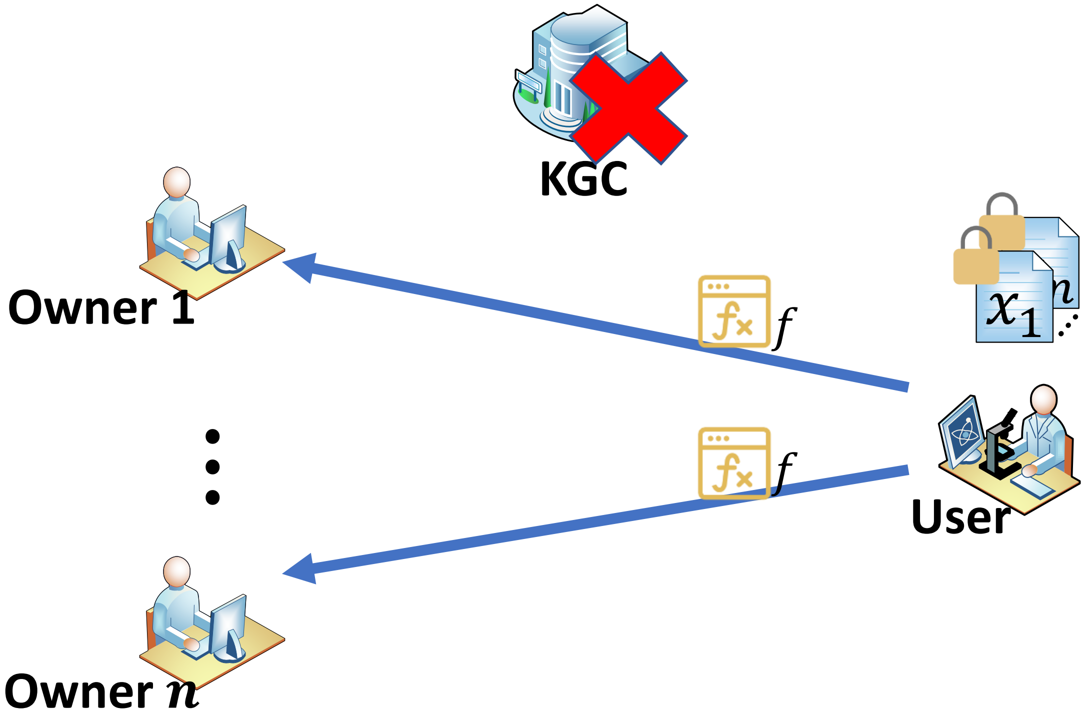


All owners interactively generate the encryption keys ek_i and the secret keys sk_i.

1. Introduction

Decentralized Multi-Client Functional Encryption (DMCFE) [CSG+ 18]

- $(pp, \{ek_i\}_{i\in[n]}, \{sk_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n)$
- $Ct_{i,l} \leftarrow Enc(ek_i, x_i, l)$
- $dk_{f,i} \leftarrow KGShare(sk_i, f)$
- $sk_f \leftarrow KGComb(f, \{dk_{f,i}\}_{i\in[n]})$
- $f(x_1, \cdots, x_n) \leftarrow Dec(sk_f, \{Ct_{i,l}\}_{i\in[n]})$

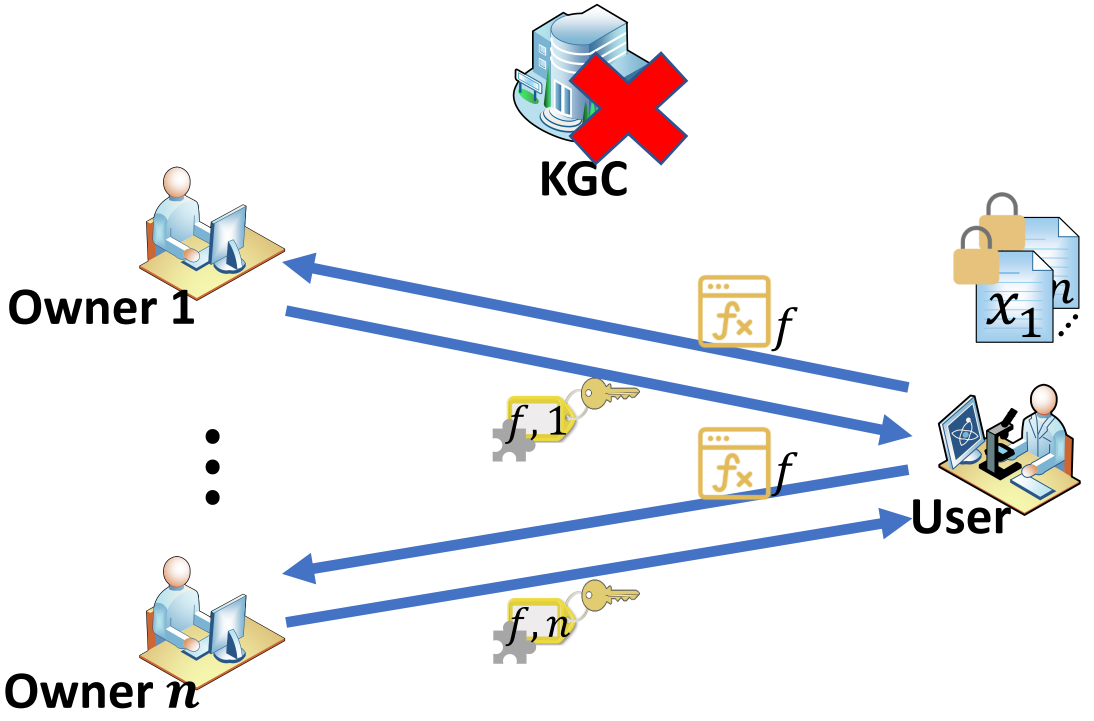


All owners interactively generate the encryption keys ek_i and the secret keys sk_i.

1. Introduction

Decentralized Multi-Client Functional Encryption (DMCFE) [CSG+ 18]

- $(pp, \{ek_i\}_{i\in[n]}, \{sk_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n)$
- $Ct_{i,l} \leftarrow Enc(ek_i, x_i, l)$
- $dk_{f,i} \leftarrow KGShare(sk_i, f)$
- $sk_f \leftarrow KGComb(f, \{dk_{f,i}\}_{i\in[n]})$
- $f(x_1, \cdots, x_n) \leftarrow Dec(sk_f, \{Ct_{i,l}\}_{i\in[n]})$

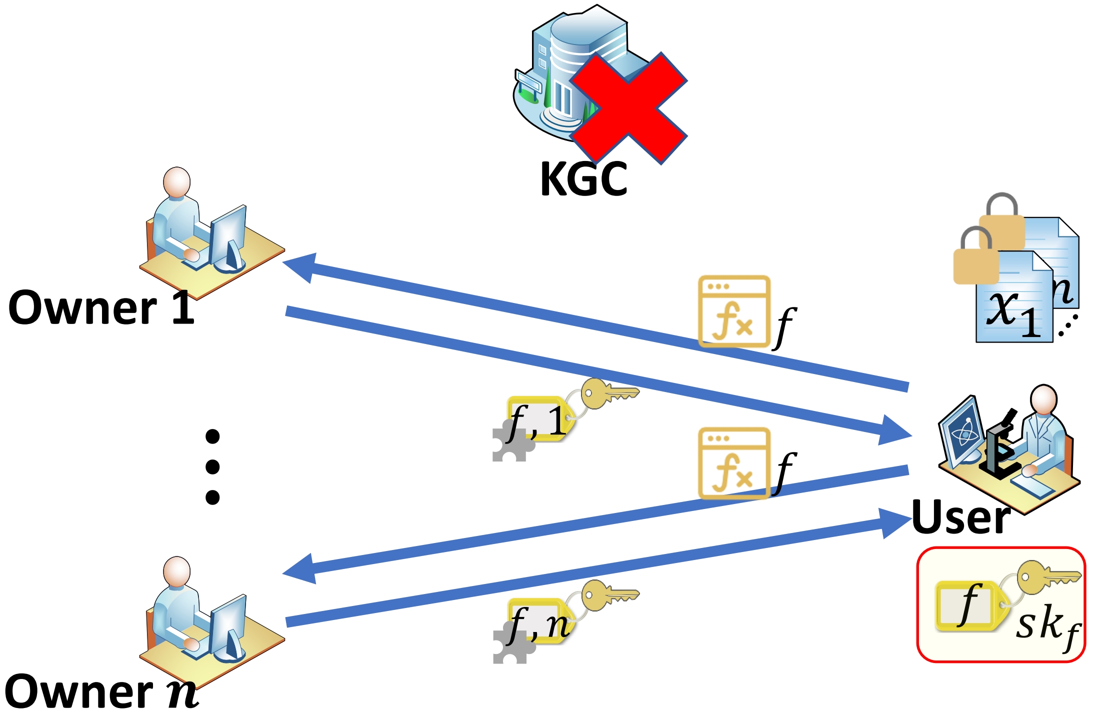


All owners interactively generate the encryption keys ek_i and the secret keys sk_i.

1. Introduction

Decentralized Multi-Client Functional Encryption (DMCFE) [CSG⁺ 18]

- $(pp, \{ek_i\}_{i\in[n]}, \{sk_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n)$
- $Ct_{i,l} \leftarrow Enc(ek_i, x_i, l)$
- $dk_{f,i} \leftarrow KGShare(sk_i, f)$
- $sk_f \leftarrow KGComb(f, \{dk_{f,i}\}_{i\in[n]})$
- $f(x_1, \cdots, x_n) \leftarrow Dec(sk_f, \{Ct_{i,l}\}_{i\in[n]})$



All owners interactively generate the encryption keys ek_i and the secret keys sk_i.

1. Introduction

The concept of functional encryption has been continuously expanded . . .

FE

- The functional key is generated by KGC.
- The user learns a specific function of encrypted data from single owner.

MCFE

- The functional key is generated by KGC.
- The user learns aggregate information from several different data owners.

DMCFE

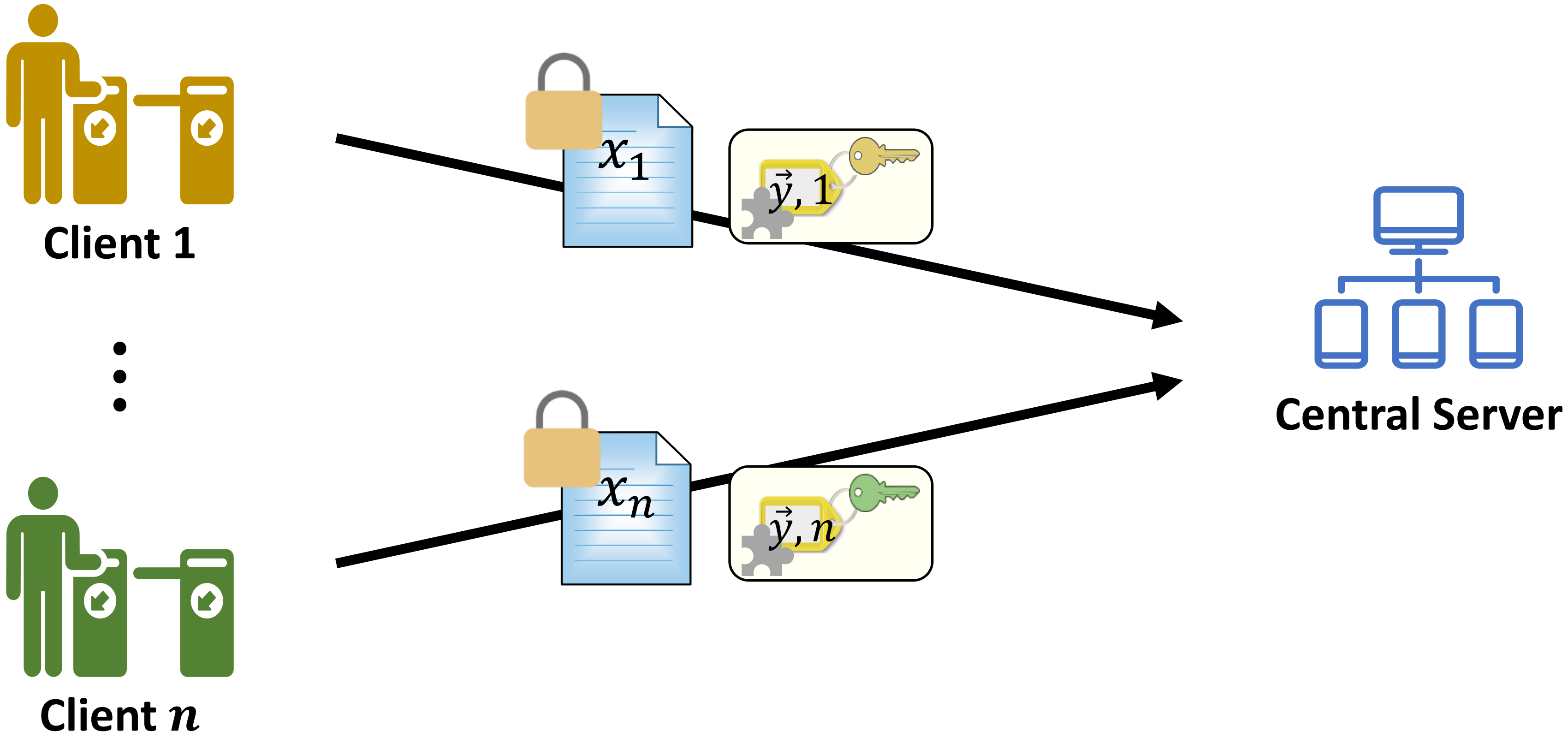
- The functional key is generated under the control of all owners themselves.
- The user learns aggregate information from several different data owners.

