Fixing Issues and Achieving Maliciously Secure Verifiable Aggregation in "VeriFL: Communication-Efficient and Fast Verifiable Aggregation for Federated Learning"*

Xiaojie Guo
Nankai University
xiaojie.guo@mail.nankai.edu.cn

August 24, 2022

Abstract

This work addresses the security flaw in the original VeriFL protocol and proposes a patched protocol. The patched protocol is secure against any static malicious adversary with a certain threshold and only introduces moderate modifications to the original protocol.

1 Introduction

In this work, we fix the security flaw in our original VeriFL protocol [GLL+21] and show that the patched VeriFL protocol can achieve malicious security. We do not claim the contribution of this work. The patched protocol preserves the efficiency feature (i.e., the overall communication between each party and the server is dominated by sending the masked input and receiving the masked aggregate result, no other dimension-dependent communication is required) claimed in the original work. This communication seems to be asymptotically optimal with respect to the input dimension since even *insecure* aggregation consumes the same dimension-dependent bandwidth.

The security flaw. We discuss the security flaw as follows. In brief, the published homomorphic hash values in the verification phase may help the adversary guess the input vector of an honest client if this vector itself does not have sufficient entropy (i.e., has only a few possible values). Note that a linearly homomorphic hash is a *deterministic* function of a input vector. This means that, in the ideal execution, the simulator without knowing the actual inputs of honest parties cannot simulate some hash values such that their distribution is exactly that in the real execution. Even if secret sharing scheme and commitment scheme help us hide the distribution of simulated hash values in the aggregation phase, these values still need to be revealed in the verification phase. Therefore, the distinguisher can distinguish the two executions, failing the security proof.

What CANNOT be promised by maliciously secure verifiable aggregation? The patched VeriFL protocol is maliciously secure in the sense of secure computation and does not prevent attacks that are out of the scope of secure computation, e.g., the poisoning attacks by changing the inputs of corrupt parties, or the inference attacks by using (securely) aggregate results and the inputs of corrupt parties. Moreover, since our protocol performs aggregation over $\mathbb{Z}_q^d$, we do not prevent the attacker from using crafted inputs to make aggregate results wrap around the modulus q. This wrap-around issue may be (inefficiently) addressed by additionally using range proof (see, e.g., [Bou00]) or other non-cryptographic mechanisms.

2 Preliminaries

2.1 Notations

We use κ to denote the computational security parameter. We use $[n]$ to denote the set $\{1, \ldots, n\}$ for some integer $n \in \mathbb{N}$. We denote the set of integers by $\mathbb{Z}$ and the quotient ring of integers modulo an integer $q \in \mathbb{N}$ by $\mathbb{Z}_q$. For a finite set $\mathcal{X}$, we use $|\mathcal{X}|$ to denote its size. We write $\mathsf{negl}(\kappa)$ for an unspecified negligible function in κ.

2.2 Security Model

We consider the static malicious adversary in the standard simulation-based proof (see, e.g., [Gol04, Lin16]).

*On May 20, 2022, we found that there is a flaw in the security proof of VeriFL. We thank Jinhyun So for helping us spot this issue. Please cite this work as well if you would like to cite our original paper [GLL+21].

2.3 Linearly Homomorphic Hash

For dimension $d \in \mathbb{N}$, a linearly homomorphic hash (LHH) scheme with domain $\mathbb{G}_{\mathsf{in}}^d$ and range $\mathbb{G}_{\mathsf{out}}$, where $\mathbb{G}_{\mathsf{in}}$ and $\mathbb{G}_{\mathsf{out}}$ are Abelian groups, consists of two PPT algorithms with the following syntax:

- $\mathsf{LHHpp} \leftarrow \mathsf{LHH.Setup}(1^\kappa)$. On input 1^κ, output a public parameter LHHpp. For simplicity, we assume that the other algorithm implicitly takes this public parameter as input.
- $h \leftarrow \mathsf{LHH.Hash}(\boldsymbol{x})$. On input a vector $\boldsymbol{x} \in \mathbb{G}_{\mathsf{in}}^d$, output its hash value $h \in \mathbb{G}_{\mathsf{out}}$.

We require the above LHH scheme satisfies the following properties:

- **Linearity.** For any two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{G}_{\mathsf{in}}^d$, it holds that

$$\Pr\left[h_{\boldsymbol{x}} + h_{\boldsymbol{y}} = \mathsf{LHH.Hash}(\boldsymbol{x}+\boldsymbol{y}) \;\middle|\; \begin{array}{l} \mathsf{LHHpp} \leftarrow \mathsf{LHH.Setup}(1^\kappa), \\ h_{\boldsymbol{x}} \leftarrow \mathsf{LHH.Hash}(\boldsymbol{x}),\, h_{\boldsymbol{y}} \leftarrow \mathsf{LHH.Hash}(\boldsymbol{y}) \end{array}\right] = 1.$$

- **Collision resistance.** For any $d \in \mathbb{N}$ and any PPT adversary $\mathcal{A}$, it holds that

$$\Pr\left[\mathsf{Expt}_{\mathsf{coll}}^{d,\mathcal{A},\mathsf{LHH}}(1^\kappa) = 1\right] \leq \mathsf{negl}(\kappa),$$

where the collision experiment $\mathsf{Expt}_{\mathsf{coll}}^{d,\mathcal{A},\mathsf{LHH}}(1^\kappa)$ is defined as follows:

Experiment $\mathbf{Expt}_{\mathbf{coll}}^{d,\mathcal{A},\mathbf{LHH}}(1^\kappa)$

1. Generate $\mathsf{LHHpp} \leftarrow \mathsf{LHH.Setup}(1^\kappa)$ and send it to $\mathcal{A}$.
2. Receive two distinct $\boldsymbol{x}_0, \boldsymbol{x}_1 \in \mathbb{G}_{\mathsf{in}}^d$ from $\mathcal{A}$.
3. Output 1 if $\mathsf{LHH.Hash}(\boldsymbol{x}_0) = \mathsf{LHH.Hash}(\boldsymbol{x}_1)$; otherwise output 0.

In this work, we can instantiate this LHH scheme with [BGG95] under the hardness assumption of discrete logarithm over some finite group $\mathbb{G}$ with prime order q. The resulting LHH scheme sets $\mathbb{G}_{\mathsf{in}} = \mathbb{Z}_q$ and $\mathbb{G}_{\mathsf{out}} = \mathbb{G}$.

2.4 Non-interactive Commitment with Equivocality and Straight-line Extractability

We focus on the non-interactive commitment scheme in the random oracle model (ROM) [BR93]. In ROM, a non-interactive commitment scheme with message space $\mathcal{M}$, randomness space $\mathcal{R}$, commitment space C, and a random oracle $O = O_{\mathcal{M},\mathcal{R},C}$, consists of two PPT algorithms with the access to O and the following syntax:

- $\mathsf{com} \leftarrow \mathsf{COM.Commit}^O(m, r)$. On input a message $m \in \mathcal{M}$ and a randomness $r \in \mathcal{R}$, output a commitment $\mathsf{com} \in C$.
- $\{0,1\} \leftarrow \mathsf{COM.Open}^O(\mathsf{com}, m', r')$. On input a commitment string $\mathsf{com} \in C$, a message $m' \in \mathcal{M}$, and a randomness $r' \in \mathcal{R}$, output whether or not the three transcripts are consistent, i.e., $\mathsf{com} = \mathsf{COM.Commit}^O(m', r')$.

