Dynamic Decentralized Functional Encryption

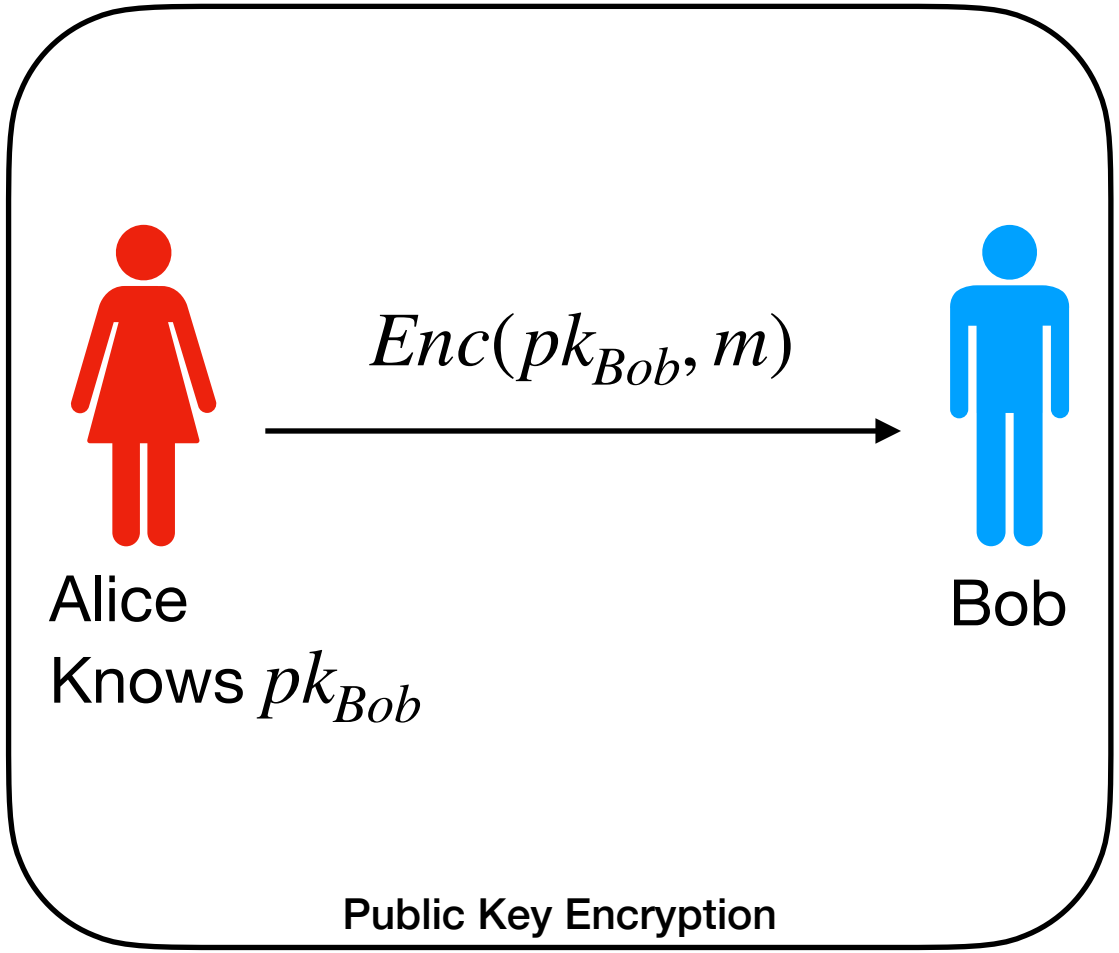
An introduction

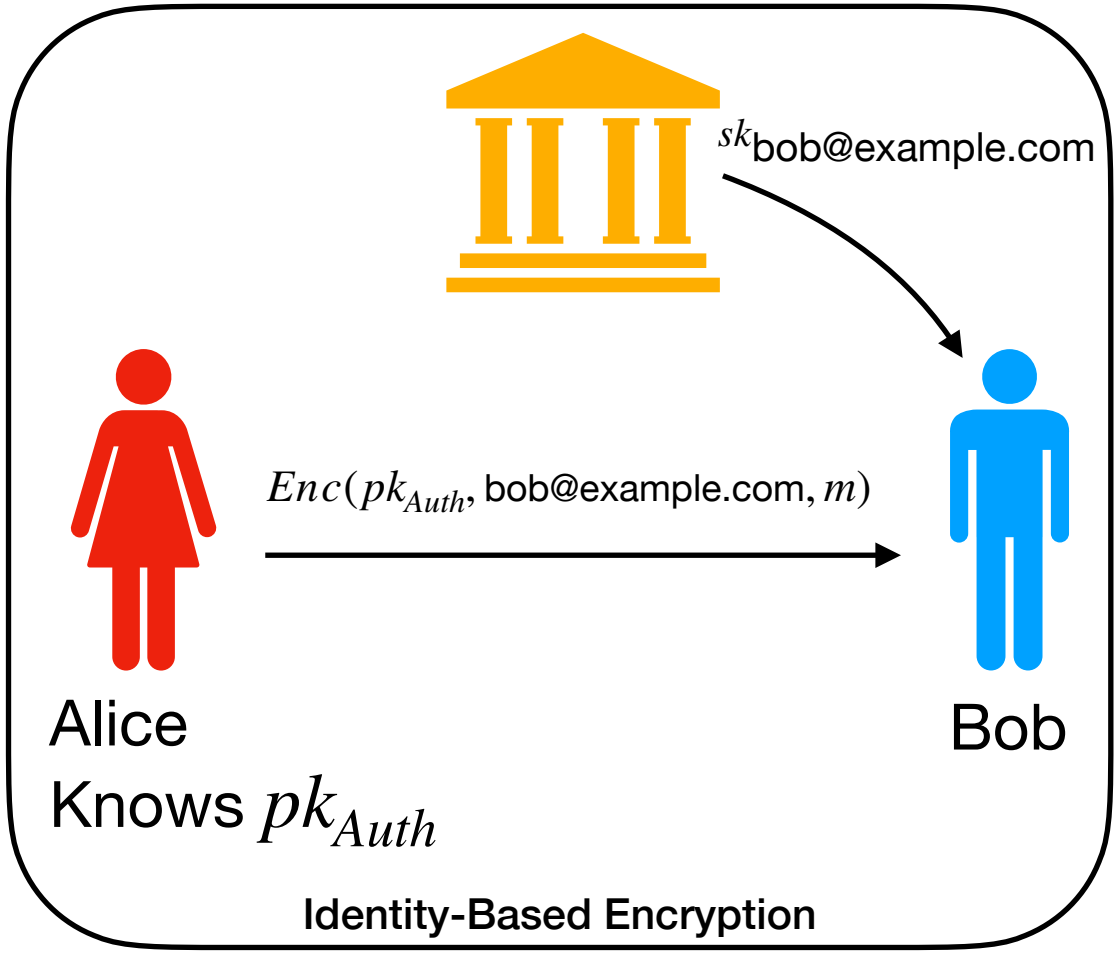
Edouard Dufour - November 2026
Based on joint work with Jérémy Chotard, Romain Gay, Duong Hieu Phan, and David Pointcheval.

Background
What is Functional Encryption?