Outline

1. **Introduction**
2. **Motivation**
3. **Definition (RDMCFE)**
4. **IP-RDMCFE Constructions**
5. **Conclusion**

2. Motivation

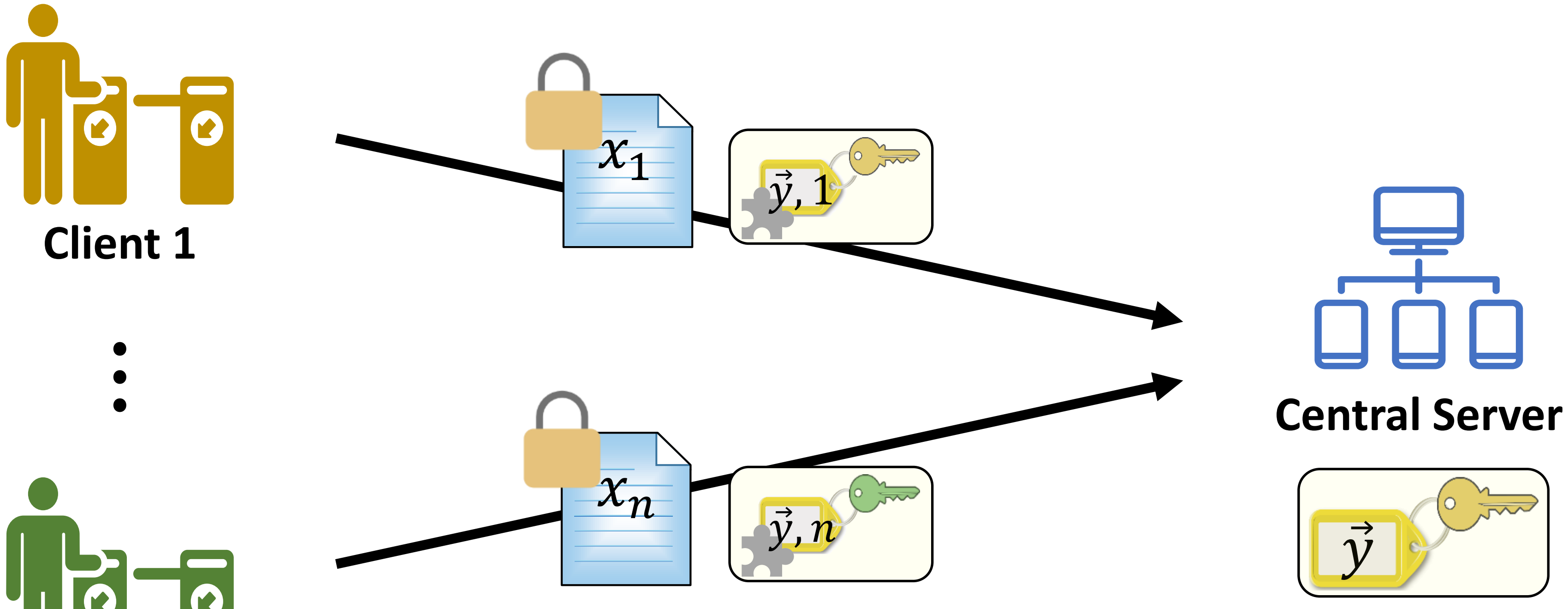
DMCFE was used to measure the traffic density at an underground station.[MSH+ 19]



Each client encrypts the location data $x_i = 0$ or 1, and provides a partial functional key for the vector $\vec{y} = (1, \cdots, 1)$.

2. Motivation

DMCFE was used to measure the traffic density at an underground station.[MSH+ 19]

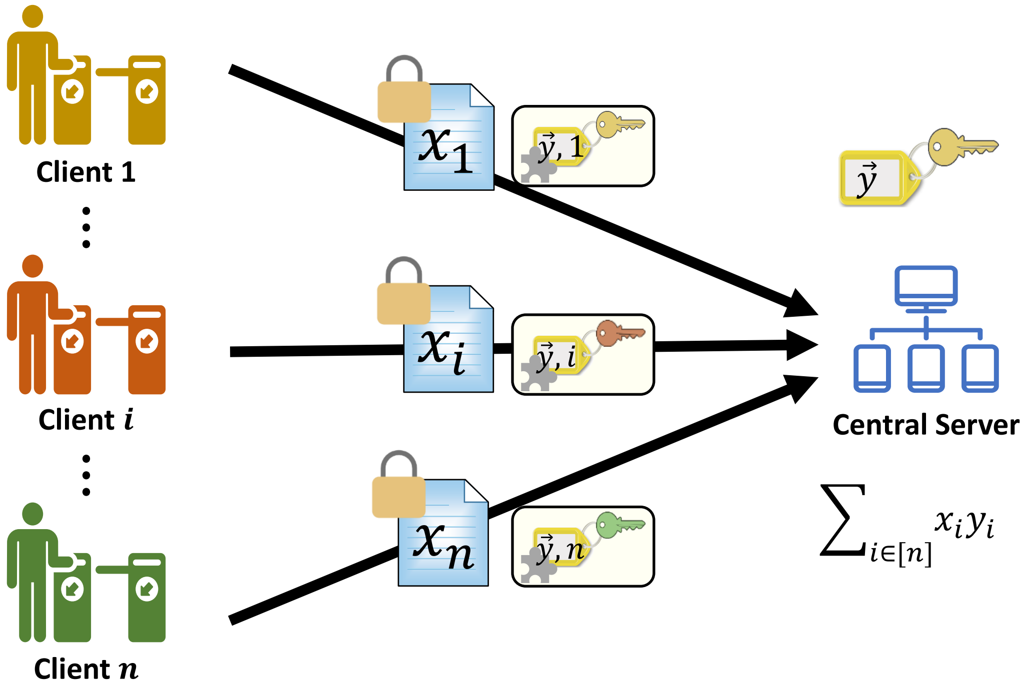


$$\langle \vec{x}, \vec{y} \rangle = \sum_{i \in [n]} x_i y_i$$

The central server obtains $\langle \vec{x}, \vec{y} \rangle$, where $\vec{x} = (x_1, x_2, \cdots, x_n)$. $\langle \vec{x}, \vec{y} \rangle$ is the traffic density, which represents the number of clients travelling through the station.

2. Motivation

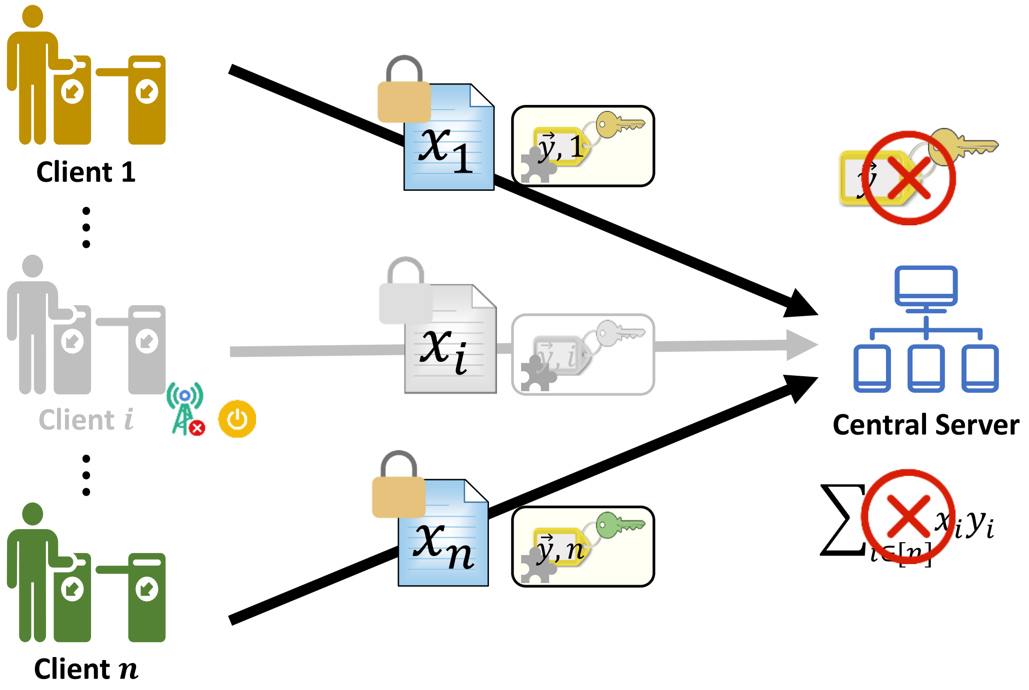
- **Limitation:**
  - The central server must collect the partial functional keys and ciphertexts from all clients.
  - Some clients accidentally go offline due to hardware issues.
  - Some clients intentionally stop participating.



2. Motivation

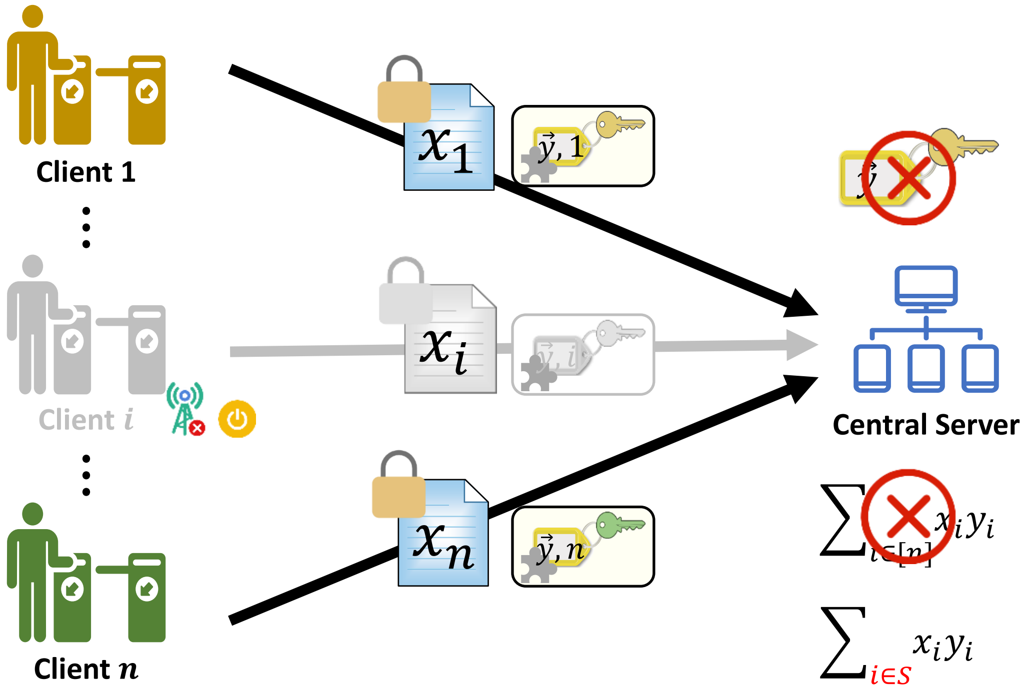
Limitation:

- The central server must collect the partial functional keys and ciphertexts from all clients.
- Some clients accidentally go offline due to hardware issues.
- Some clients intentionally stop participating.