We require the above commitment scheme satisfies the following properties:

- **Hiding.** For any PPT adversary $\mathcal{A}^O$, it holds that

$$\left|\Pr\left[\mathsf{Expt}_{\mathsf{hiding}}^{\mathcal{A}^O,\mathsf{COM},0}(1^\kappa) = 1\right] - \Pr\left[\mathsf{Expt}_{\mathsf{hiding}}^{\mathcal{A}^O,\mathsf{COM},1}(1^\kappa) = 1\right]\right| \leq \mathsf{negl}(\kappa),$$

where the hiding experiment $\mathsf{Expt}_{\mathsf{hiding}}^{\mathcal{A}^O,\mathsf{COM},b}(1^\kappa)$ for $b \in \{0,1\}$ is defined as follows:

Experiment $\mathbf{Expt}_{\mathbf{hiding}}^{\mathcal{A}^O,\mathbf{COM},b}(1^\kappa)$

1. Receive two distinct $m_0, m_1 \in \mathcal{M}$ from $\mathcal{A}^O$, sample $r \leftarrow \mathcal{R}$, and send $\mathsf{com} \leftarrow \mathsf{COM.Commit}^O(m_b, r)$ to $\mathcal{A}^O$.
2. Output the guessing bit b' output by $\mathcal{A}^O$.

- **Binding.** For any PPT adversary $\mathcal{A}^{\mathcal{O}}$, it holds that

$$\Pr\left[\mathsf{Expt}_{\mathsf{binding}}^{\mathcal{A}^{\mathcal{O}},\mathsf{COM}}(1^\kappa)=1\right]\leq \mathsf{negl}(\kappa),$$

where the binding experiment $\mathsf{Expt}_{\mathsf{binding}}^{\mathcal{A}^{\mathcal{O}},\mathsf{COM}}(1^\kappa)$ is defined as follows:

Experiment $\mathbf{Expt}_{\mathbf{binding}}^{\mathcal{A}^{\mathcal{O}},\mathbf{COM}}(1^\kappa)$

1. Receive $(\mathsf{com},(m,r),(m',r'))$ from $\mathcal{A}^{\mathcal{O}}$, where $m\neq m'$.
2. Output 1 if $(\mathsf{COM.Open}^{\mathcal{O}}(\mathsf{com},m,r)=1)\wedge(\mathsf{COM.Open}^{\mathcal{O}}(\mathsf{com},m',r')=1)$; otherwise output 0.

- **Equivocality.** In the ideal world, there exists a PPT equivocation algorithm $\mathsf{COM.Equiv}^{\mathcal{O}}$ such that, for any $m,m'\in\mathcal{M}$,

$$\Pr\left[\mathsf{COM.Open}^{\mathcal{O}}(\mathsf{com},m',r')=1 \;\middle|\; \begin{array}{l} r\leftarrow\mathcal{R},\mathsf{com}\leftarrow\mathsf{COM.Commit}^{\mathcal{O}}(m,r),\\ r'\leftarrow\mathsf{COM.Equiv}^{\mathcal{O}}(\mathsf{com},(m,r),m')\end{array}\right]\geq 1-\mathsf{negl}(\kappa).$$

- **Straight-line extractability.** In the ideal world, there exists a PPT straight-line extractor $\mathsf{COM.Extract}^{\mathcal{O}}$ such that, for any $m\in\mathcal{M}$ and $r\in\mathcal{R}$,

$$\Pr\left[m=m^* \;\middle|\; \begin{array}{l}\mathsf{com}\leftarrow\mathsf{COM.Commit}^{\mathcal{O}}(m,r),\\ m^*\leftarrow\mathsf{COM.Extract}^{\mathcal{O}}(\mathsf{com})\end{array}\right]\geq 1-\mathsf{negl}(\kappa).$$

It is known in [Pas03] that, by using $\mathcal{O}:\mathcal{M}\times\mathcal{R}\to\mathcal{C}$, the folklore commitment scheme, where $\mathsf{COM.Commit}^{\mathcal{O}}(m,r):=\mathcal{O}(m,r)$, is secure (i.e., hiding and binding) and straight-line extractable in ROM. This scheme also satisfies the equivocality due to the programmability of random oracle: given a well-formed commitment $\mathsf{com}=\mathsf{COM.Commit}^{\mathcal{O}}(m,r)$ and a message m', the equivocation algorithm $\mathsf{COM.Equiv}^{\mathcal{O}}$ samples a fresh randomness $r'\leftarrow\mathcal{R}$ and programs $\mathcal{O}$ such that $\mathcal{O}(m',r')=\mathsf{com}$. It is clear that the equivocality holds if this programming succeeds. The probability that the programming fails is bounded by that for the event that com is not a uniform value or $\mathcal{O}(m',r')$ was defined. Taking a union bound, this probability is at most $2q/|\mathcal{R}|=\mathsf{negl}(\kappa)$ since we only make $q=\mathsf{poly}(\kappa)$ commitments/decommitments and $|\mathcal{R}|$ can be exponentially large in κ.

We instantiate this folklore commitment scheme with $\mathcal{M}=\mathcal{R}=\mathcal{C}=\{0,1\}^\kappa$ and have the following lemma.

Lemma 1. *For any C such that $|C|\approx 2^\kappa$ and any PPT algorithm $\mathcal{A}'^{\mathcal{O}}(1^\kappa)$ that returns a commitment $com\in C$ and an opening string $(m,r)\in\mathcal{M}\times\mathcal{R}$, it holds for the folklore commitment scheme in ROM that*

$$\Pr\left[\mathsf{COM.Open}^{\mathcal{O}}(com,m,r)=1 \;\middle|\; \begin{array}{l}(com,m,r)\leftarrow\mathcal{A}'^{\mathcal{O}}(1^\kappa),\\ ((m,r),com)\notin\mathcal{L}\end{array}\right]\leq\mathsf{negl}(\kappa),$$

where $\mathcal{L}$ denotes the list that records $\mathcal{A}'^{\mathcal{O}}$'s queries to $\mathcal{O}$ and their responses.

Proof. Note that the event $\mathsf{COM.Open}^{\mathcal{O}}(\mathsf{com},m,r)=1$ happens if and only if $\mathsf{com}=\mathsf{COM.Commit}^{\mathcal{O}}(m,r)=\mathcal{O}(m,r)$, which in turn occurs with probability at most $1/|C|$ (exactly equaling $1/|C|$ if $\mathcal{O}(m,r)$ was not defined) according to the randomness of $\mathcal{O}$. By using an exponentially large C, we can see this lemma holds. □

2.5 Key Agreement

A key agreement scheme with secret key space $\mathcal{K}_{\mathsf{sk}}$, public key space $\mathcal{K}_{\mathsf{pk}}$, and agreed key space $\mathcal{K}$, consists of three PPT algorithms with the following syntax:

- $\mathsf{KApp}\leftarrow\mathsf{KA.Setup}(1^\kappa)$. On input 1^κ, output a public parameter KApp. For simplicity, we assume that the other algorithms implicitly take this public parameter as input.
- $(\mathsf{sk},\mathsf{pk})\leftarrow\mathsf{KA.KeyGen}(1^\kappa)$. On input 1^κ, output a key pair $(\mathsf{sk},\mathsf{pk})\in\mathcal{K}_{\mathsf{sk}}\times\mathcal{K}_{\mathsf{pk}}$.
- $\mathsf{ak}_{i,j}\leftarrow\mathsf{KA.Agree}(\mathsf{sk}_i,\mathsf{pk}_j)$. On input a secret key sk_i and a public key pk_j, output an agreed key $\mathsf{ak}_{i,j}\in\mathcal{K}$.

