ASIACRYPT 2023

Verifiable Decentralized Multi-Client Functional Encryption for Inner Product

Dinh Duy Nguyen
Télécom Paris, France

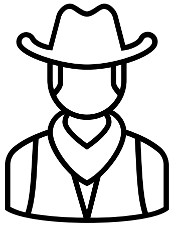
Duong Hieu Phan
Télécom Paris, France

David Pointcheval
École Normale Supérieure de Paris, France

Functional Encryption [BSW11]

(FE)

Authority



ek

$\mathrm{dk_f}$

Sender

$\mathrm{ek, sk} \leftarrow \mathrm{Setup}(1^\lambda)$

$\mathrm{dk_f} \leftarrow \mathrm{DKeyGen(sk, f)}$

Receiver

$\mathrm{ct_{data}} \leftarrow \mathrm{Enc(ek, data)}$

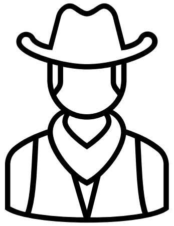
$\xrightarrow{\mathrm{ct_{data}}}$

$\mathrm{f(data)} \leftarrow \mathrm{Dec(ct_{data}, dk_f)}$

Multi-Client Functional Encryption [GGGJKLSSZ14]

(MCFE)

Authority



$\mathrm{ek_i}$ $\mathrm{dk_f}$

Sender i

$(\mathrm{ek_i})_{\mathrm{i}\in[n]}$, $\mathrm{sk} \leftarrow \mathrm{Setup}(1^{\lambda}, n)$

$\mathrm{dk_f} \leftarrow \mathrm{DKeyGen}(\mathrm{sk}, \mathrm{f})$

Receiver

$\mathrm{ct}_{\ell,\mathrm{data_i}} \leftarrow \mathrm{Enc}(\mathrm{ek_i}, \mathrm{data_i}, \ell)$

$\mathrm{ct}_{\ell,\mathrm{data_i}}$

By collecting ciphertexts of label ℓ from all senders,

$\mathrm{f}((\mathrm{data_i})_{\mathrm{i}\in[n]}) \leftarrow \mathrm{Dec}\left((\mathrm{ct}_{\ell,\mathrm{data_i}})_{\mathrm{i}\in[n]}, \mathrm{dk_f}\right)$

Decentralized MCFE [CDGPP18]

(DMCFE)

Sender i

- $\mathrm{ek_i}\ ,\mathrm{sk_i} \leftarrow \mathrm{Setup}\left(1^{\lambda}, n\right)$
- $\mathrm{ct}_{\ell,\mathrm{data_i}} \ \leftarrow \mathrm{Enc}(\mathrm{ek_i}, \mathrm{data_i}, \ell\ )$
- $\mathrm{dk_{f,i}} \leftarrow \mathrm{DKeyGenShare}(\mathrm{sk_i}, \mathrm{f})$

$\xrightarrow{\mathrm{dk_{f,i}}}$

$\xrightarrow{\mathrm{ct}_{\ell,\mathrm{data_i}}}$

Receiver

- $\mathrm{dk_f} \leftarrow \mathrm{DKeyComb}\left((\mathrm{dk_{f,i}})_{\mathrm{i}\in[n]}, \mathrm{f}\right)$
- $\mathrm{f}((\mathrm{data_i})_{\mathrm{i}\in[n]}) \leftarrow \mathrm{Dec}\left((\mathrm{ct}_{\ell,\mathrm{data_i}})_{\mathrm{i}\in[n]}, \mathrm{dk_f}, \ell\right)$

Decentralized MCFE for Inner Product

(IP-DMCFE)

- An inner product (or weighted sum) function represented by $\vec{y}$ is defined as:

$$f_{\vec{y}} : \vec{x} \mapsto \langle \vec{x}, \vec{y} \rangle = \sum_i x_i \, y_i$$

- In IP-DMCFE, each sender encrypts x_i and generates key share for y_i:

$$dk_{f_{\vec{y}}} \leftarrow \mathrm{DKeyComb}\left((dk_{y_i})_{i \in [n]}, f_{\vec{y}} \right)$$

$$\langle \vec{x}, \vec{y} \rangle \leftarrow \mathrm{Dec}\left((ct_{\ell, x_i})_{i \in [n]}, dk_{f_{\vec{y}}}, \ell \right)$$

- There are concrete DDH-based and lattice-based constructions in the literature.
- In practice, IP-DMCFE allows computing statistical analysis for private data from multiple sources.