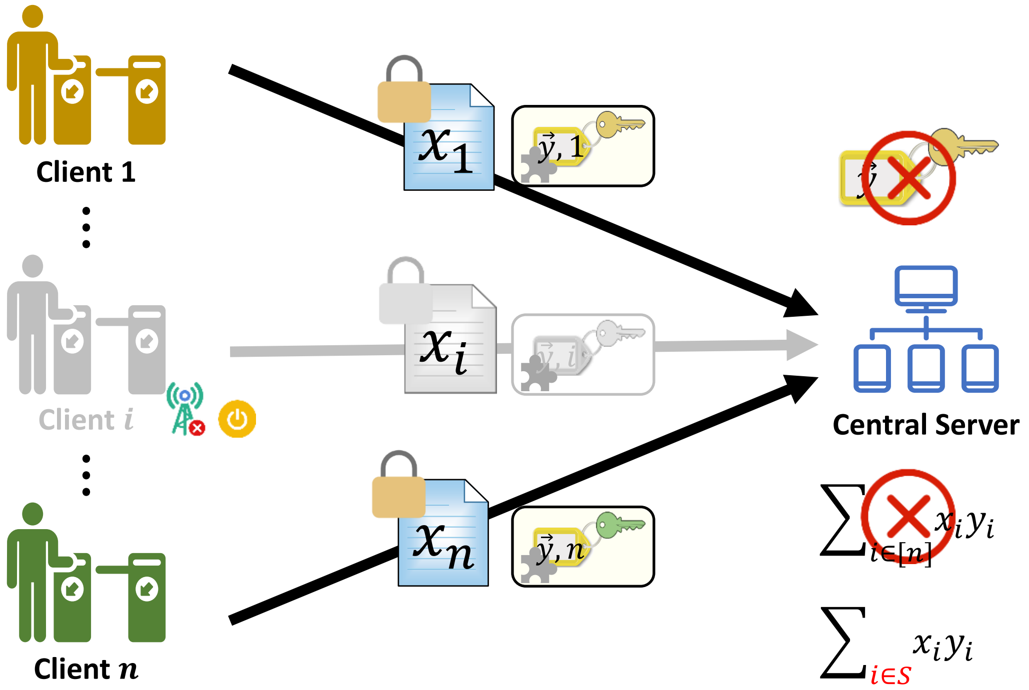
2. Motivation

- **Limitation:**
  - The central server must collect the partial functional keys and ciphertexts from all clients.
  - Some clients accidentally go offline due to hardware issues.
  - Some clients intentionally stop participating.



2. Motivation

- **Limitation:**
  - The central server must collect the partial functional keys and ciphertexts from all clients.
  - Some clients accidentally go offline due to hardware issues.
  - Some clients intentionally stop participating.



- **Question:**

Can DMCFE still work even when some clients do not generate partial functional keys for the function or encrypt their sensitive data?

Outline

1. **Introduction**
2. **Motivation**
3. **Definition (RDMCFE)**
4. **IP-RDMCFE Constructions**
5. **Conclusion**

3. Definition (RDMCFE)

- RDMCFE supports a more flexible function family $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$. A function $f \in \mathcal{F}_n$ is defined as

$$f: \mathcal{X}_1 \times \cdots \times \mathcal{X}_n \to \mathcal{Y}$$

where $\mathcal{X}_i$ $(i \in [n])$ contains a pre-defined default value x_0.

- An RDMCFE scheme for $\mathcal{F}$, the label set $\mathcal{L}$ and the threshold t is comprised of a setup protocol and three algorithms:
  - $(pp, \{sk_i\}_{i\in[n]}, \{ek_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n, t)$
  - $(i, Ct_{i,l}) \leftarrow Enc(ek_i, x_i, l)$
  - $(i, dk_{i,l}) \leftarrow PFunKG(sk_i, f, l, S)$
  - $f(x'_1, \cdots, x'_n) \leftarrow Dec(l, \{dk_{i,l}\}_{i\in S}, \{Ct_{i,l}\}_{i\in S})$

For $i \in [n]$, if $i \in S$, there is $x'_i = x_i$, otherwise $x'_i = x_0$.

3. Definition (RDMCFE)

DMCFE VS **RDMCFE**

DMCFE

- $(pp, \{ek_i\}_{i\in[n]}, \{sk_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n)$
- $Ct_{i,l} \leftarrow Enc(ek_i, x_i, l)$
- $dk_{f,i} \leftarrow KGShare(sk_i, f)$
- $sk_f \leftarrow KGComb(f, \{dk_{f,i}\}_{i\in[n]})$
- $f(x_1, \cdots, x_n) \leftarrow Dec(sk_f, \{Ct_{i,l}\}_{i\in[n]})$

RDMCFE

- $(pp, \{sk_i\}_{i\in[n]}, \{ek_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n, t)$
- $(i, Ct_{i,l}) \leftarrow Enc(ek_i, x_i, l)$
- $(i, dk_{i,l}) \leftarrow PFunKG(sk_i, f, l, S)$
- $f(x'_1, \cdots, x'_n) \leftarrow Dec(l, \{dk_{i,l}\}_{i\in S}, \{Ct_{i,l}\}_{i\in S})$

  (if $i \in S$, there is $x'_i = x_i$, otherwise $x'_i = x_0$)

- The threshold t limits the difference between the function values in the robust and non-robust settings.
- The label l achieves fine-grained access control, and S declares the state of each client so that the partial functional key can eliminate the influence of negative clients.
- Some inputs of the function $f(x'_1, \cdots, x'_n)$ may be default values x_0.

3. Definition (RDMCFE)

IND Security

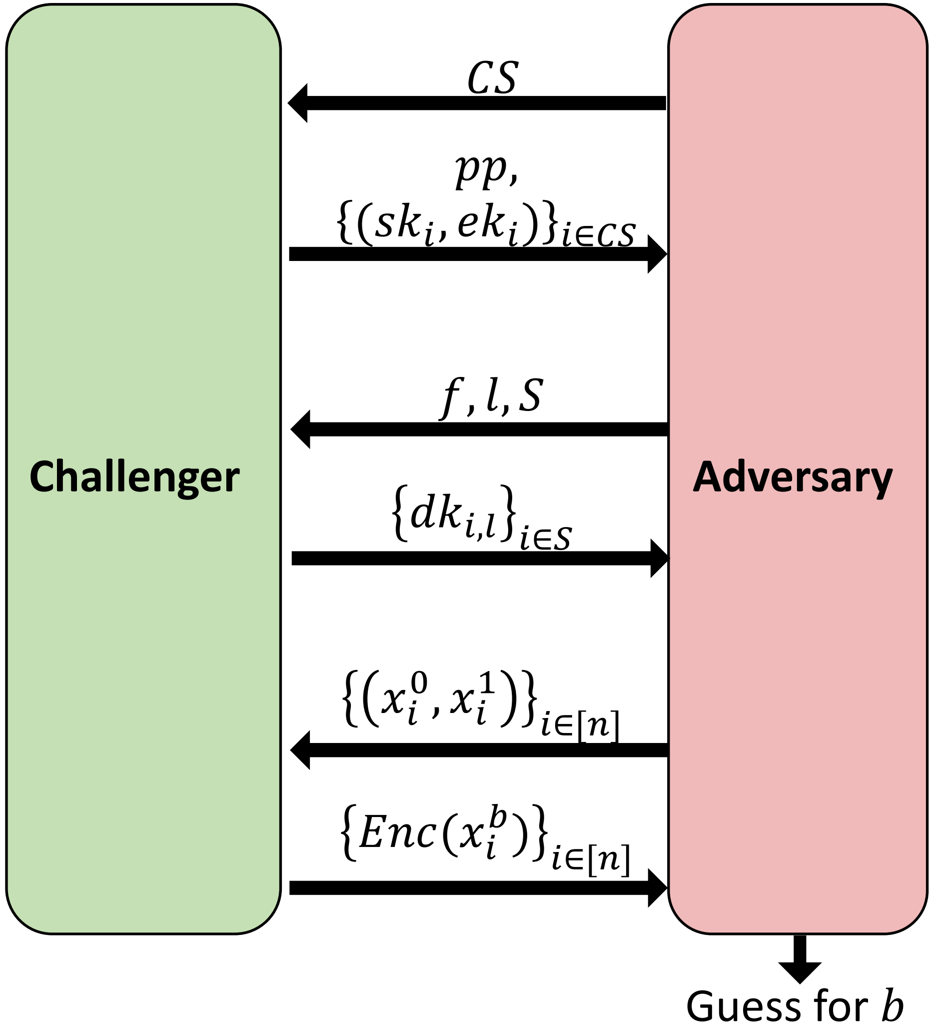
- **Corruption Query**

  $|CS| < t.$

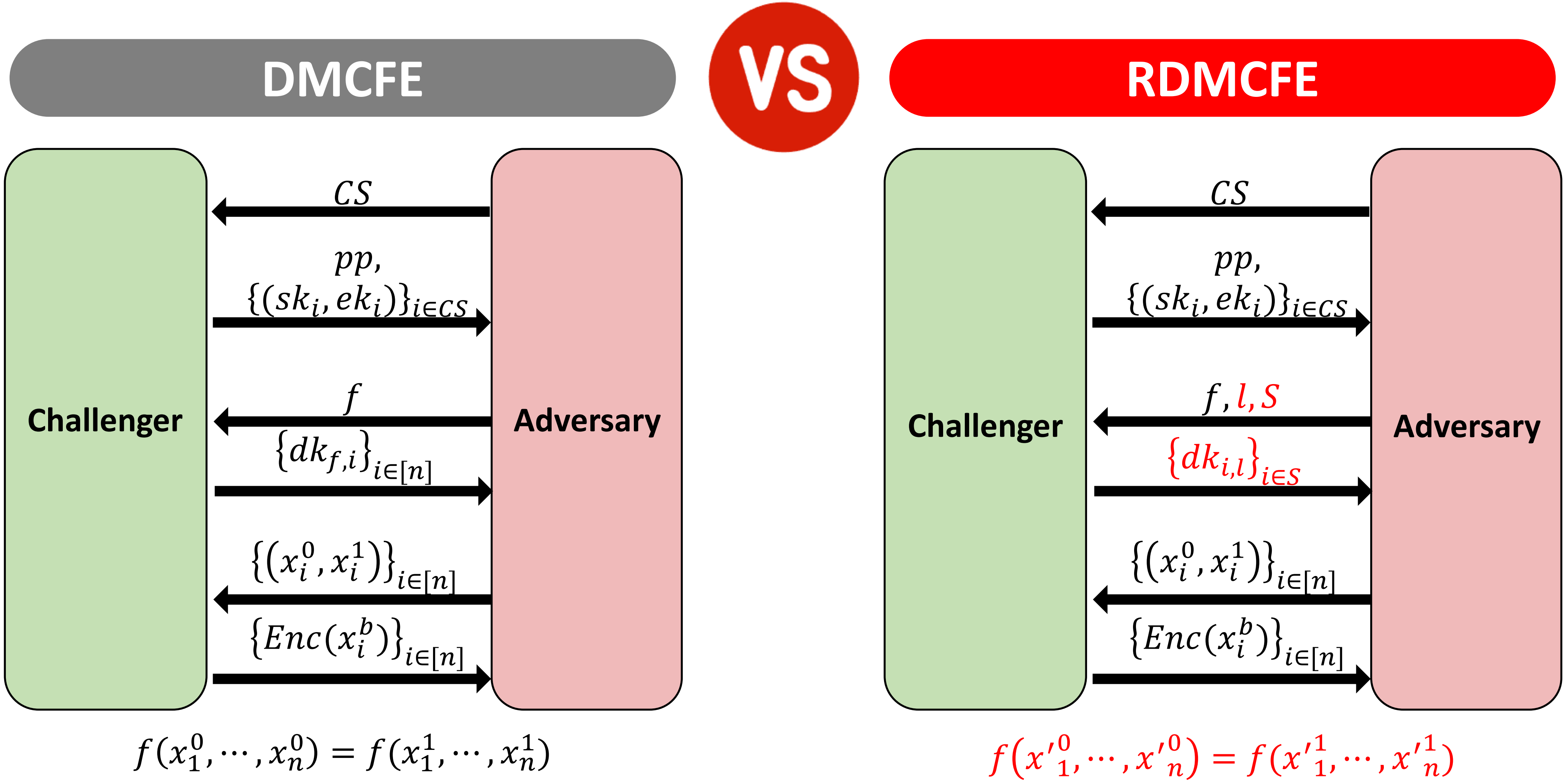
- **Key Generation Query**
  - $Q_y = 1$: One-IND Security
  - $Q_y > 1$: Many-IND Security

- **Encryption Query**
  - For $i \in CS$, there is $x_i^0 = x_i^1$.
  - $f({x'}_1^0, \cdots, {x'}_n^0) = f({x'}_1^1, \cdots, {x'}_n^1)$.