Historical perspective on Functional Encryption

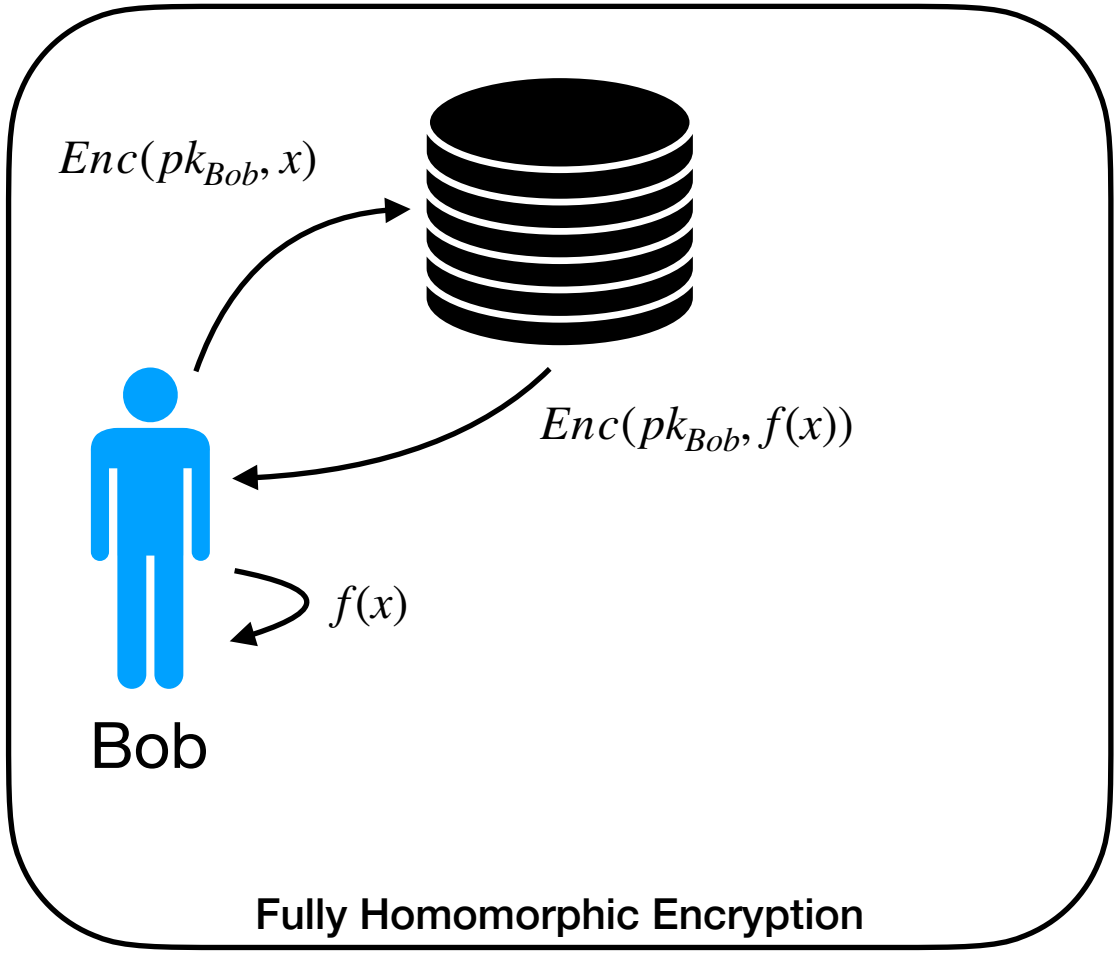
From PKE to FE

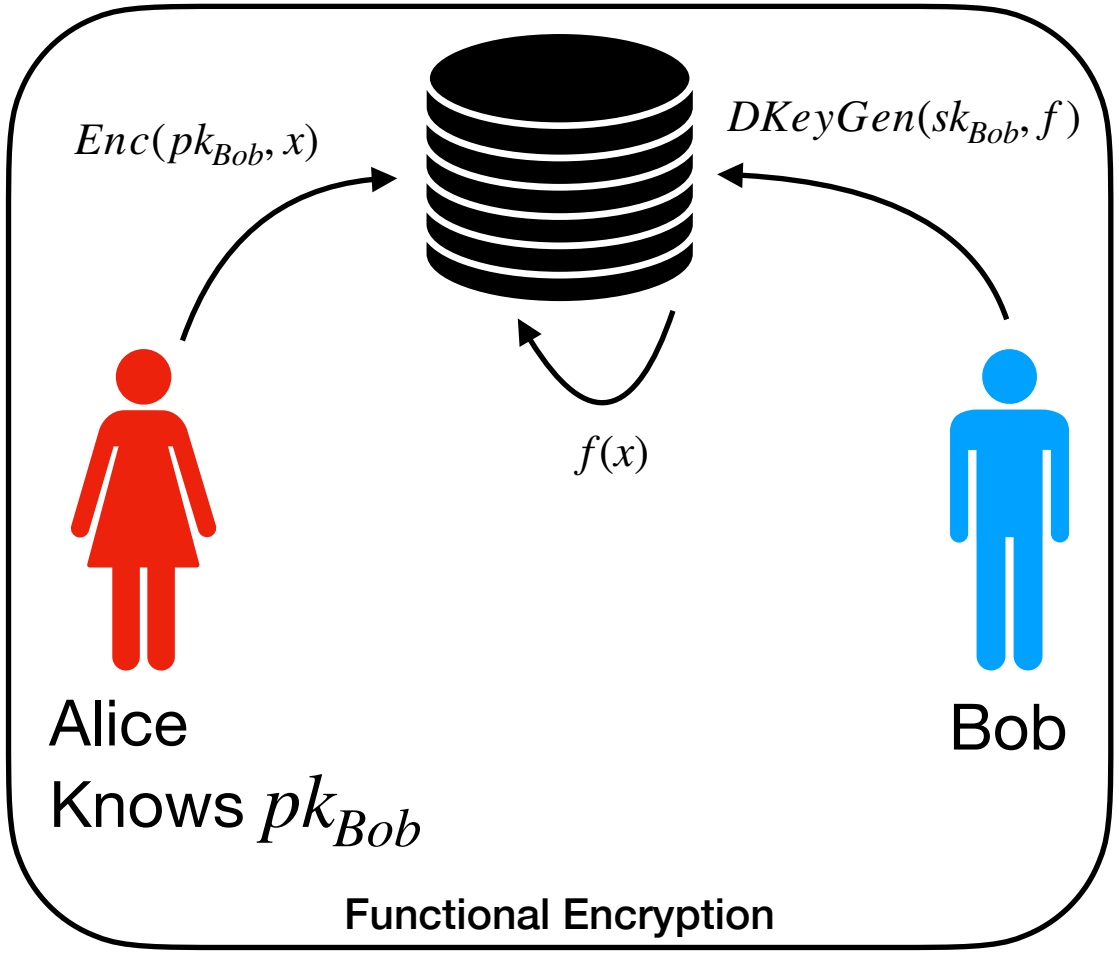




Functional Encryption vs Fully Homomorphic Encryption

Two techniques for computing on encrypted data





Functional Encryption

Formal definition

- $KeyGen()$: Samples and returns (sk, pk).
- $Encrypt(pk, x)$: Encrypts message x under public key pk, returning c.
- $DKeyGen(sk, f)$: Generate and return functional key dk_f for function f.
- $Decrypt(dk_f, c)$: Compute $f(x)$ in plaintext from a cipher text and a functional decryption key.

$$Decrypt(DKeyGen(sk, f), Encrypt(pk, x)) = f(x) \text{ when } sk, pk \leftarrow KeyGen()$$

Functional Encryption

Security definition (hand-waving version)

- $KeyGen()$: Samples and returns (sk, pk).
- $Encrypt(pk, x)$: Encrypts message x under public key pk, returning c.
- $DKeyGen(sk, f)$: Generate and return functional key dk_f for function f.