We require the above key agreement scheme satisfies the following properties:

- **Correctness.** It holds that

$$\Pr\left[\mathsf{KA.Agree}(\mathsf{sk}_i,\mathsf{pk}_j)=\mathsf{KA.Agree}(\mathsf{sk}_j,\mathsf{pk}_i) \;\middle|\; \begin{array}{l}\mathsf{KApp}\leftarrow\mathsf{KA.Setup}(1^\kappa),\\ (\mathsf{sk}_i,\mathsf{pk}_i)\leftarrow\mathsf{KA.KeyGen}(1^\kappa),(\mathsf{sk}_j,\mathsf{pk}_j)\leftarrow\mathsf{KA.KeyGen}(1^\kappa)\end{array}\right]=1.$$

- **Security.** For any PPT adversary $\mathcal{A}$, it holds that

$$\left|\Pr\left[\mathsf{Expt}_{\mathsf{key}}^{\mathcal{A},\mathsf{KA},0}(1^\kappa)=1\right]-\Pr\left[\mathsf{Expt}_{\mathsf{key}}^{\mathcal{A},\mathsf{KA},1}(1^\kappa)=1\right]\right|\leq \mathsf{negl}(\kappa),$$

where the experiment $\mathsf{Expt}_{\mathsf{key}}^{\mathcal{A},\mathsf{KA},b}(1^\kappa)$ for $b\in\{0,1\}$ is defined as follows:

Experiment $\mathbf{Expt}_{\mathbf{key}}^{\mathcal{A},\mathbf{KA},b}(1^\kappa)$

1. Generate $\mathsf{KApp}\leftarrow\mathsf{KA.Setup}(1^\kappa)$, $(\mathsf{sk}_i,\mathsf{pk}_i)\leftarrow\mathsf{KA.KeyGen}(1^\kappa)$, and $(\mathsf{sk}_j,\mathsf{pk}_j)\leftarrow\mathsf{KA.KeyGen}(1^\kappa)$.
2. If $b=0$, sample $\mathsf{ak}_{i,j}\leftarrow\mathcal{K}$. If $b=1$, generate $\mathsf{ak}_{i,j}\leftarrow\mathsf{KA.Agree}(\mathsf{sk}_i,\mathsf{pk}_j)$.
3. Send $(\mathsf{KApp},\mathsf{pk}_i,\mathsf{pk}_j,\mathsf{ak}_{i,j})$ to $\mathcal{A}$ and give $\mathcal{A}$ the oracle access to $\mathsf{KA.Agree}(\mathsf{sk}_i,\cdot)$ and $\mathsf{KA.Agree}(\mathsf{sk}_j,\cdot)$.
4. Output the guessing bit b' output by $\mathcal{A}$.

This key agreement scheme can be instantiated with [BIK+17] under the two oracle Diffie-Hellman (2ODH) assumption over some finite group $\mathbb{G}$ with order q. This scheme is with $\mathcal{K}_{\mathsf{sk}}=\mathbb{Z}_q$, $\mathcal{K}_{\mathsf{pk}}=\mathbb{G}$, and $\mathcal{K}=\{0,1\}^\kappa$.

2.6 Secret Sharing

For threshold $t\in\mathbb{N}$ and number $n\in\mathbb{N}$ of parties where $t\leq n$, a (t,n)-secret sharing scheme with secret space $\mathcal{M}$, and share space $\mathcal{S}$, consists of three PPT algorithms with the following syntax:

- $\{\langle s\rangle_i\}_{i\in\mathcal{P}}\leftarrow\mathsf{SS.Share}(s,\mathcal{P})$. On input a secret $s\in\mathcal{M}$ and an n-sized set $\mathcal{P}$ of parties, output a set of secret shares $\{\langle s\rangle_i\}_{i\in\mathcal{P}}\subseteq\mathcal{S}$.
- $\{s',\perp\}\leftarrow\mathsf{SS.Combine}(\{\langle s\rangle_i\}_{i\in\mathcal{P}'},\mathcal{P}')$. On input a set of secret shares $\{\langle s\rangle_i\}_{i\in\mathcal{P}'}\subseteq\mathcal{S}$ and a set $\mathcal{P}'$ of parties, output a secret $s'\in\mathcal{M}$ or an abort symbol $\perp$.

We require the above secret sharing scheme satisfies the following properties:

- **Correctness.** For any $t,n\in\mathbb{N}$ where $t\leq n$, any $s\in\mathcal{M}$, any n-sized set $\mathcal{P}$, and any $\mathcal{P}'\subseteq\mathcal{P}$ where $|\mathcal{P}'|\geq t$, it holds that

$$\Pr\left[s=s'\;\middle|\;\begin{array}{l}\{\langle s\rangle_i\}_{i\in\mathcal{P}}\leftarrow\mathsf{SS.Share}(s,\mathcal{P}),\\ s'\leftarrow\mathsf{SS.Combine}(\{\langle s\rangle_i\}_{i\in\mathcal{P}'},\mathcal{P}')\end{array}\right]=1.$$

- **Reverse samplability.** For the secret sharing scheme SS, there also exists a PPT algorithm SS.RSample such that, for any $t,n\in\mathbb{N}$ where $t\leq n$, any $s\in\mathcal{M}$, any n-sized set $\mathcal{P}$, any subsets $\mathcal{P}',\mathcal{P}_\perp\subseteq\mathcal{P}$ where $|\mathcal{P}'|\geq t$ and $|\mathcal{P}_\perp|<t$, and any (unqualified) set of shares $\{\langle s\rangle_i\}_{i\in\mathcal{P}_\perp}\in\mathcal{S}^{|\mathcal{P}_\perp|}$, it holds that

$$\Pr\left[s=s'\;\middle|\;\begin{array}{l}\{\langle s\rangle_i\}_{i\in\mathcal{P}\setminus\mathcal{P}_\perp}\leftarrow\mathsf{SS.RSample}(s,\mathcal{P},\mathcal{P}_\perp,\{\langle s\rangle_i\}_{i\in\mathcal{P}_\perp}),\\ s'\leftarrow\mathsf{SS.Combine}(\{\langle s\rangle_i\}_{i\in\mathcal{P}'},\mathcal{P}')\end{array}\right]=1.$$

- **Security.** For any $t,n\in\mathbb{N}$ where $t\leq n$, and any PPT adversary $\mathcal{A}$, it holds that

$$\left|\Pr\left[\mathsf{Expt}_{\mathsf{share}}^{t,n,\mathcal{A},\mathsf{SS},0}(1^\kappa)\right]-\Pr\left[\mathsf{Expt}_{\mathsf{share}}^{t,n,\mathcal{A},\mathsf{SS},1}(1^\kappa)\right]\right|\leq\mathsf{negl}(\kappa),$$

where the experiment $\mathsf{Expt}_{\mathsf{share}}^{t,n,\mathcal{A},\mathsf{SS},b}(1^\kappa)$ for $b\in\{0,1\}$ is defined as follows:

Experiment $\mathbf{Expt}_{\mathbf{share}}^{t,n,\mathcal{A},\mathbf{SS},b}(1^\kappa)$

1. Receive from $\mathcal{A}$ a secret $s\in\mathcal{M}$, an n-sized set $\mathcal{P}$, and a subset $\mathcal{P}_\perp\subseteq\mathcal{P}$ where $|\mathcal{P}_\perp|<t$.
2. If $b=0$, sample $\{\langle s\rangle_i\}_{i\in\mathcal{P}_\perp}\leftarrow\mathcal{S}^{|\mathcal{P}_\perp|}$; otherwise generate $\{\langle s\rangle_i\}_{i\in\mathcal{P}}\leftarrow\mathsf{SS.Share}(s_b,\mathcal{P})$. Send $\{\langle s\rangle_i\}_{i\in\mathcal{P}_\perp}$ to $\mathcal{A}$.
3. Output the guessing bit b' output by $\mathcal{A}$.

This secret sharing scheme can be instantiated with the well-known Shamir's secret sharing [Sha79] with $\mathcal{M}=\mathcal{S}=\mathbb{F}^2$, where $\mathbb{F}$ is a finite field such that $|\mathbb{F}|\geq\max(2^\kappa,q)$.

2.7 Symmetric Encryption

A symmetric encryption scheme with key space $\mathcal{K}$, plaintext space $\mathcal{M}$, and ciphertext space $\mathcal{C}$, consists of three PPT algorithms with the following syntax:

- $k \leftarrow \mathsf{SE.KeyGen}(1^\kappa)$. On input 1^κ, output a key $k \in \mathcal{K}$.
- $c \leftarrow \mathsf{SE.Enc}(k, m)$. On input a key $k \in \mathcal{K}$ and a plaintext $m \in \mathcal{M}$, output a ciphertext $c \in \mathcal{C}$.
- $\{m', \perp\} \leftarrow \mathsf{SE.Dec}(k', c)$. On input a key $k' \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$, output a plaintext $m' \in \mathcal{M}$ or an abort symbol $\perp$.

We require the above symmetric encryption scheme satisfies the following properties:

- **Correctness.** For any $m \in \mathcal{M}$, it holds that

$$\Pr\left[m = m' \;\middle|\; \begin{array}{l} k \leftarrow \mathsf{SE.KeyGen}(1^\kappa), \\ c \leftarrow \mathsf{SE.Enc}(k, m), m' \leftarrow \mathsf{SE.Dec}(k, c) \end{array}\right] = 1.$$

- **IND-CCA security.** For any PPT adversary $\mathcal{A}$, it holds that

$$\left|\Pr\left[\mathsf{Expt}^{\mathcal{A},\mathsf{SE},0}_{\mathsf{IND\text{-}CCA}}(1^\kappa) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathcal{A},\mathsf{SE},1}_{\mathsf{IND\text{-}CCA}}(1^\kappa) = 1\right]\right| \leq \mathsf{negl}(\kappa),$$

where the IND-CCA experiment $\mathsf{Expt}^{\mathcal{A},\mathsf{SE},b}_{\mathsf{IND\text{-}CCA}}(1^\kappa)$ for $b \in \{0,1\}$ is defined as follows:

Experiment $\mathbf{Expt}^{\mathcal{A},\mathbf{SE},b}_{\mathbf{IND\text{-}CCA}}(1^\kappa)$

1. Generate $k \leftarrow \mathsf{SE.KeyGen}(1^\kappa)$ and give $\mathcal{A}$ the oracle access to $\mathsf{LR}_{k,b}(\cdot,\cdot)$ and $\mathsf{SE.Dec}(k,\cdot)$, where (i) $\mathsf{LR}_{k,b}(\cdot,\cdot)$ takes two $m_0, m_1 \in \mathcal{M}$ as input and outputs $\mathsf{SE.Enc}(k, m_b)$, and (ii) $\mathcal{A}$ cannot queries $\mathsf{SE.Dec}(k,\cdot)$ with any ciphertext output by $\mathsf{LR}_{k,b}(\cdot,\cdot)$.
2. Output the guessing bit b' output by $\mathcal{A}$.

This symmetric encryption scheme can be instantiated with AES with 128-bit keys.

2.8 Digital signature

A digital signature scheme with message space $\mathcal{M}$, signing key space $\mathcal{K}_{\mathsf{sigk}}$, verification key space $\mathcal{K}_{\mathsf{verk}}$, and signature space $\mathcal{S}$, consists of three PPT algorithms with the following syntax:

- $(\mathsf{sigk}, \mathsf{verk}) \leftarrow \mathsf{SIG.KeyGen}(1^\kappa)$. On input 1^κ, output a key pair $(\mathsf{sigk}, \mathsf{verk}) \in \mathcal{K}_{\mathsf{sigk}} \times \mathcal{K}_{\mathsf{verk}}$.
- $\Sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sigk}, m)$. On input a signing key $\mathsf{sigk} \in \mathcal{K}_{\mathsf{sigk}}$ and a message $m \in \mathcal{M}$, output a signature $\Sigma \in \mathcal{S}$.
- $\{0,1\} \leftarrow \mathsf{SIG.Ver}(\mathsf{verk}, m, \Sigma)$. On input a verification key $\mathsf{verk} \in \mathcal{K}_{\mathsf{verk}}$, a message $m \in \mathcal{M}$, and a signature $\Sigma \in \mathcal{S}$, output whether or not Σ is a valid signature on m.

We require the above digital signature scheme satisfies the following properties:

- **Correctness.** For any $m \in \mathcal{M}$, it holds that

$$\Pr\left[\mathsf{SIG.Ver}(\mathsf{verk}, m, \Sigma) = 1 \;\middle|\; \begin{array}{l} (\mathsf{sigk}, \mathsf{verk}) \leftarrow \mathsf{SIG.KeyGen}(1^\kappa), \\ \Sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sigk}, m) \end{array}\right] = 1.$$

- **Security.** For any PPT adversary $\mathcal{A}$, it holds that

$$\Pr\left[\mathsf{Expt}^{\mathcal{A},\mathsf{SIG}}_{\mathsf{forge}}(1^\kappa) = 1\right] \leq \mathsf{negl}(\kappa),$$

where the experiment $\mathsf{Expt}^{\mathcal{A},\mathsf{SIG}}_{\mathsf{forge}}(1^\kappa)$ is defined as follows:

Experiment $\mathbf{Expt}^{\mathcal{A},\mathbf{SIG}}_{\mathbf{forge}}(1^\kappa)$

1. Generate $(\mathsf{sigk}, \mathsf{verk}) \leftarrow \mathsf{SIG.KeyGen}(1^\kappa)$, send verk to $\mathcal{A}$, and give $\mathcal{A}$ the oracle access to $\mathsf{SIG.Sign}(\mathsf{sigk},\cdot)$.
2. Receive $(m, \Sigma) \in \mathcal{M} \times \mathcal{S}$ from $\mathcal{A}$.
3. Output 1 if $\mathsf{SIG.Ver}(\mathsf{verk}, m, \Sigma) = 1$ and $\mathcal{A}$ never queried $\mathsf{SIG.Sign}(\mathsf{sigk}, m)$; otherwise output 0.