$\{Enc(x_i^0)\}_{i\in[n]} \approx_C \{Enc(x_i^1)\}_{i\in[n]}$



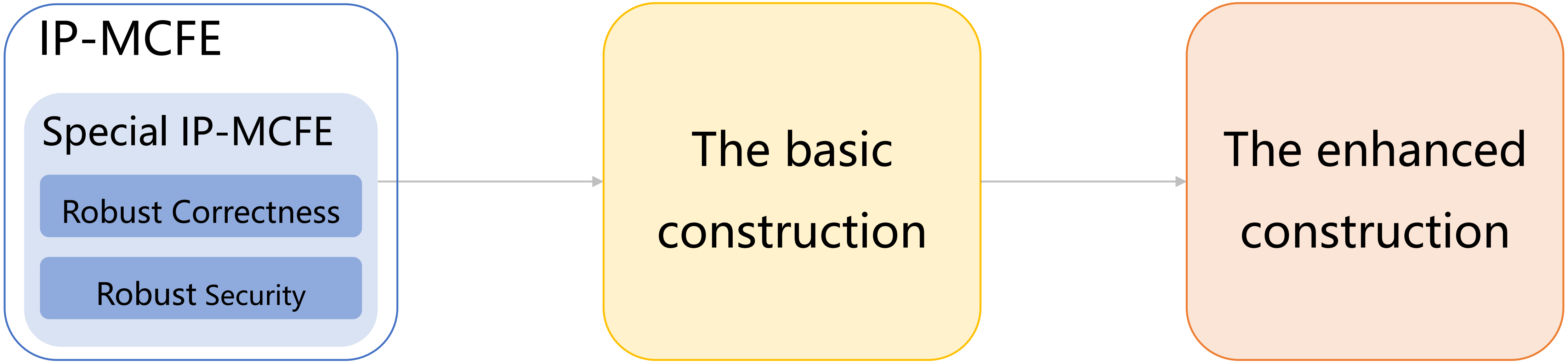
3. Definition (RDMCFE)



Outline

1. **Introduction**
2. **Motivation**
3. **Definition (RDMCFE)**
4. **IP-RDMCFE Constructions**
5. **Conclusion**

4. IP-RDMCFE Constructions

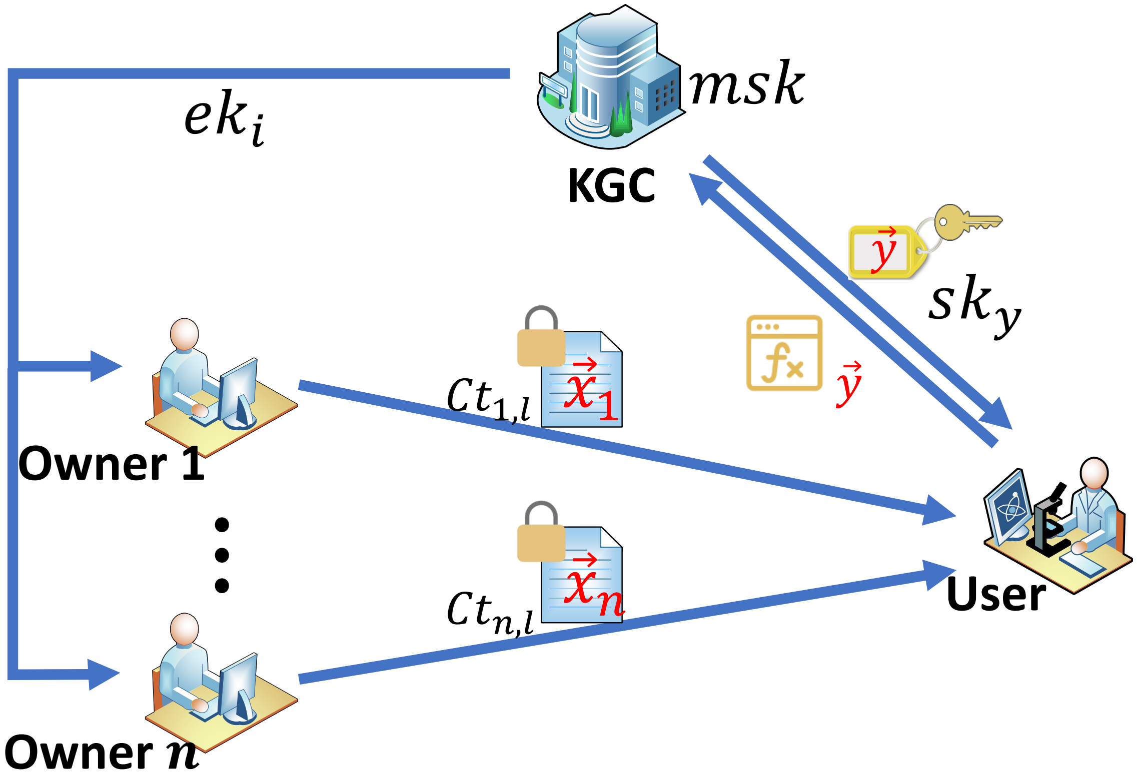


	The basic construction	The enhanced construction
Structure	Double-masking	Single-masking
Tools	Threshold secret sharing(SS), Non-Interactive Key Exchange (NIKE), Special IP-MCFE	Homomorphic SS, Inner Product FE (IP-FE), Special IP-ID-MCFE
Security	One-IND secure	Many-IND secure

4. IP-RDMCFE Constructions

Inner Product Multi-Client Functional Encryption (IP-MCFE) [AGRW 17]

- $(pp, msk, \{ek_i\}_{i\in[n]}) \leftarrow Setup(1^\lambda, \mathcal{F}_n)$
- $Ct_{i,l} \leftarrow Enc(ek_i, \vec{x}_i, l)$
- $sk_y \leftarrow KeyGen(msk, \vec{y} = (\vec{y}_1, \cdots, \vec{y}_n))$
- $\sum_{i\in[n]}\langle\vec{x}_i, \vec{y}_i\rangle \leftarrow Dec(sk_y, \{Ct_{i,l}\}_{i\in[n]})$



Data User can only learn $\sum_{i\in[n]}\langle\vec{x}_i, \vec{y}_i\rangle$ and nothing else about $\vec{x}_1, \cdots, \vec{x}_n$.

4. IP-RDMCFE Constructions

Special IP-MCFE [ABKW 19]

- An IP-MCFE scheme has the special key generation property, if:
  - $msk = \{ek_i\}_{i\in[n]}$ and $ek_i = (s_i, \vec{u}_i)$.
  - $sk_y = (\{s_{i,y}\}_{i\in[n]}, dk_y)$ and $dk_y = \sum_{i\in[n]}\langle\vec{u}_i, \vec{y}_i\rangle$.

- Robust Correctness：

  $Dec\left(sk'_y, \{Ct_{i,l}\}_{i\in S}\right) = \sum_{i\in S}\langle\vec{x}_i, \vec{y}_i\rangle$, where $sk'_y = (\{s_{i,y}\}_{i\in S}, \sum_{i\in S}\langle\vec{u}_i, \vec{y}_i\rangle)$.

- Robust Security:

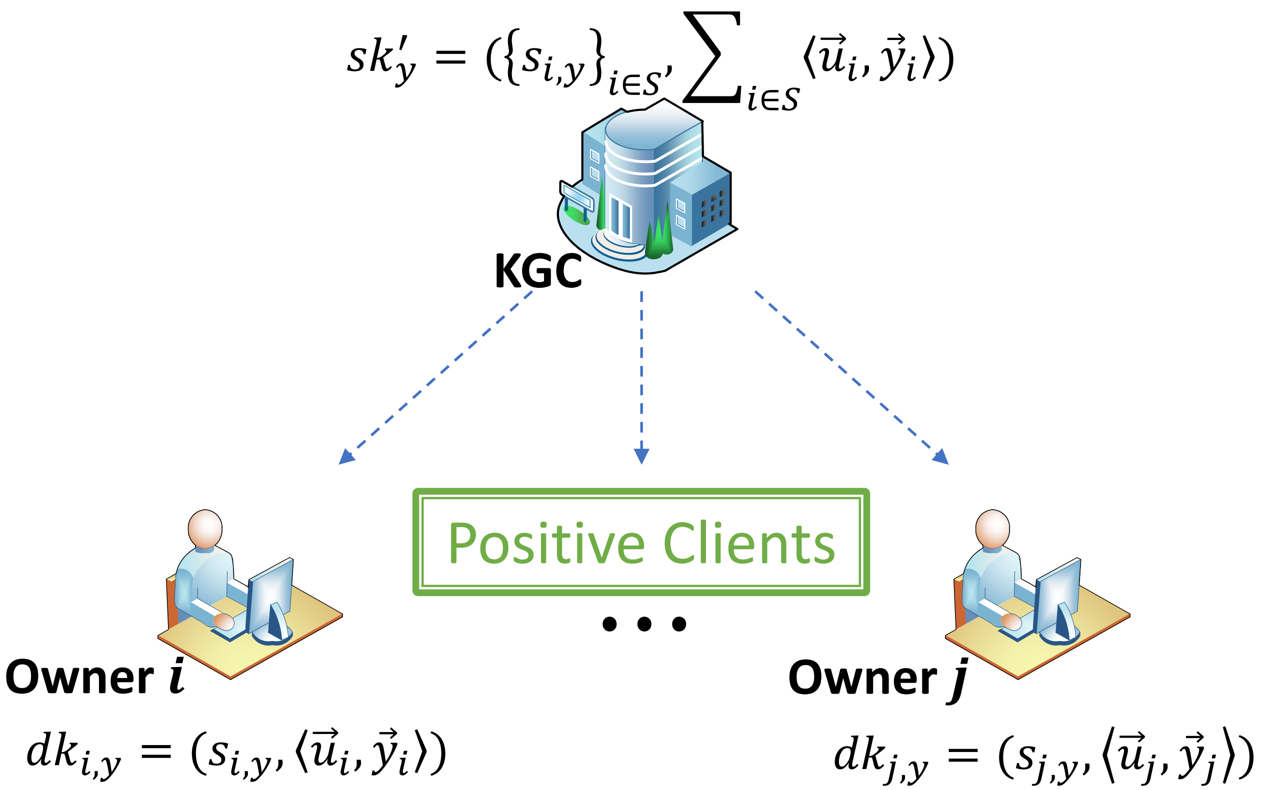
  On input $\{\vec{y}_i\}_{i\in S}$, the oracle $QKeyGen(\cdot)$ outputs $sk'_y \leftarrow KeyGen(\{ek_i\}_{i\in S}, \{\vec{y}_i\}_{i\in S})$.