- $Decrypt(dk_f, c)$: Compute $f(x)$ in plaintext from a cipher text and a functional decryption key.

An attacker learns no more about x from $(\{c_i\}_i, \{dk_{f_j}\}_j)$ than $\{f_j(x_i)\}_{i,j}$

Background
Functional Encryption for Inner Products

Recap

ElGamal Public Key Encryption (1985)

- $KeyGen()$: Sample $s \xleftarrow{\$} \mathbb{Z}_p$ and return (s, g^s).
- $Encrypt(pk, x)$: Sample $r \xleftarrow{\$} \mathbb{Z}_p$. Return $(g^r, x \cdot (pk^r)) = (g^r, x \cdot g^{rs})$.
- $Decrypt(sk, c)$: Parse c as (a, b). Return $\frac{b}{a^s}$.

Recap

ElGamal Public Key Encryption (1985)

- $KeyGen()$: Sample $s \xleftarrow{\$} \mathbb{Z}_p$ and return (s, g^s).
- $Encrypt(pk, x)$: Sample $r \xleftarrow{\$} \mathbb{Z}_p$. Return $(g^r, x \cdot (pk^r)) = (g^r, x \cdot g^{rs})$.
- $Decrypt(sk, c)$: Parse c as (a, b). Return $\frac{b}{a^s}$.

$$\frac{b}{a^s} = \frac{x \cdot g^{rs}}{(g^r)^s} = x$$

Recap

ElGamal Public Key Encryption (1985)