In this work, we assume that each party has registered its public verification key in a public bulletin board provided by some trusted public key infrastructure (PKI). The underlying digital signature scheme is standard.

Functionality $\mathcal{F}_{\mathsf{SecVerAgg}}$

Parameters: The threshold t, the number N of parties, the batch size ℓ, and the domain $\mathbb{Z}_q^d$.

Aggregate: This procedure can be executed ℓ times. Upon receiving $(\mathsf{aggregate}, \sigma, i, \boldsymbol{v}_i)$, where $\boldsymbol{v}_i \in \mathbb{Z}_q^d$, from some party $i \in [N]$:

1. Assert $\sigma \in [\ell]$ is the identifier of the current execution. If this is NOT the first $(\mathsf{aggregate})$ message from i under the identifier σ, ignore this message; otherwise, record $(\sigma, i, \boldsymbol{v}_i)$, send (σ, i) to the server and the adversary, and wait for the server's input:
   - If the server sends $(\mathsf{ready}, \sigma, \mathcal{U}_3)$ such that (i) $\mathcal{U}_3 \subseteq [N]$ and $|\mathcal{U}_3| \geq t$, and (ii) all $\{(\sigma, i, \boldsymbol{v}_i)\}_{i \in \mathcal{U}_3}$ tuples were recorded, compute $\boldsymbol{a} := \sum_{i \in \mathcal{U}_3} \boldsymbol{v}_i \in \mathbb{Z}_q^d$, send $(\sigma, \boldsymbol{a})$ to the adversary, and wait for the adversary's input for each honest party $i \in \mathcal{U}_3$:
     - If the server is corrupt and the adversary sends $(\mathsf{deliver}, \sigma, i, \boldsymbol{a}')$ for $\boldsymbol{a}' \neq \boldsymbol{a}$, set $\mathsf{Cheat}[\sigma, i] = 1$ and send $(\sigma, \boldsymbol{a}')$ to the party i.
     - If the adversary sends $(\mathsf{deliver}, \sigma, i, \boldsymbol{a})$, set $\mathsf{Cheat}[\sigma, i] = 0$ and send $(\sigma, \boldsymbol{a})$ to the party i.
     - Otherwise, send (abort, i) to the party i and stop sending/receiving messages to/from it.

     After the message delivery, ignore future messages to the σ-th execution and move to the $(\sigma + 1)$-th execution.
   - Otherwise, ignore this message.

Verify: Upon receiving (verify, i) from a party $i \in [N]$, send $(\mathsf{success}, i)$ to the party i if $\forall \sigma \in [\ell] : \mathsf{Cheat}[\sigma, i] = 0$; otherwise, send (abort, i) to the party i.

Abort: Upon receiving (abort, i) from the adversary, send (abort, i) to the party i and stop sending/receiving messages to/from it.

Figure 1: Functionality of secure verifiable aggregation.

3 Protocol

3.1 High-level Overview

The technique behind our malicious-secure patched protocol is "commit-publish-unmask". More specifically, all parties should first commit their seeds used to mask their actual inputs and send their masked inputs as well as the LHH values of these masked inputs. Then, the server publishes the **masked aggregate result** to all parties, and the LHH values are broadcast for the future verification that this masked aggregate result is correctly computed. Since the LHH scheme is applied to masked inputs rather than actual inputs, we get rid of the information leakage in the original VeriFL protocol, where the adversary can see the LHH values dependent on the actual inputs of honest parties. Finally, all parties jointly unmask their received masked aggregate result by opening and using the seeds committed before publishing the masked aggregate result.

Roughly speaking, the commitments of these seeds ensure that the corrupt parties should choose *the sum of their inputs* before seeing the aggregate result, and they cannot change this sum after the aggregate result is implicitly determined in the published masked aggregate result without being detected with overwhelming probability. From the perspective of security proof, the use of commitment allows the simulator to extract the seeds to be used by the corrupt parties *before it opening the commitments of honest parties*. Using these seeds and the published masked aggregate result, the simulator can extract the sum of the corrupt parties' inputs, which is sufficient for the simulation.

3.2 Security Proof

We model the ideal secure verifiable aggregation in the functionality $\mathcal{F}_{\mathsf{SecVerAgg}}$ in Figure 1. For simplicity, we focus on the dropout model where any dropped party will never be online again. We consider the case where the server is corrupt (note that the malicious security for an honest server is easier to prove since the simulator can extract the input of *each* corrupt party and the view of the adversary is much simpler). Our patched VeriFL protocol Π_{VeriFL^+} is given in Figure 2 and Figure 3. The security is stated in the following theorem.

Theorem 1. *For any prime modulus $q \approx 2^\kappa$ and any set C of corrupt parties such that $|C| < \min(t, 2t - N)$, Π_{VeriFL^+} securely realizes $\mathcal{F}_{\mathsf{SecVerAgg}}$ against any static malicious adversary that also corrupts the server.*

Proof. Let $C \subseteq [N] \cup \{\mathsf{server}\}$ denote the set of corrupt parties such that $|C \setminus \{\mathsf{server}\}| \leq t$, and $\mathcal{H} := ([N] \cup \{\mathsf{server}\}) \setminus C$ denote the complement. We focus on server $\in C$. Without loss of generality, we assume the deterministic real-world adversary $\mathcal{A}$ that simply outputs the joint view of corrupt parties and the ideal-world simulator $\mathcal{S}$ that internally runs $\mathcal{A}$. We prove that the ideal execution with respect to $\mathcal{S}$ is computationally indistinguishable from the real execution with respect to $\mathcal{A}$ via the following hybrid argument, which implicitly gives out the construction of $\mathcal{S}$.

- Hybrid_0. This is the real execution.
- Hybrid_1. Same as the previous one, except that all honest parties are played by a simulator that is given the inputs of honest parties, emulates the random oracle H on-the-fly, "interacts" with the real-world adversary $\mathcal{A}$ internally run by itself, and

Protocol Π_{VERIFL^+} (Part 1)

Setup: The security parameter κ, the threshold t, the number N of parties, the batch size ℓ, and the domain $\mathbb{Z}_q^d$. All cryptographic primitives defined in Section 2 and two public parameters: $\mathsf{LHHpp} \leftarrow \mathsf{LHH.Setup}(1^\kappa)$ and $\mathsf{KApp} \leftarrow \mathsf{KA.Setup}(1^\kappa)$. A random oracle $\mathsf{H} : \{0,1\}^\kappa \to \mathbb{Z}_q^d$. Each party $i \in [N]$ has a signing key sigk_i and receives from PKI the other parties' verification keys $\{\mathsf{verk}_j\}_{j\in[N]\setminus\{i\}}$.