  Lemma 1. If the Special IP-MCFE has sta-IND security, it also has robust security.

NEW

4. IP-RDMCFE Constructions

Naïve Manner for Decentralization



Problem:

The partial functional keys can be combined arbitrarily, *i.e.*, it suffers the mix-and-match attack.

The decryptor must be limited to receiving not each individual $\langle \vec{u}_i, \vec{y}_i \rangle$ but only the aggregated $\sum_{i \in S} \langle \vec{u}_i, \vec{y}_i \rangle$.

4. IP-RDMCFE Constructions

Basic Construction

$$mk_i = \langle \vec{u}_i, \vec{y}_i \rangle + r_i + w_i$$

$$w_i = \sum_{j \in [n], i > j} v_{i,j} - \sum_{j \in [n], i < j} v_{i,j}$$

$$\sum_{i \in S} \langle \vec{u}_i, \vec{y}_i \rangle = \sum_{i \in S} mk_i - \sum_{i \in S} r_i - \sum_{i \in S} w_i$$

Non-Interactive Key Exchange (NIKE) scheme:

- $pp \leftarrow NIKE.Setup(1^{\lambda})$
- $(nsk_i, npk_i) \leftarrow NIKE.KeyGen(pp)$
- $v_{i,j} \leftarrow NIKE.Agree(nsk_i, npk_j)$

$$0 = \sum_{i \in [n]} w_i$$

4. IP-RDMCFE Constructions

Basic Construction

$$mk_i = \langle \vec{u}_i, \vec{y}_i \rangle + r_i + w_i$$

$$w_i = \sum_{j \in [n], i > j} v_{i,j} - \sum_{j \in [n], i < j} v_{i,j}$$

Non-Interactive Key Exchange (NIKE) scheme:

- $pp \leftarrow NIKE.Setup(1^\lambda)$
- $(nsk_i, npk_i) \leftarrow NIKE.KeyGen(pp)$
- $v_{i,j} \leftarrow NIKE.Agree(nsk_i, npk_j)$

$$\sum_{i \in S} \langle \vec{u}_i, \vec{y}_i \rangle = \sum_{i \in S} mk_i - \sum_{i \in S} r_i + \sum_{i \in [n] \setminus S} w_i$$

$$\sum_{i \in S} w_i + \sum_{i \in [n] \setminus S} w_i = 0 = \sum_{i \in [n]} w_i$$

4. IP-RDMCFE Constructions

Basic Construction

$$mk_i = \langle \vec{u}_i, \vec{y}_i \rangle + r_i + w_i$$

$$w_i = \sum_{j\in[n], i>j} v_{i,j} - \sum_{j\in[n], i<j} v_{i,j}$$

Non-Interactive Key Exchange (NIKE) scheme:

- $pp \leftarrow NIKE.Setup(1^\lambda)$
- $(nsk_i, npk_i) \leftarrow NIKE.KeyGen(pp)$
- $v_{i,j} \leftarrow NIKE.Agree(nsk_i, npk_j)$

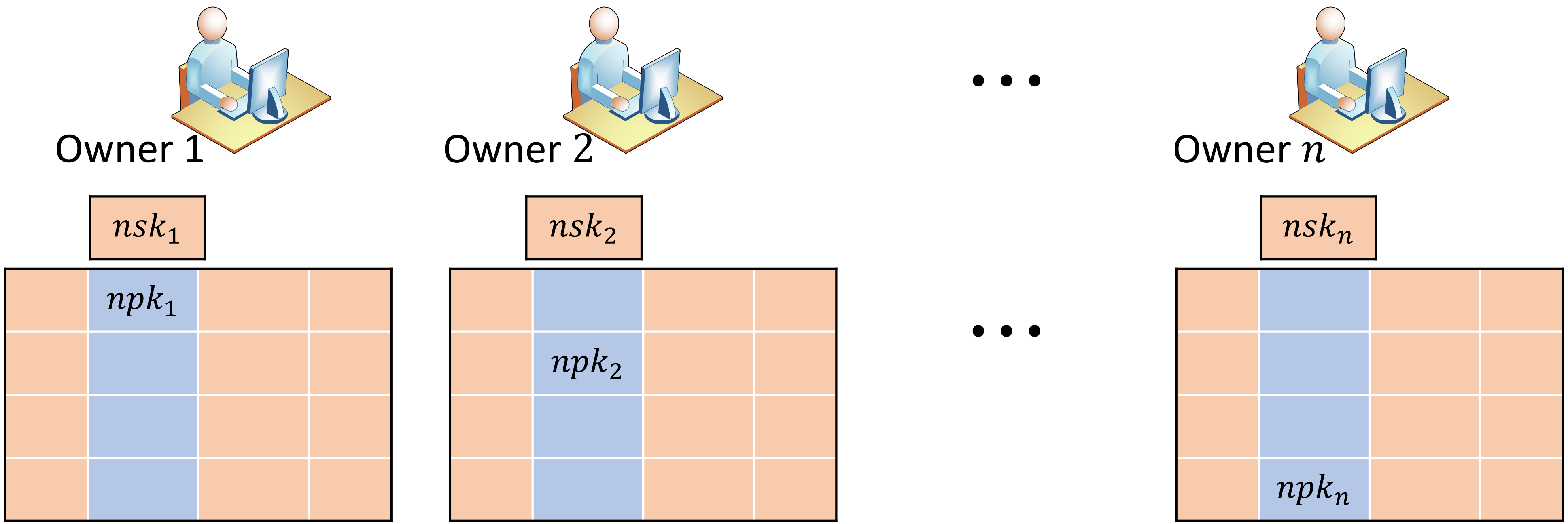
$$\sum_{i\in S}\langle \vec{u}_i, \vec{y}_i \rangle = \sum_{i\in S} mk_i - \sum_{i\in S} r_i + \sum_{i\in[n]\backslash S} w_i \quad \Leftarrow \quad \sum_{i\in S} w_i + \sum_{i\in[n]\backslash S} w_i = 0 = \sum_{i\in[n]} w_i$$

- ☐ For each owner, the user can only obtain one mask, while another mask can still protect $\langle \vec{u}_i, \vec{y}_i \rangle$.
- ☐ For robustness, r_i and nsk_i are shared with other owners through a (t, n) secret sharing (SS) scheme.
- ☐ When $|S| \geq t$, even if some owners are negative, the user can still obtain shares from positive owners to reconstruct $\{r_i\}_{i\in S}$ and $\{nsk_i\}_{i\in[n]\backslash S}$.

4. IP-RDMCFE Constructions

① Generate and share the masks r_i and w_i

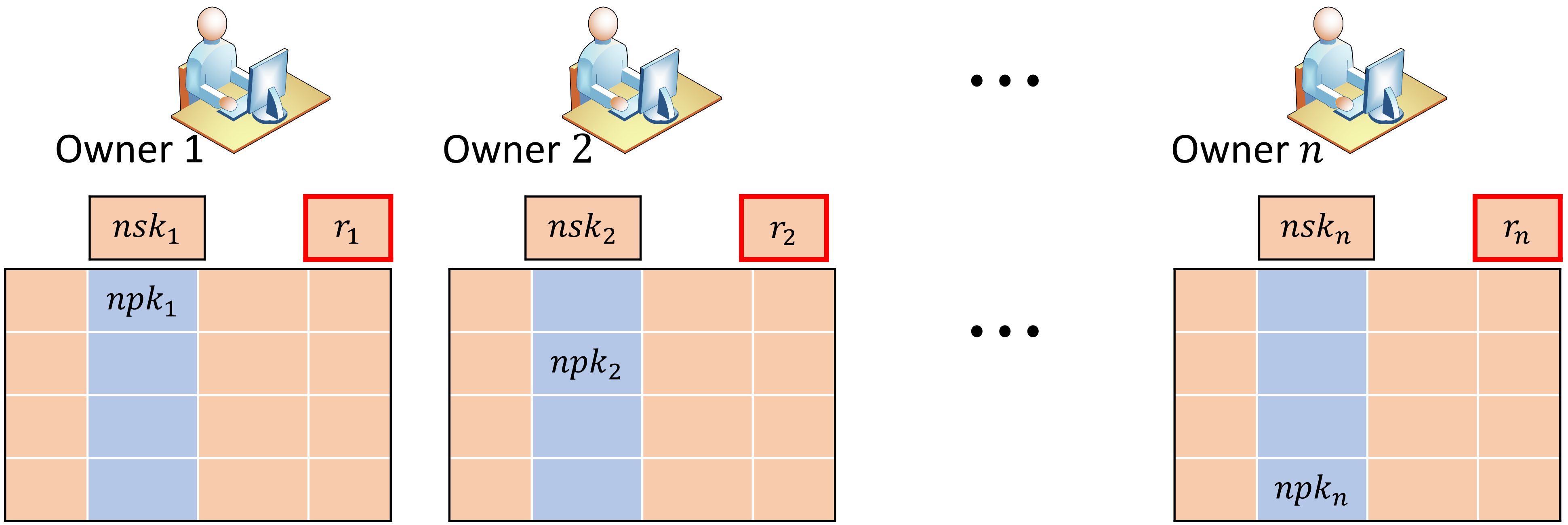
- $(nsk_i, npk_i) \leftarrow NIKE.KeyGen(pp_2)$



4. IP-RDMCFE Constructions

① Generate and share the masks r_i and w_i

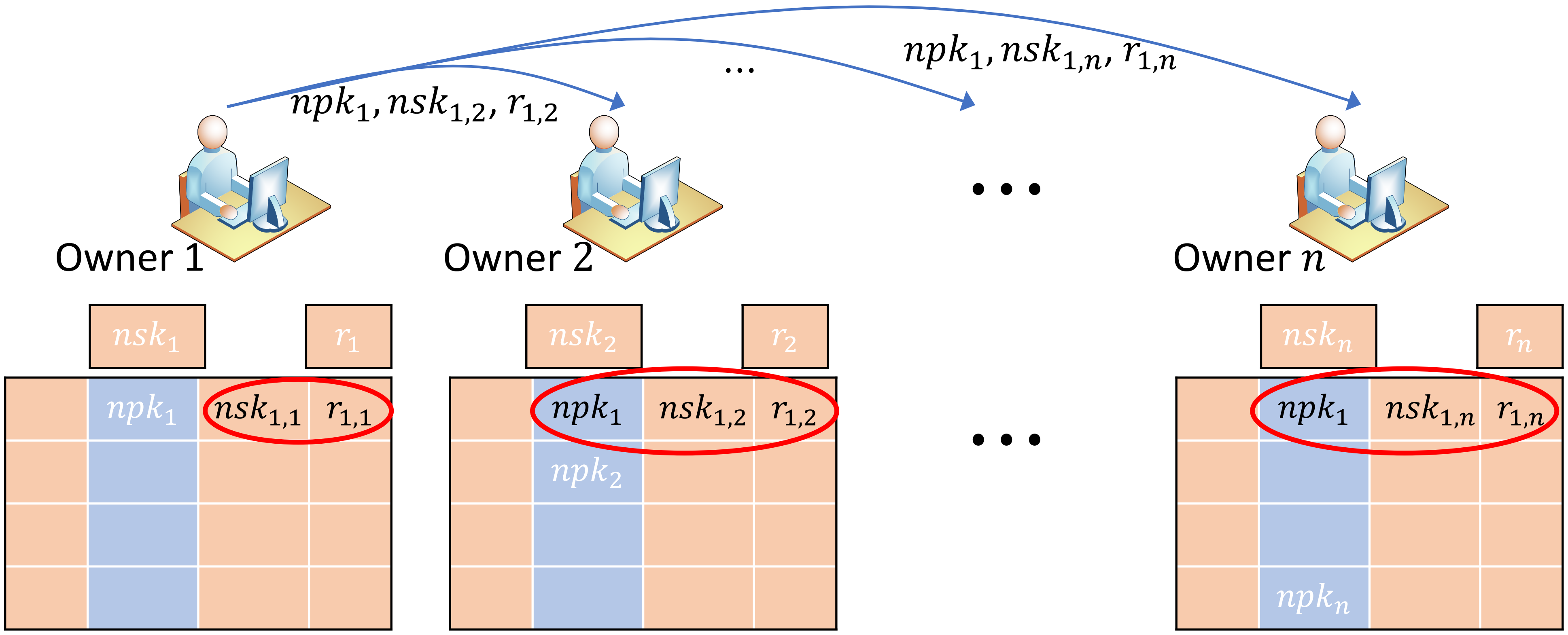
- $(nsk_i, npk_i) \leftarrow NIKE.KeyGen(pp_2)$
- $r_i \leftarrow \mathbb{Z}_L$