- $KeyGen()$: Sample $s \xleftarrow{\$} \mathbb{Z}_p$ and return (s, g^s).
- $Encrypt(pk, x)$: Sample $r \xleftarrow{\$} \mathbb{Z}_p$. Return $(g^r, x \cdot (pk^r)) = (g^r, x \cdot g^{rs})$.
- $Decrypt(sk, c)$: Parse c as (a, b). Return $\frac{b}{a^s}$.

$$
\begin{aligned}
Encrypt(pk, x) \cdot Encrypt(pk, y) &= (g^{r_1} \cdot g^{r_2}, x \cdot y \cdot g^{r_1 s} \cdot g^{r_2 s}) \\
&= (g^{r_1 + r_2}, x \cdot y \cdot g^{(r_1 + r_2)s}) \\
&= Encrypt(pk, x \cdot y)
\end{aligned}
$$

Recap

ElGamal Public Key Encryption (1985)

- $KeyGen()$: Sample $s \xleftarrow{\$} \mathbb{Z}_p$ and return (s, g^s).
- $Encrypt(pk, x)$: Sample $r \xleftarrow{\$} \mathbb{Z}_p$. Return $(g^r, x \cdot (pk^r)) = (g^r, x \cdot g^{rs})$.
- $Decrypt(sk, c)$: Parse c as (a, b). Return $\frac{b}{a^s}$.

$$Encrypt(pk, g^x) \cdot Encrypt(pk, g^y) = Encrypt(pk, g^{x+y})$$

Functional Encryption for Inner Products

ABDP15

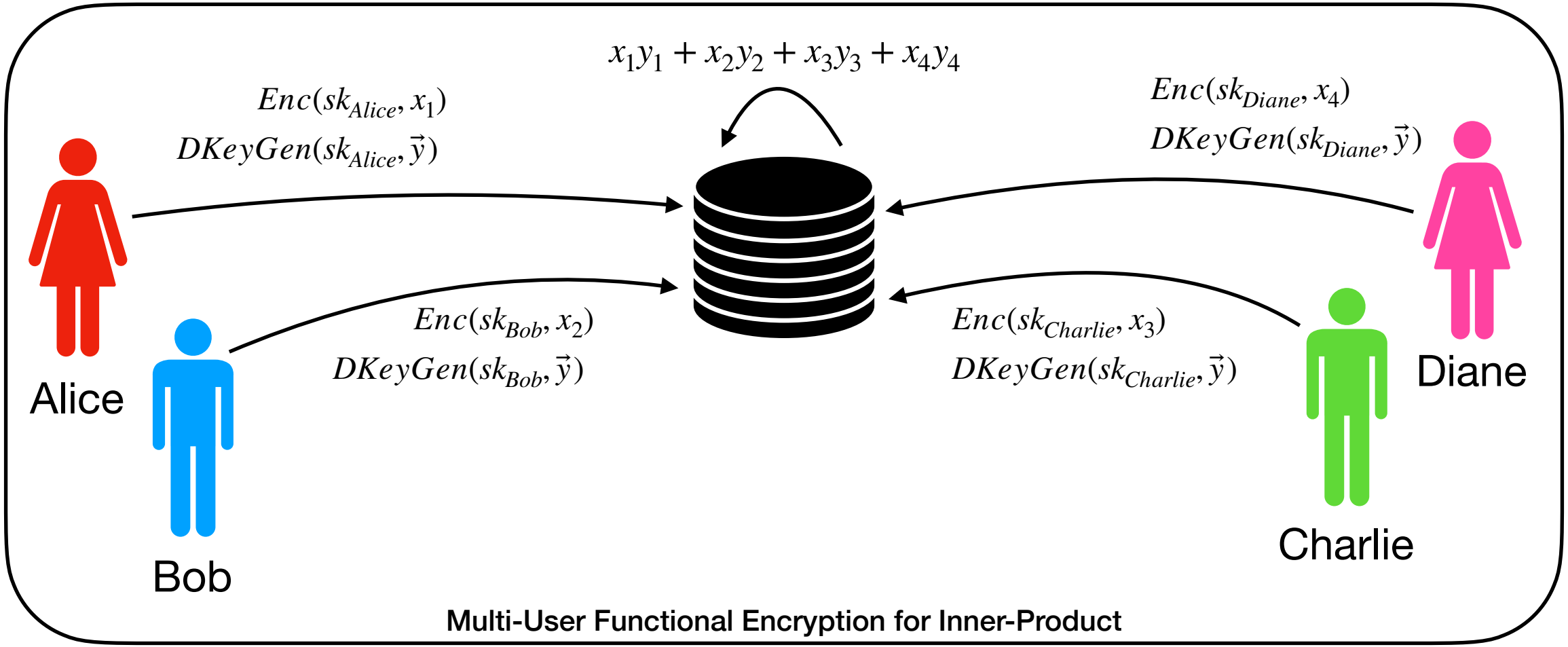
- $KeyGen()$: Sample $sk = (s_1, \ldots, s_n) \xleftarrow{\$} \mathbb{Z}_p^n$. Set $pk = (g^{s_1}, \ldots, g^{s_n})$. Return (sk, pk).
- $Encrypt(pk, \vec{x})$: Sample $r \xleftarrow{\$} \mathbb{Z}_p$. Parse $pk = (P_1, \ldots, P_n)$. Return $(g^r, g^{x_1} \cdot P_1^r, \ldots, g^{x_n} \cdot P_n^r)$.
- $DKeyGen(sk, \vec{y})$: Parse $sk = (s_1, \ldots, s_n), \vec{y} = (y_1, \ldots, y_n)$. Return $dk_{\vec{y}} = (\vec{y}, \sum_i s_i y_i)$.
- $Decrypt(dk_{\vec{y}}, c)$: Parse $c = (R, c_1, \ldots, c_n), dk_{\vec{y}} = ((y_i)_i, D_y)$. Return the discrete logarithm of $\frac{\prod_i C_i^{y_i}}{R^{D_y}}$.