Verify: In the σ-th execution, the party $i \in [N]$ has input $\boldsymbol{v}_i^\sigma \in \mathbb{Z}_q^d$ and interacts with the server as follows:

- **Round 0 (Advertising keys)**

  For each party $i \in [N]$:

  1. Generate $(\mathsf{sk}_i^\sigma, \mathsf{pk}_i^\sigma) \leftarrow \mathsf{KA.KeyGen}(1^\kappa)$, $(\mathsf{msk}_i^\sigma, \mathsf{mpk}_i^\sigma) \leftarrow \mathsf{KA.KeyGen}(1^\kappa)$, and $\Sigma_i^{\mathsf{key},\sigma} \leftarrow \mathsf{SIG.Sign}(\mathsf{sigk}_i, (\sigma, \mathsf{pk}_i^\sigma, \mathsf{mpk}_i^\sigma))$.
  2. Send $(\mathsf{pk}_i^\sigma, \mathsf{mpk}_i^\sigma, \Sigma_i^{\mathsf{key},\sigma})$ to the server and move to the next round.

  Server:

  1. Receive messages from at least t parties; otherwise abort. Let $\mathcal{U}_1^\sigma \subseteq [N]$ denote the set of surviving parties.
  2. Send to each party $i \in \mathcal{U}_1^\sigma$ the set $K_i^\sigma := \{(j, \mathsf{pk}_j^\sigma, \mathsf{mpk}_j^\sigma, \Sigma_j^{\mathsf{key},\sigma})\}_{j\in\mathcal{U}_1^\sigma\setminus\{i\}}$ and move to the next round.

- **Round 1 (Sharing metadata)**

  For each party $i \in [N]$:

  1. Receive K_i^σ from the server, deduce the set $\mathcal{U}_1^\sigma$ implicit in K_i^σ, and **assert**

  $$(\mathcal{U}_1^\sigma \subseteq [N]) \wedge (|\mathcal{U}_1^\sigma| \geq t) \wedge (\forall j \in \mathcal{U}_1^\sigma \setminus \{i\} : \mathsf{SIG.Ver}(\mathsf{verk}_j, (\sigma, \mathsf{pk}_j^\sigma, \mathsf{mpk}_j^\sigma), \Sigma_j^{\mathsf{key},\sigma}) = 1).$$

  2. Sample a self-mask seed $b_i^\sigma \leftarrow \{0,1\}^\kappa$.
  3. Generate $\mathsf{com}_i^{\mathsf{seed},\sigma} \leftarrow \mathsf{COM.Commit}^O(b_i^\sigma, r_i^\sigma)$ and $\mathsf{com}_i^{\mathsf{msk},\sigma} \leftarrow \mathsf{COM.Commit}^O(\mathsf{msk}_i^\sigma, s_i^\sigma)$, where $r_i^\sigma, s_i^\sigma \leftarrow \{0,1\}^\kappa$.
  4. Generate t-out-of-$|\mathcal{U}_1^\sigma|$ shares of (b_i^σ, r_i^σ): $\{\langle(b_i^\sigma, r_i^\sigma)\rangle_j\}_{j\in\mathcal{U}_1^\sigma} \leftarrow \mathsf{SS.Share}((b_i^\sigma, r_i^\sigma), \mathcal{U}_1^\sigma)$.
  5. Generate t-out-of-$|\mathcal{U}_1^\sigma|$ shares of $(\mathsf{msk}_i^\sigma, s_i^\sigma)$: $\{\langle(\mathsf{msk}_i^\sigma, s_i^\sigma)\rangle_j\}_{j\in\mathcal{U}_1^\sigma} \leftarrow \mathsf{SS.Share}((\mathsf{msk}_i^\sigma, s_i^\sigma), \mathcal{U}_1^\sigma)$.
  6. For $j \in \mathcal{U}_1^\sigma \setminus \{i\}$, compute $k_{i,j}^\sigma \leftarrow \mathsf{KA.Agree}(\mathsf{sk}_i^\sigma, \mathsf{pk}_j^\sigma)$ and ciphertext $c_{i,j}^\sigma \leftarrow \mathsf{SE.Enc}(k_{i,j}^\sigma, (\langle(b_i^\sigma, r_i^\sigma)\rangle_j, \langle(\mathsf{msk}_i^\sigma, s_i^\sigma)\rangle_j))$.
  7. Generate $\Sigma_i^{\mathsf{com},\sigma} \leftarrow \mathsf{SIG.Sign}(\mathsf{sigk}_i, (\sigma, \mathsf{com}_i^{\mathsf{seed},\sigma}, \mathsf{com}_i^{\mathsf{msk},\sigma}))$.
  8. Send $(\{(j, c_{i,j}^\sigma)\}_{j\in\mathcal{U}_1^\sigma\setminus\{i\}}, \mathsf{com}_i^{\mathsf{seed},\sigma}, \mathsf{com}_i^{\mathsf{msk},\sigma}, \Sigma_i^{\mathsf{com},\sigma})$ to the server and move to the next round.

  Server:

  1. Receive messages from at least t parties; otherwise abort. Let $\mathcal{U}_2^\sigma \subseteq \mathcal{U}_1^\sigma$ denote the set of surviving parties.
  2. Send to each party $i \in \mathcal{U}_2^\sigma$ the set $C_i^\sigma := \{(j, c_{j,i}^\sigma, \mathsf{com}_j^{\mathsf{seed},\sigma}, \mathsf{com}_j^{\mathsf{msk},\sigma}, \Sigma_j^{\mathsf{com},\sigma})\}_{j\in\mathcal{U}_2^\sigma\setminus\{i\}}$ and move to the next round.

- **Round 2 (Publishing hash & masked aggregate result)**

  For each party $i \in [N]$:

  1. Receive C_i^σ from the server, deduce the set $\mathcal{U}_2^\sigma$ implicit in C_i^σ, and **assert**

  $$(\mathcal{U}_2^\sigma \subseteq \mathcal{U}_1^\sigma) \wedge (|\mathcal{U}_2^\sigma| \geq t) \wedge (\forall j \in \mathcal{U}_2^\sigma \setminus \{i\} : \mathsf{SIG.Ver}(\mathsf{verk}_j, (\sigma, \mathsf{com}_j^{\mathsf{seed},\sigma}, \mathsf{com}_j^{\mathsf{msk},\sigma}), \Sigma_j^{\mathsf{com},\sigma}) = 1).$$

  2. For $j \in \mathcal{U}_2^\sigma \setminus \{i\}$, compute $\mathsf{mak}_{i,j}^\sigma \leftarrow \mathsf{KA.Agree}(\mathsf{msk}_i^\sigma, \mathsf{mpk}_j^\sigma)$.
  3. Generate the masked input $\boldsymbol{p}_i^\sigma := \boldsymbol{v}_i^\sigma + \mathsf{H}(b_i^\sigma) + \sum_{j\in\mathcal{U}_2^\sigma} \Delta_{i,j} \cdot \mathsf{H}(\mathsf{mak}_{i,j}^\sigma) \in \mathbb{Z}_q^d$, where $\Delta_{i,j} = 1$ if $i < j$, $\Delta_{i,j} = -1$ if $i > j$, and $\Delta_{i,j} = 0$ if $i = j$, the linearly homomorphic hash $h_i^\sigma \leftarrow \mathsf{LHH.Hash}(\boldsymbol{p}_i^\sigma)$, and $\Sigma_i^{\mathsf{input},\sigma} \leftarrow \mathsf{SIG.Sign}(\mathsf{sigk}_i, (\sigma, h_i^\sigma))$.
  4. Send $(\boldsymbol{p}_i^\sigma, h_i^\sigma, \Sigma_i^{\mathsf{input},\sigma})$ to the server and move to the next round.