4. IP-RDMCFE Constructions

① Generate and share the masks r_i and w_i

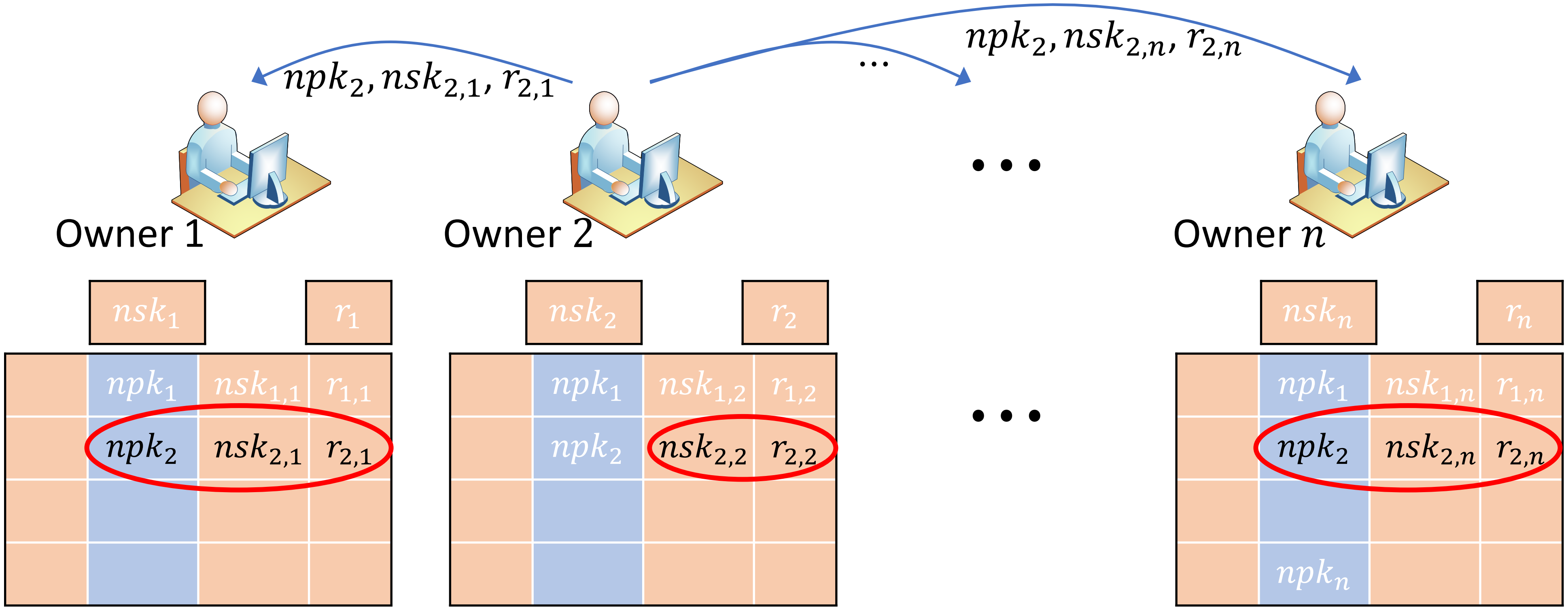
- $\{nsk_{1,i}\}_{i\in[n]} \leftarrow SS.Share(nsk_1, t, [n])$
- $\{r_{1,i}\}_{i\in[n]} \leftarrow SS.Share(r_1, t, [n])$



4. IP-RDMCFE Constructions

① Generate and share the masks r_i and w_i

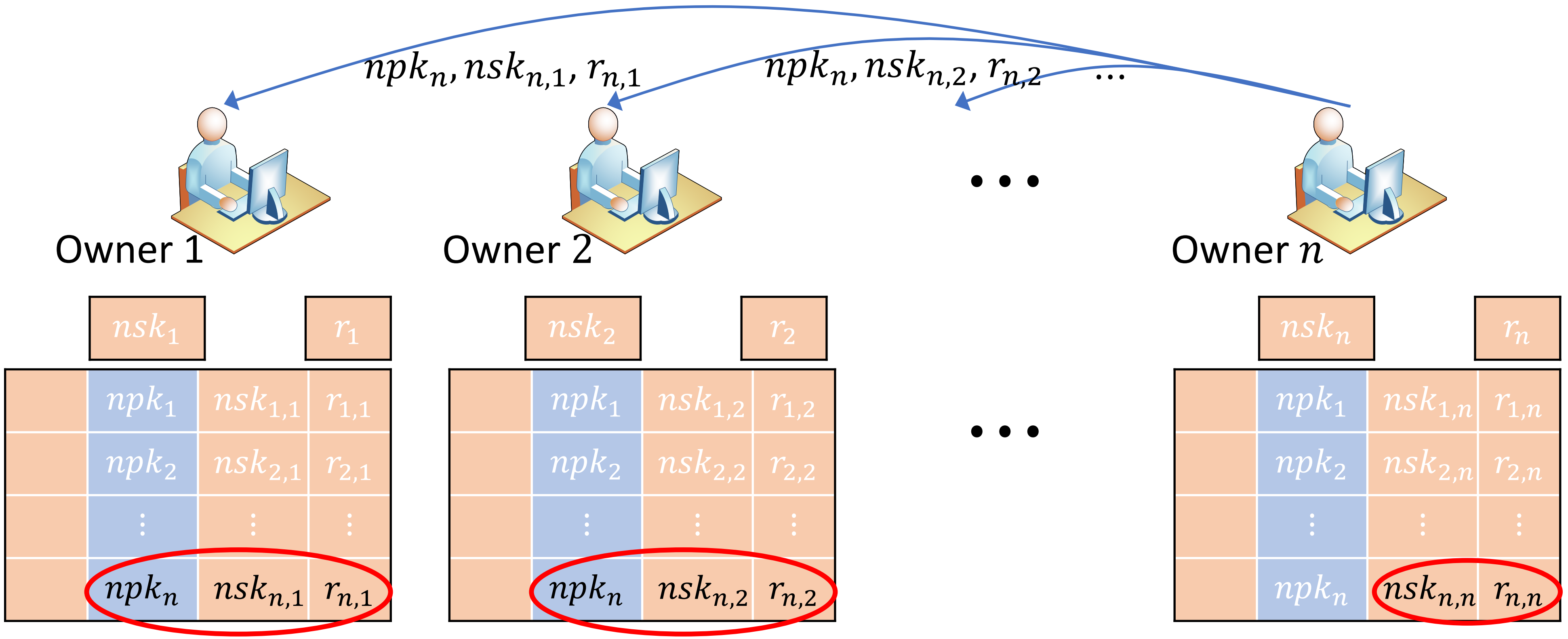
- $\{nsk_{2,i}\}_{i\in[n]} \leftarrow SS.Share(nsk_2, t, [n])$
- $\{r_{2,i}\}_{i\in[n]} \leftarrow SS.Share(r_2, t, [n])$



4. IP-RDMCFE Constructions

① Generate and share the masks r_i and w_i

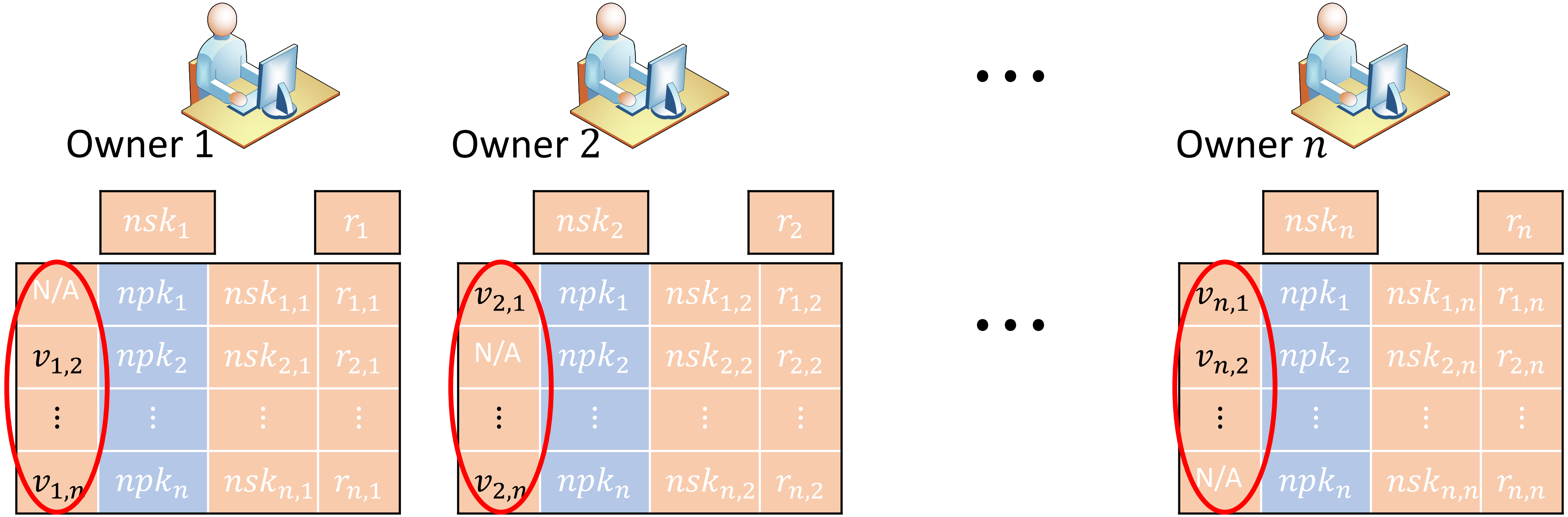
- $\{nsk_{n,i}\}_{i\in[n]} \leftarrow SS.Share(nsk_n, t, [n])$
- $\{r_{n,i}\}_{i\in[n]} \leftarrow SS.Share(r_n, t, [n])$



4. IP-RDMCFE Constructions

① Generate and share the masks r_i and w_i

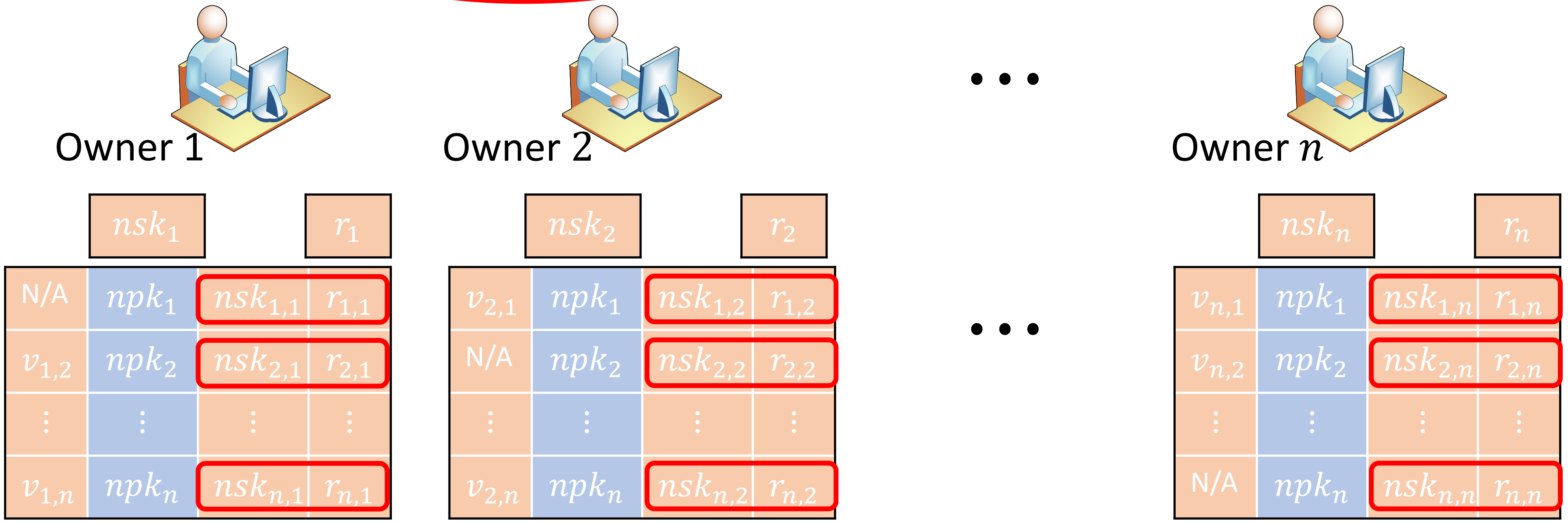
- $v_{i,j} \leftarrow NIKE.Agree(nsk_i, npk_j)$