$$\frac{\prod_i c_i^{y_i}}{R^{D_y}} = \frac{\prod_i (g^{x_i} \cdot P_i^r)^{y_i}}{(g^r)^{\sum_i s_i y_i}} = \frac{\prod_i (g^{x_i} \cdot (g^{s_i})^r)^{y_i}}{(g^r)^{\sum_i s_i y_i}} = g^{\sum_i x_i y_i + r s_i y_i - r s_i y_i} = g^{\sum_i x_i y_i} = g^{\langle \vec{x}, \vec{y} \rangle}$$

Can we use this to aggregate data from multiple parties?

Multi-User Functional Encryption for Inner-Product

What does it mean?



Decentralizing Inner Product Functional Encryption

Scheme	Citation	Multi-user?	Trust model	Setup	Membership	Natural Security Model	Assumptions
Single-Input Functional Encryption	ABDP15	No	N/A	N/A	N/A	Yes	Prime Order Groups
Multi-Client Functional Encryption	CDG+18	Yes	Trusted third party	Non-interactive	Static	No	ROM Prime Order Groups
Decentralized Multi-Client Functional Encryption	CDG+18	Yes	Decentralized	Interactive	Static	No	ROM Pairing Groups
Dynamic Decentralized Functional Encryption	CDG+20	Yes	Decentralized	Non-interactive	Dynamic	Yes	ROM Prime Order Groups (or Pairings)

Decentralizing ABDP15

[CDG+18]

Decentralizing ABDP15

KeyGen

- $KeyGen()$: Sample $sk = (s_1, \ldots, s_n) \xleftarrow{\$} \mathbb{Z}_p^n$. Set $pk = (g^{s_1}, \ldots, g^{s_n})$. Return (sk, pk).

KeyGen from ABDP15

Each user i can sample s_i. No need to publish g^{s_i}.

[CDG+18]

Decentralizing ABDP15

Encrypt

- $Encrypt(pk, x)$: Sample $r \xleftarrow{\$} \mathbb{Z}_p$. Parse $pk = (P_1, \ldots, P_n)$. Return $(g^r, g^{x_1} \cdot P_1^r, \ldots, g^{x_n} \cdot P_n^r)$.

Encrypt from ABDP15

How do we sample the same r distributedly?

Idea: hash functions! Use $\mathscr{H}(\ell)$ to replace g^r. Parties agree on a *label* (i.e. date).