Verifiability for IP-DMCFE*

Malicious sender i

Receiver

- Produce malformed $(\mathrm{dk_i},\ \mathrm{ct'}_{\ell,\mathrm{i}})$

$(\mathrm{dk_i}, \mathrm{y_i}) \longrightarrow$

$\mathrm{ct'}_{\ell,\mathrm{i}} \longrightarrow$

- $\mathrm{dk}_{\vec{y}} \leftarrow \mathrm{DKeyComb}\left((\mathrm{dk}_{\mathrm{y_j}})_{\mathrm{j}\in[\mathrm{n}]\setminus\{\mathrm{i}\}}, \mathrm{dk_i}, \vec{y}\right)$
- $\alpha \leftarrow \mathrm{Dec}\left((\mathrm{ct}_{\ell,\mathrm{x_j}})_{\mathrm{j}\in[n]\setminus\{\mathrm{i}\}}, \mathrm{ct'}_{\ell,\mathrm{i}}, \mathrm{dk}_{\vec{y}}, \ell\right)$

where α is strongly biased from the inner product function on the honest data set $(\mathrm{x_j})_{\mathrm{j}\in[\mathrm{n}]\setminus\{\mathrm{i}\}}$, i. e. $\sum_{\mathrm{j}\in[\mathrm{n}]\setminus\{\mathrm{i}\}} \mathrm{x_j y_j}$

*A concurrent work focuses on the input validation for Secure Aggregation: [BGLLMRY22]

Our contributions

- **Concept:** definition of verifiable DMCFE with the ability to identify malicious senders.
- **Techniques** to facilitate verification of key share:
    1. One-time Decentralized Sum (ODSUM) based on class groups.
    2. A full-fledged DSUM tailored for verification from ODSUM.
- **Scheme:** Efficient range-verifiable DMCFE for inner product.

Formalization of Verifiable DMCFE

For all users:

- Let $\mathcal{p}^{\mathfrak{m}}$ be a family of PPT message predicates and let $\mathcal{p}^{\mathfrak{f}}$ be a family of PPT function predicates
- $\mathrm{pp} \leftarrow \mathrm{Setup}(\lambda)$

For each sender:

- $[(\mathrm{sk}_i, \mathrm{ek}_i), \mathrm{vk}_{CT}, \mathrm{vk}_{DK}, \mathrm{pk}] \leftarrow \mathrm{KeyGen}(\mathrm{pp})$
- $C_{\ell,i} \leftarrow \mathrm{Enc}(\mathrm{ek}_i, x_i, \ell, P_i^{\mathfrak{m}} \in \mathcal{p}^{\mathfrak{m}})$
- $\mathrm{dk}_{f,i} \leftarrow \mathrm{DKeyGenShare}(\mathrm{sk}_i, \ell_f, P^{\mathfrak{f}} \in \mathcal{p}^{\mathfrak{f}})$

For a receiver:

- $\beta \leftarrow \mathrm{VerifyDK}(\left(\mathrm{dk}_{f,i}\right)_{i\in[n]}, \mathrm{vk}_{DK}, P^{\mathfrak{f}})$
- $\beta \leftarrow \mathrm{VerifyCT}(\left(C_{\ell,i}\right)_{i\in[n]}, \mathrm{vk}_{CT}, (P_i^{\mathfrak{m}})_{i\in[n]})$

where $\beta = 1$ for accepting and $\beta = (0, \mathcal{MS})$ for rejecting along with a set of malicious senders $\mathcal{MS}$.