  Server:

  1. Receive messages from at least t parties; otherwise abort. Let $\mathcal{U}_3^\sigma \subseteq \mathcal{U}_2^\sigma$ denote the set of surviving parties.
  2. Send to each party $i \in \mathcal{U}_3^\sigma$ the transcript $H_i^\sigma := (\{(j, h_j^\sigma, \Sigma_j^{\mathsf{input},\sigma})\}_{j\in\mathcal{U}_3^\sigma\setminus\{i\}}, \boldsymbol{p}^\sigma)$, where $\boldsymbol{p}^\sigma := \sum_{j\in\mathcal{U}_3^\sigma} \boldsymbol{p}_j^\sigma$, and move to the next round.

Figure 2: Patched VERIFL protocol. If any **assertion** fails, a party will abort without output.

outputs whatever $\mathcal{A}$ outputs. This hybrid is syntactically the same as the real execution, and the joint output distributions in the two hybrids are identical.

- Hybrid_2. Same as the previous one, except that the simulator additionally aborts if there exist two honest parties $i, j \in \mathcal{H}$ such that (i) K_i^σ contains different $(\mathsf{pk}_j^\sigma, \mathsf{mpk}_j^\sigma)$ from that used by the party j in the σ-th **Round 0**, and (ii) the party i's

Protocol Π_{VERIFL^+} (Part 2)

- **Round 3 (Checking consistency)**
  For each party $i \in [N]$:
  1. Receive H_i^σ from the server, deduce the set $\mathcal{U}_3^\sigma$ implicit in H_i^σ, and **assert**
  $$(\mathcal{U}_3^\sigma \subseteq \mathcal{U}_2^\sigma) \wedge (|\mathcal{U}_3^\sigma| \geq t) \wedge (\forall j \in \mathcal{U}_3^\sigma \setminus \{i\} : \mathsf{SIG.Ver}(\mathsf{verk}_j, (\sigma, h_j^\sigma), \Sigma_j^{\mathsf{input},\sigma}) = 1).$$
  2. Define sync message $\mathsf{syncMsg}_i^\sigma := (\{(j, h_j^\sigma, \mathsf{com}_j^{\mathsf{seed},\sigma}, \mathsf{mpk}_j^\sigma)\}_{j \in \mathcal{U}_3^\sigma}, \{(j, \mathsf{com}_j^{\mathsf{msk},\sigma})\}_{j \in \mathcal{U}_2^\sigma \setminus \mathcal{U}_3^\sigma})$.
  3. Send $\Sigma_i^{\mathsf{sync},\sigma} \leftarrow \mathsf{SIG.Sign}(\mathsf{sigk}_i, (\sigma, \mathsf{syncMsg}_i^\sigma))$ to the server and move to the next round.

  Server:
  1. Receive messages from at least t parties; otherwise abort. Let $\mathcal{U}_4^\sigma \subseteq \mathcal{U}_3^\sigma$ denote the set of surviving parties.
  2. Send to each party $i \in \mathcal{U}_4^\sigma$ the set $S_i^\sigma := \{(j, \Sigma_j^{\mathsf{sync},\sigma})\}_{j \in \mathcal{U}_4^\sigma \setminus \{i\}}$ and move to the next round.
- **Round 4 (Opening commitments)**
  For each party $i \in [N]$:
  1. Receive S_i^σ from the server, deduce the set $\mathcal{U}_4^\sigma$ implicit in S_i^σ, and **assert**
  $$(\mathcal{U}_4^\sigma \subseteq \mathcal{U}_3^\sigma) \wedge (|\mathcal{U}_4^\sigma| \geq t) \wedge (\forall j \in \mathcal{U}_4^\sigma \setminus \{i\} : \mathsf{SIG.Ver}(\mathsf{verk}_j, (\sigma, \mathsf{syncMsg}_i^\sigma), \Sigma_j^{\mathsf{sync},\sigma}) = 1).$$
  2. For $j \in \mathcal{U}_2^\sigma \setminus \{i\}$, compute $(\langle (b_j^\sigma, r_j^\sigma) \rangle_i, \langle (\mathsf{msk}_j^\sigma, s_j^\sigma) \rangle_i) \leftarrow \mathsf{SE.Dec}(k_{i,j}^\sigma, c_{j,i}^\sigma)$.
  3. Send $((b_i^\sigma, r_i^\sigma), \{(j, \langle (b_j^\sigma, r_j^\sigma) \rangle_i)\}_{j \in \mathcal{U}_3^\sigma}, \{(j, \langle (\mathsf{msk}_j^\sigma, s_j^\sigma) \rangle_i)\}_{j \in \mathcal{U}_2^\sigma \setminus \mathcal{U}_3^\sigma})$ to the server and move to the next round.

  Server:
  1. Receive messages from at least t parties; otherwise abort. Let $\mathcal{U}_5^\sigma \subseteq \mathcal{U}_4^\sigma$ denote the set of surviving parties.
  2. For $i \in \mathcal{U}_3^\sigma \setminus \mathcal{U}_5^\sigma$, reconstruct $(b_i^\sigma, r_i^\sigma) \leftarrow \mathsf{SS.Combine}(\{\langle (b_i^\sigma, r_i^\sigma) \rangle_j\}_{j \in \mathcal{U}_5^\sigma}, \mathcal{U}_5^\sigma)$.
  3. For $i \in \mathcal{U}_2^\sigma \setminus \mathcal{U}_3^\sigma$, reconstruct $(\mathsf{msk}_i^\sigma, s_i^\sigma) \leftarrow \mathsf{SS.Combine}(\{\langle (\mathsf{msk}_i^\sigma, s_i^\sigma) \rangle_j\}_{j \in \mathcal{U}_5^\sigma}, \mathcal{U}_5^\sigma)$.
  4. Send to each party $i \in \mathcal{U}_5^\sigma$ the transcript $T_i^\sigma := (\{(j, b_j^\sigma, r_j^\sigma)\}_{j \in \mathcal{U}_3^\sigma}, \{(j, \mathsf{msk}_j^\sigma, s_j^\sigma)\}_{j \in \mathcal{U}_2^\sigma \setminus \mathcal{U}_3^\sigma})$.
- **Round 5 (Unmasking)**
  For each party $i \in [N]$:
  1. Receive T_i^σ from the server and **assert** that "the two sets in T_i^σ are indexed by $\mathcal{U}_3^\sigma$ and $\mathcal{U}_2^\sigma \setminus \mathcal{U}_3^\sigma$, respectively" and
  $$(\forall j \in \mathcal{U}_3^\sigma \setminus \{i\} : \mathsf{COM.Open}^O(\mathsf{com}_j^{\mathsf{seed},\sigma}, b_j^\sigma, r_j^\sigma) = 1) \wedge (\forall j \in \mathcal{U}_2^\sigma \setminus \mathcal{U}_3^\sigma : \mathsf{COM.Open}^O(\mathsf{com}_j^{\mathsf{msk},\sigma}, \mathsf{msk}_j^\sigma, s_j^\sigma) = 1).$$
  2. For $j \in \mathcal{U}_3^\sigma$ and $k \in \mathcal{U}_2^\sigma \setminus \mathcal{U}_3^\sigma$, compute $\mathsf{mak}_{j,k}^\sigma \leftarrow \mathsf{KA.Agree}(\mathsf{msk}_k^\sigma, \mathsf{mpk}_j^\sigma)$.
  3. Compute the aggregate result $\boldsymbol{a}^\sigma := \boldsymbol{p}^\sigma - \sum_{j \in \mathcal{U}_3^\sigma} \mathsf{H}(b_j^\sigma) - \sum_{j \in \mathcal{U}_3^\sigma, k \in \mathcal{U}_2^\sigma \setminus \mathcal{U}_3^\sigma} \Delta_{j,k} \cdot \mathsf{H}(\mathsf{mak}_{j,k}^\sigma)$ and output $(\sigma, \boldsymbol{a}^\sigma)$.