4. IP-RDMCFE Constructions

② Help the user to reconstruct $\{r_i\}_{i\in S}$ and $\{nsk_i\}_{i\in[n]\backslash S}$

- $\left(s_{i,y}, \langle \vec{u}_i, \vec{y}_i \rangle\right) \leftarrow MCFE.KeyGen(mek_i, \vec{y}_i)$.
- $mk_i = \langle \vec{u}_i, \vec{y}_i \rangle + w_i + r_i$.
- $\boldsymbol{dk_{i,y}} := \left(s_{i,y}, mk_i, \{nsk_{j,i}\}_{j\in[n]\backslash S}, \{r_{j,i}\}_{j\in S}\right)$.



4. IP-RDMCFE Constructions

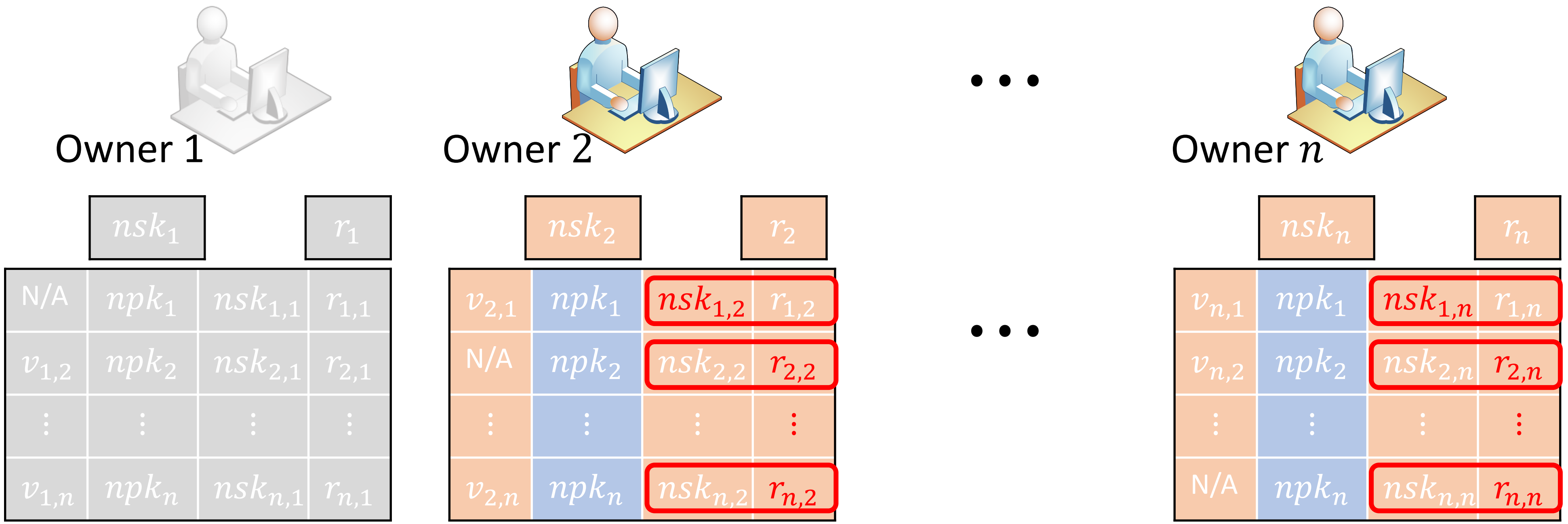
② Help the user to reconstruct $\{r_i\}_{i\in S}$ and $\{nsk_i\}_{i\in[n]\backslash S}$

eg.

$[n]\backslash S = \{1\}$

$S = \{2, \dots, n\}$

$\boldsymbol{dk_{2,y}} := (s_{2,y}, mk_2, nsk_{1,2}, \{r_{j,2}\}_{j\in\{2,\dots,n\}})$ $\boldsymbol{dk_{n,y}} := (s_{n,y}, mk_n, nsk_{1,n}, \{r_{j,n}\}_{j\in\{2,\dots,n\}})$



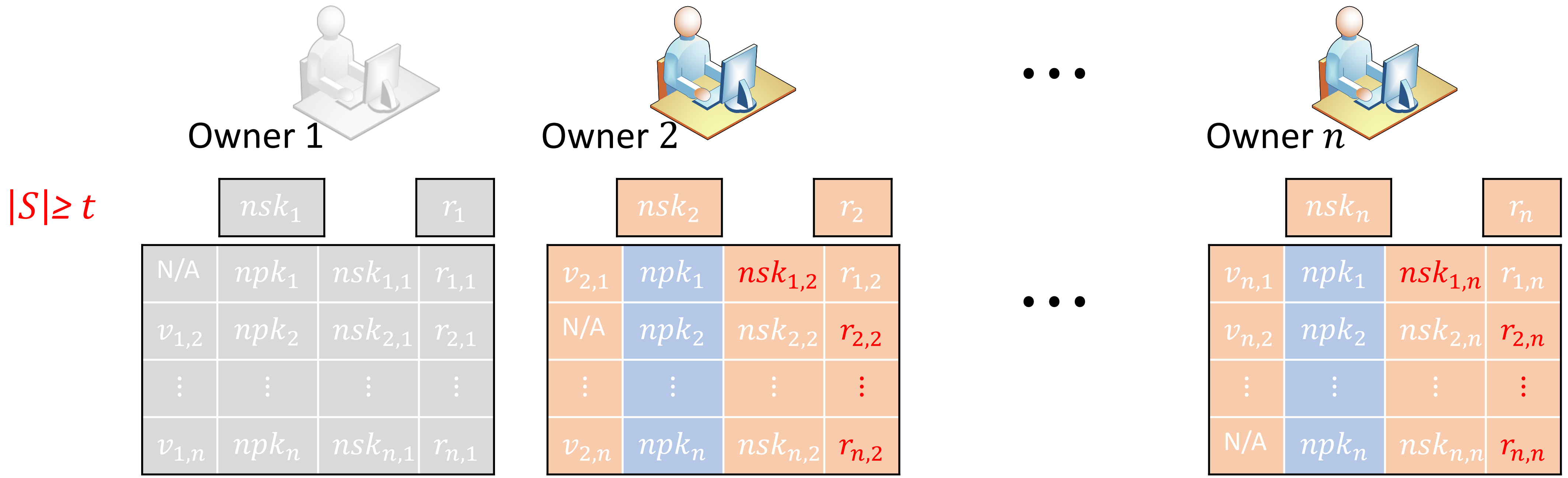
4. IP-RDMCFE Constructions

② Help the user to reconstruct $\{r_i\}_{i\in S}$ and $\{nsk_i\}_{i\in[n]\setminus S}$

eg.

$[n]\setminus S = \{1\}$

$S = \{2, \ldots, n\}$



4. IP-RDMCFE Constructions

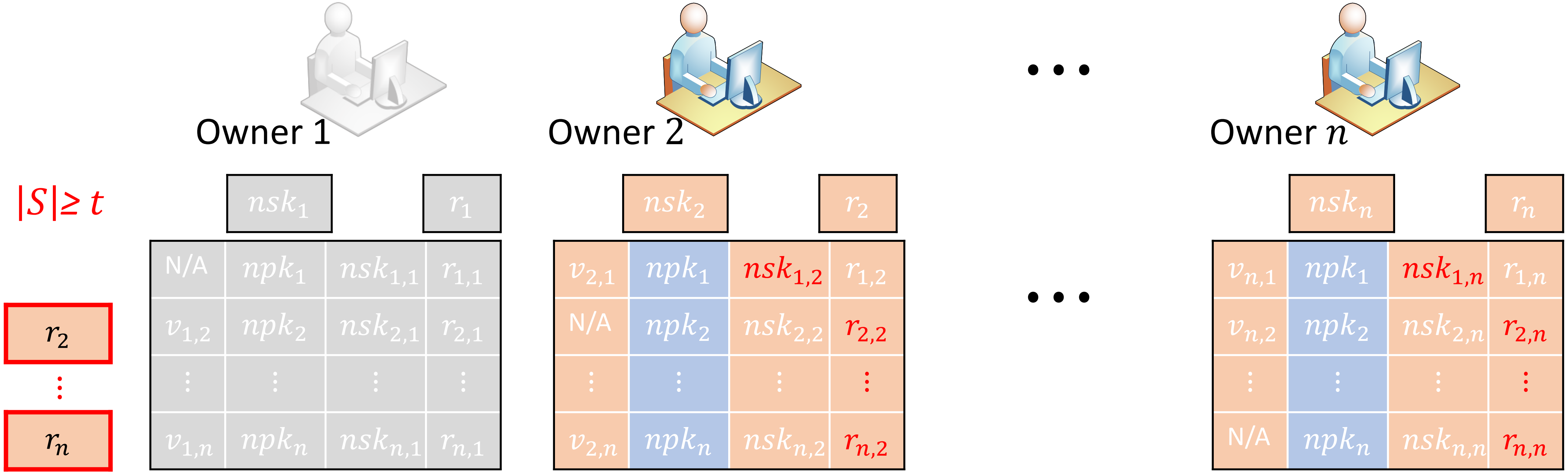
② Help the user to reconstruct $\{r_i\}_{i \in S}$ and $\{nsk_i\}_{i \in [n] \setminus S}$

eg.

a. For $j \in S$, $r_j \leftarrow SS.Recon(\{r_{j,i}\}_{i \in S}, t)$.

$[n] \setminus S = \{1\}$

$S = \{2, \ldots, n\}$



4. IP-RDMCFE Constructions

② Help the user to reconstruct $\{r_i\}_{i\in S}$ and $\{nsk_i\}_{i\in[n]\backslash S}$

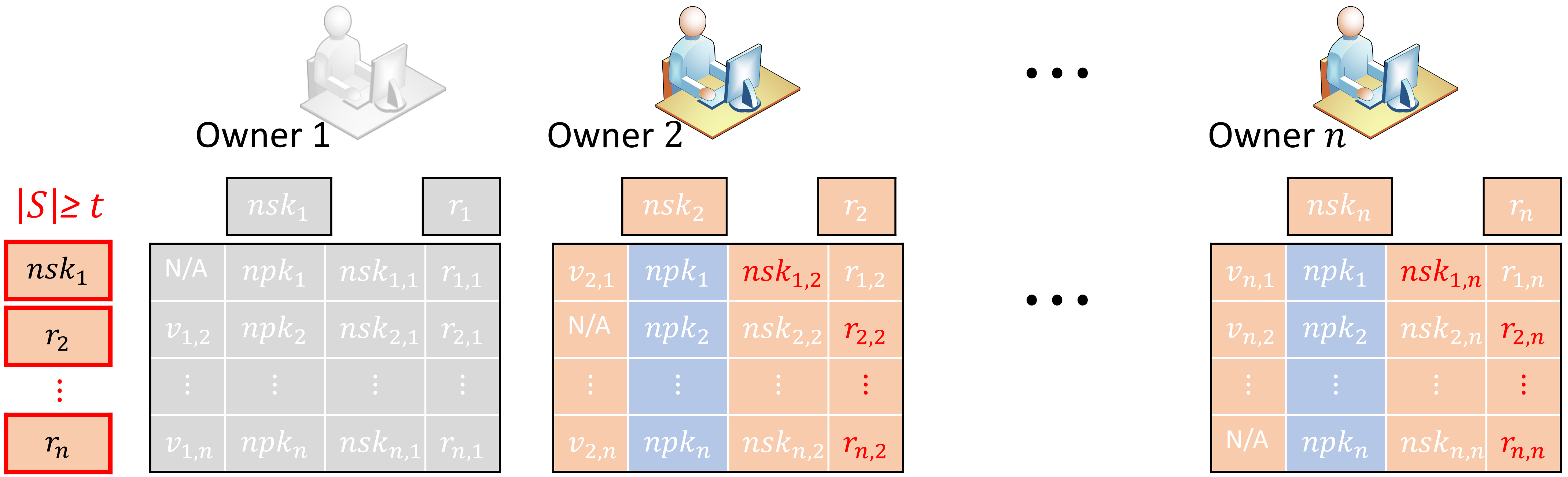
eg.