- $\mathrm{dk}_f \leftarrow \mathrm{DKeyComb}(\left(\mathrm{dk}_{f,i}\right)_{i\in[n]}, \ell_f)$
- $f(x)/\perp \leftarrow \mathrm{Decrypt}(\mathrm{dk}_f, \left(C_{\ell,i}\right)_{i\in[n]})$

Formalization of Verifiability

Verifiability (for non-function-hiding): Given families of predicates $(\mathcal{P}^{\mathfrak{f}}, \mathcal{P}^{\mathfrak{m}})$, an adversary wins by one of the following conditions

$$\alpha := \mathrm{VerifyCT}\left(\left(C_{\ell,i}\right)_{i\in[HS]}, \left(C_{\ell,i}\right)_{i\in[CS]}, vk_{CT}, \left(P_i^m\right)_{i\in[n]}\right)$$

$$\beta_j := \mathrm{VerifyDK}\left(\left(dk_{f_j,i}\right)_{i\in[HS]}, \left(dk_{f_j,i}\right)_{i\in[CS]}, vk_{DK}, P^f\right)$$

- If $\alpha = 1$ and $\beta_j = 1$ for all poly. number of function queries f_j, there does not exist $(x_i)_{i\in[n]}$ such that

$$P_i^m(x_i) = 1\ \forall i \text{ and } \mathrm{Decrypt}\left(dk_{f_j}, \left(C_{\ell,i}\right)_{i\in[n]}\right) = f_j(x_1, \dots, x_n)$$

→ Input validation

- If $\alpha = (0, \mathcal{MS}_{CT})$ or $\beta_j = (0, \mathcal{MS}_{DK})$ for some function query f_j and

$$\mathcal{MS}_{CT} \cup \mathcal{MS}_{DK} \text{ contains a non-corrupted sender}$$

→ Malicious sender identification

Goal: A verifiable IP-DMCFE compatible with practical building blocks

Schnorr-like proof for discrete logarithm equalities:

$$\mathcal{R}_{\mathrm{DL}}\left(\vec{\mathrm{S}}, \vec{\mathrm{G}}; \vec{\mathrm{s}}\right) = 1 \leftrightarrow \mathrm{S_i} = \mathrm{s_i} \cdot \mathrm{G_i} \; \forall\, i$$

Range proof for Pedersen commitment:

$$\mathcal{R}_{\mathrm{range}}\left((\mathrm{com_{Ped}}, \mathrm{l}, \mathrm{r}); (\mathrm{x}, \mathrm{s})\right) = 1 \leftrightarrow \; \mathrm{com_{Ped}} = \mathrm{s} \cdot [\mathrm{h}] + [\mathrm{x}] \; \wedge \mathrm{x} \in [\mathrm{l}, \mathrm{r}]$$

MCFE scheme from [CDGPP18] (simplified):

1. $(\mathrm{ek_i})_\mathrm{i} = (\mathrm{sk_i})_\mathrm{i} = (\mathrm{s_i})_\mathrm{i} \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{n}\right)$
2. $\mathrm{ct}_{\mathrm{i},\ell} = \mathrm{s_i} \cdot [\mathrm{h}_\ell] + [\mathrm{x_i}] \leftarrow \mathrm{Enc}(\mathrm{ek_i}, \mathrm{x_i}, \ell)$ where $[\mathrm{h}_\ell] = \mathrm{hash}(\ell)$
3. $\mathrm{dk}_{\vec{\mathrm{y}}} = \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{s_i y_i} \leftarrow \mathrm{DKeyGen}((\mathrm{sk_i})_\mathrm{i}, \vec{\mathrm{y}})$

Decentralization of IP-MCFE

A decentralized sum (DSUM) is used as a generic compiler to transform MCFE to DMCFE [CDGPP20]:

Sender i

Receiver

- $\mathrm{pp} \leftarrow \mathrm{Setup}\left(1^{\lambda}, \mathrm{n}\right)$
- $\mathrm{sk_i}, \mathrm{pk} \leftarrow \mathrm{KeyGen}(\mathrm{pp})$
- $\mathrm{ct}_{\ell,\mathrm{x_i}} \leftarrow \mathrm{Enc}(\mathrm{sk_i}, \mathrm{x_i}, \ell)$

$\xrightarrow{\mathrm{ct}_{\ell,\mathrm{x_i}}}$

- $\sum_{i\in[n]} \mathrm{x_i} \leftarrow \mathrm{Dec}\left((\mathrm{ct}_{\ell,\mathrm{x_i}})_{\mathrm{i}\in[n]}\right)$