But don't hash to $\mathbb{Z}_p$, it's easy to undo.

Hash to $\mathbb{G}$. Can get away with it because we're doing private-key encryption.

$$Encrypt(s_i, \ell, x_i) = g^{x_i} \cdot \mathscr{H}(\ell)^{s_i}$$

Decentralizing ABDP15

DKeyGen

- $DKeyGen(sk, y)$: Parse $sk = (s_1, \ldots, s_n), y = (y_1, \ldots, y_n)$. Return $(y, \sum_i s_i y_i)$.

DKeyGen from ABDP15

Huh oh, we certainly can't publish $s_i y_i$…

We need to make the sum available but each party has one part.

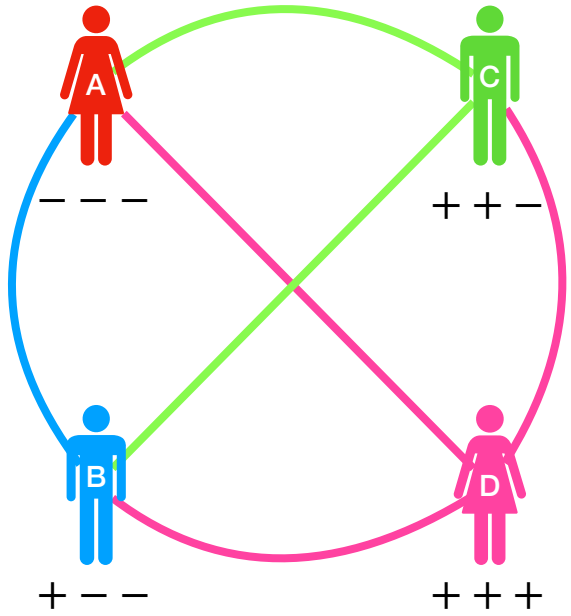
Solution 1: Use another layer of Functional Encryption via pairings. [CDG+18]

Solution 2: Use DSum! [CDG+20]

Distributed Sum

~CDG+20

- $KeyGen()$: Sample $s_i \xleftarrow{\$} \mathbb{Z}_p$. Set $pk = g^{s_i}$. Return (sk, pk).
- $Encrypt(sk_i, \ell, x_i, \{pk_j\}_j)$: Parse $sk_i = s_i$. For all $j \neq i$, compute $K_{i,j} = pk_j^{s_i} \left(= g^{s_i s_j} = K_{j,i} \right)$ and $r_{i,j} = PRF_{K_{i,j}}(\ell) \left(= r_{j,i} \right)$. Return $c_i = x_i + \sum_{j<i} r_{i,j} - \sum_{j>i} r_{i,j}$.
- $Decrypt(\epsilon, \{c_i\}_i)$: Return $\sum_i c_i = \sum_i x_i$.



Decentralized Functional Encryption for Inner Products

~CDG+20

- $KeyGen()$: Sample $s_i \xleftarrow{\$} \mathbb{Z}_p$ and run $DSum.KeyGen()$.
  Return $pk_i = DSum.pk_i, sk_i = (DSum.sk_i, s_i)$.
- $Encrypt(sk_i, \ell, x_i)$: Return $c_i = g^{x_i} \cdot \mathscr{H}(\ell)^{s_i}$.
- $DKeyGen(sk_i, \vec{y}, \{pk_j\}_j)$: Return
  $dk_{\vec{y},i} = DSum.Encrypt(DSum.sk_i, \vec{y}, s_i y_i, \{DSum.pk_j\}_j)$ where $\vec{y}$ is used as the label ℓ.
- $Decrypt(\{dk_{\vec{y},i}\}_i, \{c_i\}_i)$: Reconstruct $dk_{\vec{y}} = \sum_i s_i y_i$ via $DSum.Decrypt$.
  Return the discrete logarithm of $\frac{\prod_i c_i^{y_i}}{\mathscr{H}(\ell)^{dk_{\vec{y}}}}$.

The repeated queries / deterministic encryption trap

CDG+20

$Encrypt(sk_i, \ell, x_i)$: Return $c_i = g^{x_i} \cdot \mathscr{H}(\ell)^{s_i}$.