$[n]\backslash S = \{1\}$

$S = \{2, \ldots, n\}$

a. For $j \in S$, $r_j \leftarrow SS.Recon(\{r_{j,i}\}_{i\in S'}, t)$.

b. For $j \in [n]\backslash S$, $nsk_j \leftarrow SS.Recon(\{nsk_{j,i}\}_{i\in S'}, t)$,



4. IP-RDMCFE Constructions

② Help the user to reconstruct $\{r_i\}_{i \in S}$ and $\{nsk_i\}_{i \in [n] \setminus S}$

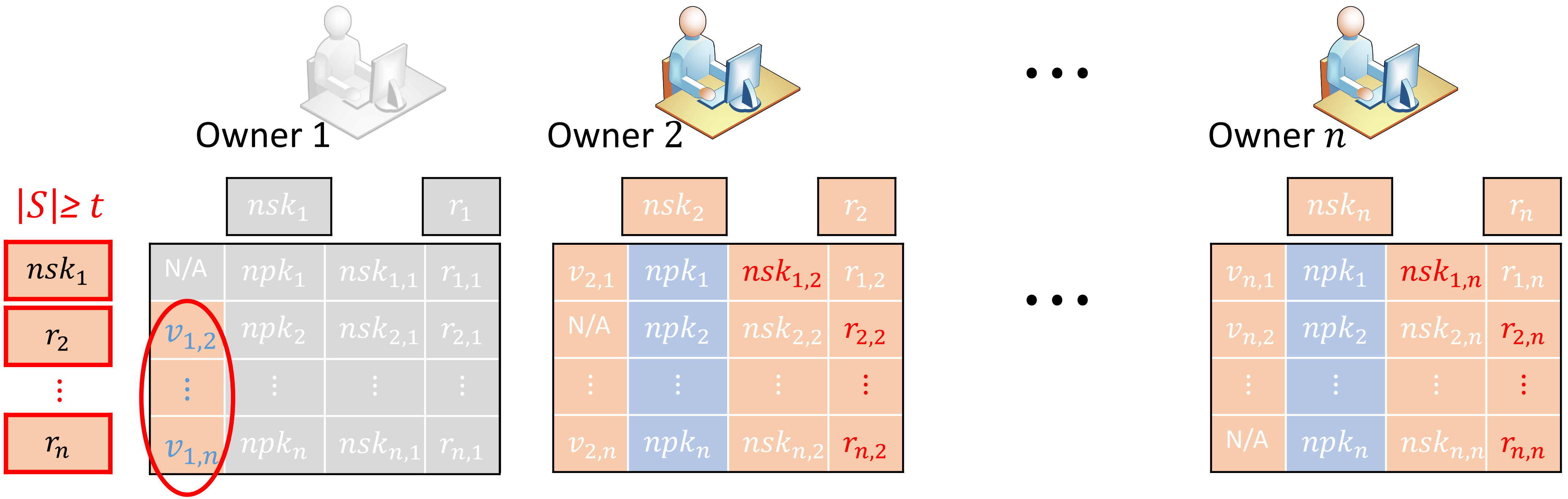
eg.

$[n] \setminus S = \{1\}$

$S = \{2, \dots, n\}$

a. For $j \in S$, $r_j \leftarrow SS.Recon(\{r_{j,i}\}_{i \in S}, t)$.

b. For $j \in [n] \setminus S$, $nsk_j \leftarrow SS.Recon(\{nsk_{j,i}\}_{i \in S}, t)$,
generate $v_{j,i} \leftarrow NIKE.Agree(nsk_j, npk_i)$ for $i \in [n]$.



4. IP-RDMCFE Constructions

② Help the user to reconstruct $\{r_i\}_{i \in S}$ and $\{nsk_i\}_{i \in [n] \backslash S}$

eg.

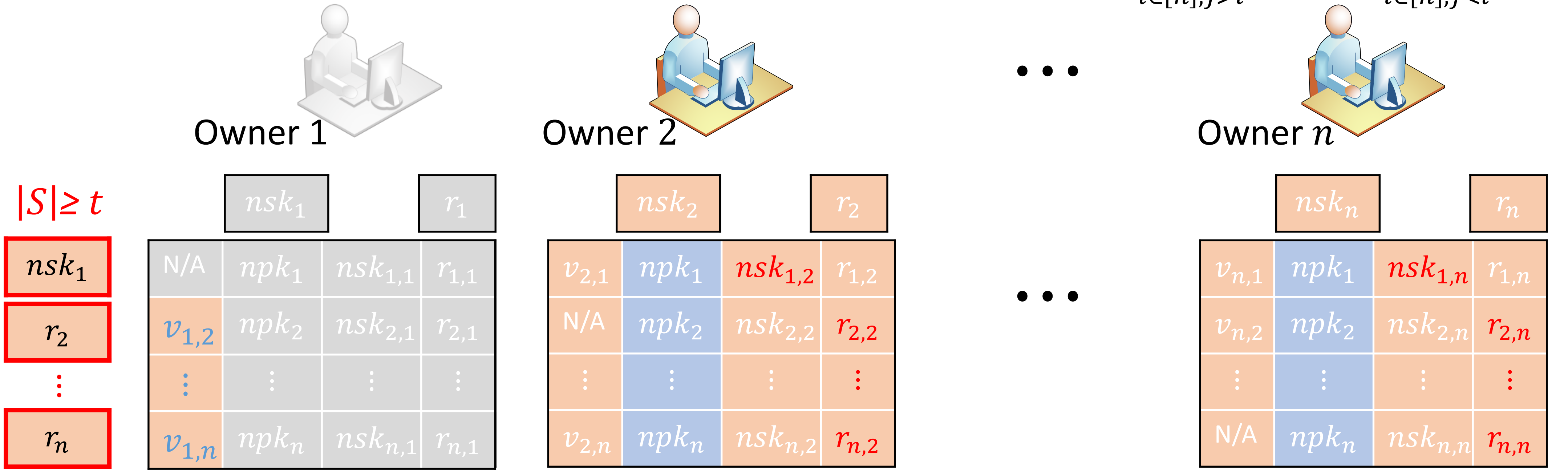
$[n] \backslash S = \{1\}$

$S = \{2, \dots, n\}$

a. For $j \in S$, $r_j \leftarrow SS.Recon(\{r_{j,i}\}_{i \in S}, t)$.

b. For $j \in [n] \backslash S$, $nsk_j \leftarrow SS.Recon(\{nsk_{j,i}\}_{i \in S}, t)$, generate $v_{j,i} \leftarrow NIKE.Agree(nsk_j, npk_i)$ for $i \in [n]$.

c. For $j \in [n] \backslash S$, $w_j = \sum_{i \in [n], j > i} v_{j,i} - \sum_{i \in [n], j < i} v_{j,i}$.



4. IP-RDMCFE Constructions

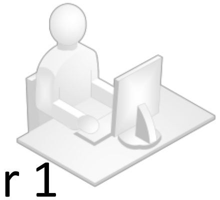
② Help the user to reconstruct $\{r_i\}_{i\in S}$ and $\{nsk_i\}_{i\in[n]\backslash S}$

$$dk_y = \sum_{i\in S} mk_i - \sum_{i\in S} r_i + \sum_{i\in[n]\backslash S} w_i$$
$$= \sum_{i\in S} \langle \vec{u}_i, \vec{y}_i \rangle$$

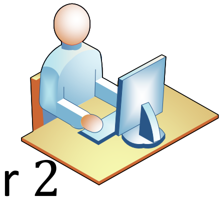
a. For $j \in S$, $r_j \leftarrow SS.Recon(\{r_{j,i}\}_{i\in S}, t)$.

b. For $j \in [n]\backslash S$, $nsk_j \leftarrow SS.Recon(\{nsk_{j,i}\}_{i\in S}, t)$, generate $v_{j,i} \leftarrow NIKE.Agree(nsk_j, npk_i)$ for $i \in [n]$.

c. For $j \in [n]\backslash S$, $w_j = \sum_{i\in[n], j>i} v_{j,i} - \sum_{i\in[n], j<i} v_{j,i}$.

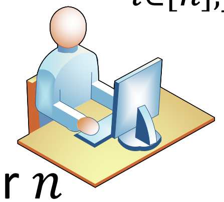


Owner 1



Owner 2

…



Owner n

$|S| \geq t$

nsk_1
r_2
⋮
r_n

Owner 1: nsk_1, r_1

N/A	npk_1	$nsk_{1,1}$	$r_{1,1}$
$v_{1,2}$	npk_2	$nsk_{2,1}$	$r_{2,1}$
⋮	⋮	⋮	⋮
$v_{1,n}$	npk_n	$nsk_{n,1}$	$r_{n,1}$

Owner 2: nsk_2, r_2

$v_{2,1}$	npk_1	$nsk_{1,2}$	$r_{1,2}$
N/A	npk_2	$nsk_{2,2}$	$r_{2,2}$
⋮	⋮	⋮	⋮
$v_{2,n}$	npk_n	$nsk_{n,2}$	$r_{n,2}$

…

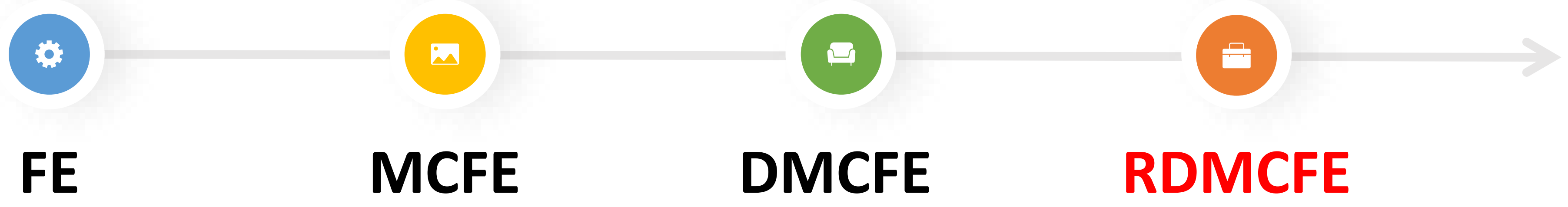
Owner n: nsk_n, r_n

$v_{n,1}$	npk_1	$nsk_{1,n}$	$r_{1,n}$
$v_{n,2}$	npk_2	$nsk_{2,n}$	$r_{2,n}$
⋮	⋮	⋮	⋮
N/A	npk_n	$nsk_{n,n}$	$r_{n,n}$

Outline

1. **Introduction**
2. **Motivation**
3. **Definition (RDMCFE)**
4. **IP-RDMCFE Constructions**
5. **Conclusion**

6. Conclusion



☐ New notion

- ➢Robust Decentralized Multi-Client Functional Encryption

☐ New properties for Special IP-MCFE

- ➢Robust Correctness
- ➢Robust Security

☐ Constructions

- ➢The basic IP-RDMCFE construction
- ➢The enhanced IP-RDMCFE construction

Thanks for your attention!