Decentralization of IP-MCFE

A decentralized sum (DSUM) is used as a generic compiler to transform MCFE to DMCFE [CDGPP20]:

Sender i

Receiver

- $\mathrm{pp} \leftarrow \mathrm{Setup}\left(1^{\lambda}, \mathrm{n}\right)$
- $\mathrm{sk_i}, \mathrm{pk} \leftarrow \mathrm{KeyGen}(\mathrm{pp})$
- $\mathrm{ct}_{\ell,\mathrm{x_i}} \leftarrow \mathrm{Enc}(\mathrm{sk_i}, \mathrm{x_i}, \ell\,)$

$\xrightarrow{\mathrm{ct}_{\ell,\mathrm{x_i}}}$

- $\sum_{i\in[n]} \mathrm{x_i} \leftarrow \mathrm{Dec}\left((\mathrm{ct}_{\ell,\mathrm{x_i}})_{\mathrm{i}\in[n]}\right)$

Problem: verification of generic-DSUM encryption is prohibitively expensive to be done by a NIZK, similarly to the verification of individual key in ACORN protocol [BGLLMRY23] (solved by MPC)

$$\boxed{\mathrm{ct}_{\ell,\mathrm{x_i}} = \mathrm{x_i} + \sum_{\mathrm{i<j}} \mathrm{PRF}(\mathrm{NIKE}(\mathrm{sk_i}, \mathrm{sk_j}), \ell) - \sum_{\mathrm{i>j}} \mathrm{PRF}(\mathrm{NIKE}(\mathrm{sk_i}, \mathrm{sk_j}), \ell)}$$

Combine-then-Descend Technique

The first attempt is to construct a proof-friendly Decentralized SUM (DSUM) with the following properties for encryption:

- Verification within constant costs;
- Input domain is $\mathbb{Z}_p$;
- No bound on the sum to be decrypted.

→ Hard to be instantiated in pairing-friendly groups.

Preliminaries on Class Groups

The CL framework for groups of unknown order [CL15, CCLST19, CCLST20].

- $\mathrm{pp} \leftarrow \mathrm{CLGen}(1^\lambda, \mathrm{p})$: 1^λ computational security parameter, $\mathrm{p} > 2^\lambda$ prime
- Cyclic group $\widehat{\mathrm{G}} \cong \widehat{\mathrm{G}}^{\mathrm{p}} \times \mathrm{F}$:
  - $\mathrm{F} = \langle \mathrm{f} \rangle$ - subgroup of order p with easy DLOG.
  - $\widehat{\mathrm{G}}^{\mathrm{p}} = \langle \hat{\mathrm{g}}_{\mathrm{p}} \rangle$ - subgroup of p-th powers in $\widehat{\mathrm{G}}$, of unknown order.
- Hardness assumptions:
  - Hidden Subgroup Membership
  - Low-ORD Assumption
  - Strong Root Assumption

For privacy

For soundness of verification

- Advantages:
  - Can choose p freely as a large prime
  - Transparent setup
  - Faster and smaller than Paillier group [BCIL22]

One-time DSUM in Class Groups

For all users:

- $\mathrm{pp} = (\mathrm{G} \cong \mathrm{G}^{\mathrm{p}} \times \mathrm{F}) = \langle \mathrm{g_p} \cdot \mathrm{f} \rangle \leftarrow \mathrm{Setup}(\lambda)$

For each sender:

- $\left(\mathrm{sk_i} = \mathrm{t_i}, \mathrm{T_i} = \mathrm{g_p^{t_i}}\right) \leftarrow \mathrm{KeyGen}(\mathrm{pp})$
- $\mathrm{pk} = (\mathrm{T_i})_{\mathrm{i} \in [\mathrm{n}]}$ is public
- $\mathrm{C_i} = \mathrm{f}^{\mathrm{x_i}} \cdot \left(\prod_{\mathrm{i}<\mathrm{j}} \mathrm{T_j} \cdot \prod_{\mathrm{i}>\mathrm{j}} \mathrm{T_j}^{-1}\right)^{\mathrm{t_i}} \leftarrow \mathrm{Enc}(\mathrm{sk_i}, \mathrm{x_i}, \mathrm{pk})$

For a receiver:

- $\mathrm{Decrypt}((\mathrm{C_i})_{\mathrm{i} \in [\mathrm{n}]})$:
    - No decryption key is required.
    - It combines ciphertexts

$$\mathrm{M} = \prod_{\mathrm{i} \in [\mathrm{n}]} \mathrm{C_i} = \mathrm{f}^{\sum_{\mathrm{i} \in [\mathrm{n}]} \mathrm{x_i}} \prod_{\mathrm{i} \in [\mathrm{n}]} \mathrm{g_p}^{\sum_{\mathrm{i}<\mathrm{j}} \mathrm{t_i t_j} - \sum_{\mathrm{i}>\mathrm{j}} \mathrm{t_i t_j}} = \mathrm{f}^{\sum_{\mathrm{i} \in [\mathrm{n}]} \mathrm{x_i}}.$$

    - By the class group property, it descends $\sum_{\mathrm{i} \in [\mathrm{n}]} \mathrm{x_i}$ from DLOG of M.

One-time DSUM in Class Groups

For all users:

- $\mathrm{pp} = (G \cong G^p \times F) = \langle g_p \cdot f \rangle \leftarrow \mathrm{Setup}(\lambda)$

For each sender:

- $\left(\mathrm{sk}_i = t_i, T_i = g_p^{t_i}\right) \leftarrow \mathrm{KeyGen}(\mathrm{pp})$
- $\mathrm{pk} = (T_i)_{i \in [n]}$ is public
- $C_i = f^{x_i} \cdot \left(\prod_{i<j} T_j \cdot \prod_{i>j} T_j^{-1}\right)^{t_i} \leftarrow \mathrm{Enc}(\mathrm{sk}_i, x_i, \mathrm{pk})$

For a receiver:

- $\mathrm{Decrypt}((C_i)_{i \in [n]})$:
  - No decryption key is required.
  - It combines ciphertexts

$$M = \prod_{i \in [n]} C_i = f^{\sum_{i \in [n]} x_i} \prod_{i \in [n]} g_p^{\sum_{i<j} t_i t_j - \sum_{i>j} t_i t_j} = f^{\sum_{i \in [n]} x_i}.$$

  - By the class group property, it descends $\sum_{i \in [n]} x_i$ from DLOG of M.

One-time DSUM is **not enough** to decentralize MCFE, as it only allows one-time secure encryption with the one-time mask $\left(\prod_{i<j} T_j \cdot \prod_{i>j} T_j^{-1}\right)^{t_i}$ from each key generation.

Bootstrapping to Label-Supporting DSUM

- IP-MCFE supports encryption that can be re-randomized by labels, i.e.,

$$\mathrm{ct}_{\mathrm{i},\ell} = \mathrm{s_i} \cdot [\mathrm{h}_\ell] + [\mathrm{x_i}] \leftarrow \mathrm{Enc}(\mathrm{ek_i}, \mathrm{x_i}, \ell) \text{ where } [\mathrm{h}_\ell] = \mathrm{hash}(\ell)$$

and becomes an MCFE for sum with a deterministic decryption key $\mathrm{dk}_{\vec{1}} = \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{s_i}$.

- Both ODSUM and IP-MCFE allows ciphertext verification by Σ-protocols.

We obtain benefits from both schemes:

1. Use ODSUM to decentralize MCFE key $\mathrm{dk}_{\vec{1}}$.
2. Decentralized MCFE for sum becomes a full-fledged DSUM with efficient verifiability.

Label-Supporting DSUM

ODSUM in class group

MCFE in prime-order group

1. Setup for all users:

ODSUM in class group	MCFE in prime-order group
• Generate class group: $(G \cong G^p \times F) = \langle g_p \cdot f \rangle$; $\mathrm{ord}(\langle f \rangle) = p$	• Generate prime-order DDH group: $(\mathbb{G}, G)$; $\mathrm{ord}(\mathbb{G}) = p$

2. Key generation for each sender:

ODSUM in class group	MCFE in prime-order group
• Generate randomly: $\left(t_i, T_i = g_p^{t_i}\right)$ • Publish T_i • $C_i := f^{s_i} \cdot \left(\prod_{i<j} T_j \cdot \prod_{i>j} T_j^{-1}\right)^{t_i}$ • Send C_i to receiver	• Generate randomly: $s_i \pmod p$