Encrypt from previous slide (not quite CDG+20 yet)

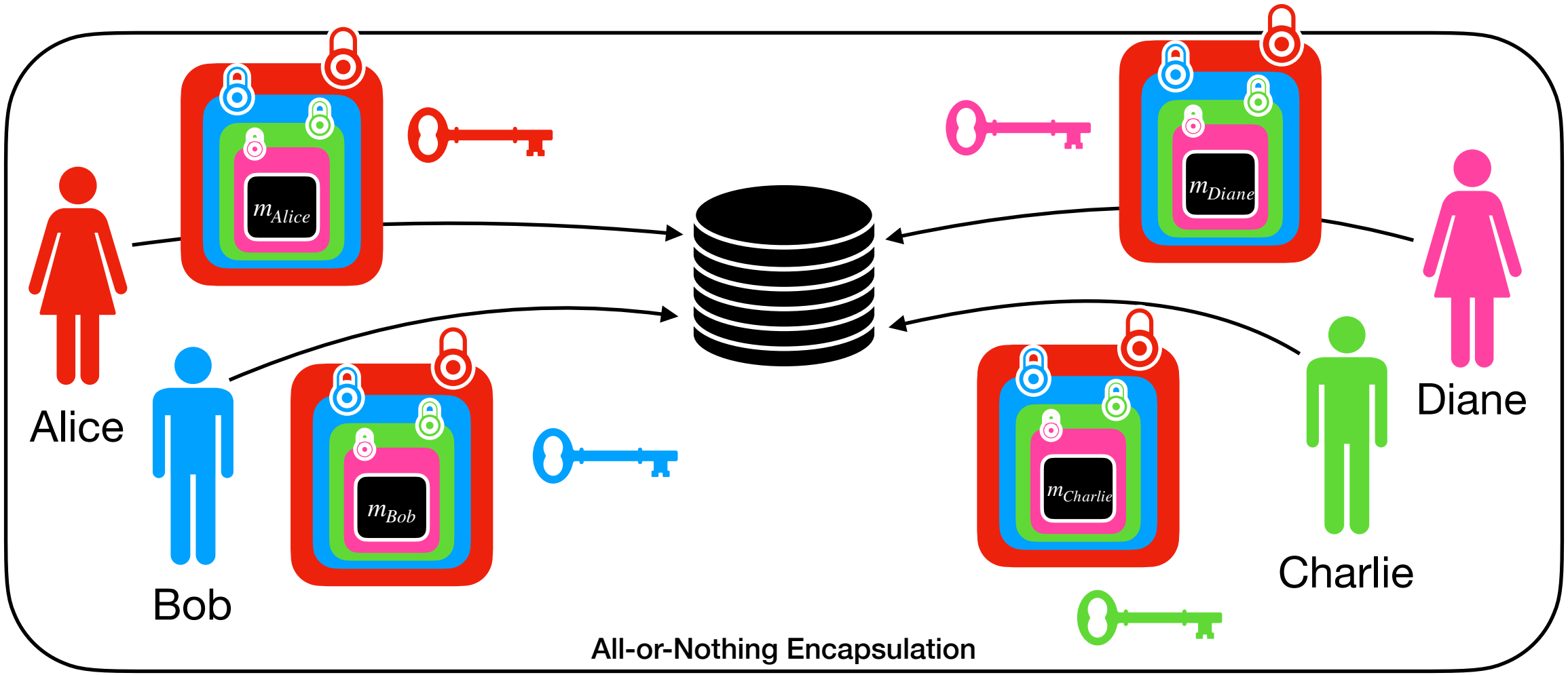
This means $Encrypt(sk_i, \ell, x_i)/Encrypt(sk_i, \ell, x_i') = g^{x_i - x_i'}$.

This reveals more information than it should and violates the security model.

We need a way to hide individual ciphertexts until all n have been collected.

All-or-Nothing Encapsulation

CDG+20



All-or-Nothing Encapsulation

CDG+20

- Wraps DSum and IP-DDFE.
- Constructed from pairings.
- Or generically from Identity-Based Encryption.
- Also implies Identity-Based Encryption.

Thank you!

Discussion topics

- Security model and privacy
- Quadratic Functional Encryption and Machine Learning
- Discrete logarithm
- Or anything on your mind!