3. Encryption for each sender:

ODSUM in class group	MCFE in prime-order group
	• $ct_{i,\ell} = s_i \cdot [h_\ell] + [x_i] \leftarrow \mathrm{Enc}(ek_i, x_i, \ell)$ where $[h_\ell] = \mathrm{hash}(\ell)$

4. Decryption for receiver:

ODSUM in class group	MCFE in prime-order group
• Combine ciphertexts $M = \prod_{i\in[n]} C_i = f^{\sum_{i\in[n]} s_i} \prod_{i\in[n]} g_p^{\sum_{i<j} t_i t_j - \sum_{i>j} t_i t_j} = f^{\sum_{i\in[n]} s_i}$ • Descend $\sum_{i\in[n]} \mathbf{s_i}$ $(\mathbf{mod\ p})$ from DLOG of M.	• $\mathbf{dk}_{\vec{1}} = \sum_{i=1}^{n} \mathbf{s_i}\ (\mathbf{mod\ p})$ • $[\alpha] = \sum_{i=1}^{n} ct_{i,\ell} - [h_\ell] \cdot dk_{\vec{1}} = [\sum_{i=1}^{n} x_i]$ • **Compute a discrete logarithm** to recover α.

One-time run suffices

Using LDSUM to Decentralize MCFE

We use a pairing group $(G_T, G_1, G_2, e: G_1 \times G_2 \rightarrow G_T \ s.t.\ e([a]_1, [b]_2) = [ab]_T)$ to avoid the discrete log calculation in LDSUM decryption.

MCFE in G_1:
$(\mathrm{ct}_{\mathrm{i},\ell} = \mathrm{s}_\mathrm{i} \cdot [\mathrm{h}_\ell]_1 + [\mathrm{x}_\mathrm{i}]_1)_{\mathrm{i} \in [\mathrm{n}]}$

LDSUM in G_2 (with the same class group operations):
$\left[\mathrm{dk}_{\vec{y}}\right]_2 = [\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{s}_\mathrm{i}\, \mathrm{y}_\mathrm{i}]_2$

DMCFE inner product to decrypt in G_T:
$[\alpha]_T = \sum_{i \in [n]} e(\mathrm{ct}_{\mathrm{i},\ell}, [\mathrm{y}_\mathrm{i}]_2) - e\left([\mathrm{h}_\ell]_1, \left[\mathrm{dk}_{\vec{y}}\right]_2\right) = [\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x}_\mathrm{i}\, \mathrm{y}_\mathrm{i}]_T$

Using LDSUM to Decentralize MCFE

We use a pairing group $(G_T, G_1, G_2, e: G_1 \times G_2 \to G_T \ s.t.\ e([a]_1, [b]_2) = [ab]_T)$ to avoid the discrete log calculation in LDSUM decryption.

MCFE in G_1:
$(\mathrm{ct}_{\mathrm{i},\ell} = \mathrm{s_i} \cdot [\mathrm{h}_\ell]_1 + [\mathrm{x_i}]_1)_{\mathrm{i}\in[\mathrm{n}]}$

LDSUM in G_2 (with the same class group operations):
$$[\mathrm{dk}_{\vec{y}}]_2 = [\textstyle\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{s_i}\, \mathrm{y_i}]_2$$

DMCFE inner product to decrypt in G_T:
$$[\alpha]_T = \textstyle\sum_{i\in[n]} e(\mathrm{ct}_{\mathrm{i},\ell}, [\mathrm{y_i}]_2) - e\left([\mathrm{h}_\ell]_1, [\mathrm{dk}_{\vec{y}}]_2\right) = [\textstyle\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i}\, \mathrm{y_i}]_T$$

- Key share (LDSUM encryption) verification can be done by a Σ-protocol w.r.t. committed $\mathrm{s_i}$ and public $\mathrm{y_i}$.
- Ciphertext (MCFE encryption) verification can be done by a range-proof + Σ-protocol w.r.t commited $\mathrm{s_i}$ and encrypted $\mathrm{x_i}$.

Efficiency

	Proving time	Proof size	Verifying time
Each ciphertext	12m + 7 GE , O(m) SO	2log(m) + 7 G, 10 S	2m + 2log(m) + 19 GE, O(m) SO
Each key share	16 GE, O(1) SO	8 G, 6 S	24 GE

GE: group exponentiations
SO: scalar operations
G: group elements
S: scalars
*These costs are estimated when Bulletproof is instantiated for $[0, 2^m - 1]$ range proof.

Batch Verification: Receiver can verify if all ciphertexts/key shares are correct or not

- For n key shares: 2n group exponentiations and 6 pairings
- For n ciphertexts: 3 multi-exponentiations of size (3 + 2n), a multi-exponentiation of size 2m + 3 + n(2 log(m) + 5), and O(n · m) scalar operations.

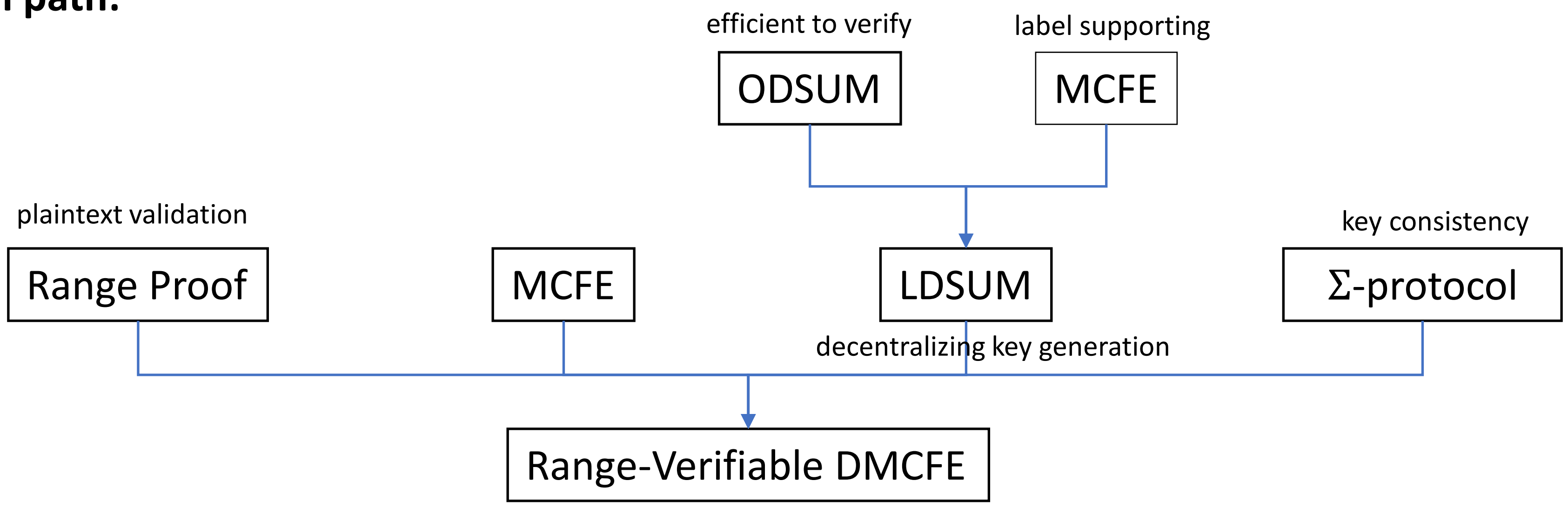
Security

- Verifiability
  1. Ciphertext verification: ROM + DDH Assump. (Symmetric eXternal Diffie Hellman pairing group).
  2. Functional key share verification: ROM + Low-ORD Assump. (class group) + Strong Root Assump. (class group).

- Static-corruption Indistinguishability: ROM + HSM Assump. (class group) + SXDH Assump. (pairing group).

Range-Verifiable Inner-Product DMCFE

Construction path:



Conclusion:

1. The scheme supports efficiently identifying an unbounded number of malicious senders with no additional interactivity between users.
2. Any other Pedersen-based proof for message predicates can also be applied.
3. LDSUM can also be used as a verifiable sharing of group identity.
4. Open question: a verifiable Dynamic IP-DMCFE?

More details at: https://ia.cr/2023/268. Thank you!