Verify: The party $i \in [N]$ locally proceeds as follows:

1. For $\sigma \in [\ell]$, compute $h^\sigma := \sum_{j \in \mathcal{U}_3^\sigma} h_j^\sigma$ and sample a random coefficient $\alpha_\sigma \leftarrow \mathbb{Z}_q$ (note that $\mathbb{Z}_q$ is included in LHHpp).
2. Compute $h_{\boldsymbol{p}} \leftarrow \mathsf{LHH.Hash}(\sum_{\sigma \in [\ell]} \alpha_\sigma \cdot \boldsymbol{p}^\sigma)$ and $h'_{\boldsymbol{p}} := \sum_{\sigma \in [\ell]} \alpha_\sigma \cdot h^\sigma$.
3. **Assert** $h_{\boldsymbol{p}} = h'_{\boldsymbol{p}}$ and output $(\mathsf{success}, i)$.

Figure 3: Patched VERIFL protocol. If any **assertion** fails, a party will abort without output.

signature verification for $(\sigma, \mathsf{pk}_j^\sigma, \mathsf{mpk}_j^\sigma)$ passes in the σ-th **Round 1**.

Any adversary $\mathcal{A}$ that can trigger this abort with non-negligible probability implies an efficient attacker against the security of the digital signature scheme. As a failure event, the simulator aborts only with negligible probability. Using the difference lemma, the joint output distributions in the two hybrids are computationally indistinguishable.

- Hybrid_3. Same as the previous one, except that, for every two honest parties $i, j \in \mathcal{H}$, the simulator replaces the agreed key $k_{i,j}^\sigma$ with a uniformly random key in the σ-th **Round 1**.

  Conditioned on that the simulator does not abort, $k_{i,j}^\sigma$ is identical between the two parties in the previous hybrid due to the correctness of the key agreement scheme. Under this condition, any adversary $\mathcal{A}$ that can distinguish the two hybrids with non-negligible probability implies an efficient attacker against the security of the key agreement scheme. Using the law of

total probability, the joint output distributions in the two hybrids are computationally indistinguishable[1].

- Hybrid_4. Same as the previous one, except that, for every two honest parties $i, j \in \mathcal{H}$, the simulator uses an equivalent way to decrypt the ciphertext $c^{\sigma}_{j,i}$ on behalf of the party i in the σ-th **Round 4**: if $c^{\sigma}_{j,i}$ equals that generated by the simulator in the σ-th **Round 1**, the simulator retrieves $(\langle (b^{\sigma}_j, r^{\sigma}_j) \rangle_i, \langle (\mathsf{msk}^{\sigma}_j, s^{\sigma}_j) \rangle_i)$ from its memory without actually performing decryption; otherwise, the simulator honestly decrypts $c^{\sigma}_{j,i}$ as per the previous hybrid.

  This hybrid is syntactically the same as the previous one due to the correctness of the symmetric encryption scheme.

- Hybrid_5. Same as the previous one, except that, for every two honest parties $i, j \in \mathcal{H}$, the simulator replaces the ciphertext $c^{\sigma}_{j,i}$ with an encryption of 0 in the σ-th **Round 1**.

  From the view of $\mathcal{A}$, the only difference between the two hybrids is the distribution of $c^{\sigma}_{j,i}$. Any adversary $\mathcal{A}$ that can distinguish the two hybrids with non-negligible probability implies an efficient attacker against the IND-CCA security of the symmetric encryption scheme. The joint output distributions in the two hybrids are computationally indistinguishable.

- Hybrid_6. Same as the previous one, except that, for every two honest parties $i, j \in \mathcal{H}$, the simulator uses an equivalent way to define the share $(\langle (b^{\sigma}_j, r^{\sigma}_j) \rangle_i, \langle (\mathsf{msk}^{\sigma}_j, s^{\sigma}_j) \rangle_i)$ in the σ-th **Round 1**: instead of directly generating this share, the simulator "lazily samples" it by running $\mathsf{SS.RSample}$ on $((b^{\sigma}_j, r^{\sigma}_j), (\mathsf{msk}^{\sigma}_j, s^{\sigma}_j))$ and the shares given to all corrupt parties.

  This hybrid is syntactically the same as the previous one due to the reverse samplability of the secret sharing scheme.

- Hybrid_7. Same as the previous one, except that, for every honest party $j \in \mathcal{H}$ and corrupt party $i \in C$, the simulator replaces the ciphertext $c^{\sigma}_{j,i}$ with an encryption of random shares in the σ-th **Round 1**.

  From the view of $\mathcal{A}$, the only difference between the two hybrids is the distribution of the shares given to the corrupt parties. Since $|C| < t$, any adversary $\mathcal{A}$ that can distinguish the two hybrids with non-negligible probability implies an efficient attacker against the security of the secret sharing scheme. The joint output distributions in the two hybrids are computationally indistinguishable.

- Hybrid_8. Same as the previous one, except that, for every honest party $i \in \mathcal{H}$, the simulator (i) generates

$$\mathsf{com}^{\mathsf{msk},\sigma}_i \leftarrow \mathsf{COM.Commit}^{\mathcal{O}}(0, s^{\sigma}_i)$$

  in the σ-th **Round 1**, and (ii) replaces the lazy sampling for $(\mathsf{msk}^{\sigma}_i, s^{\sigma}_i)$ with that for $(\mathsf{msk}^{\sigma}_i, s'^{\sigma}_i)$, where

$$s'^{\sigma}_i \leftarrow \mathsf{COM.Equiv}^{\mathcal{O}}(\mathsf{com}^{\mathsf{msk},\sigma}_i, (0, s^{\sigma}_i), \mathsf{msk}^{\sigma}_i).$$

  The simulator additionally aborts if the equivocation algorithm fails.

  The equivocality of the commitment scheme ensures that the simulator can obtain s'^{σ}_i with overwhelming probability, i.e., the probability that the simulator aborts only increases by a negligible value. Conditioned on the successful equivocation, from the view of $\mathcal{A}$, the only difference between the two hybrids is the distribution of the commitment $\mathsf{com}^{\mathsf{msk},\sigma}_i$. Under this condition, any adversary $\mathcal{A}$ that can distinguish two hybrids with non-negligible probability implies an efficient attacker against the hiding property of the commitment scheme. Using the law of total probability, the joint output distributions in the two hybrids are computationally indistinguishable.

- Hybrid_9. Same as the previous one, except that the simulator additionally aborts if there exist two honest parties $i, j \in \mathcal{H}$ such that (i) C^{σ}_i contains different $(\mathsf{com}^{\mathsf{seed},\sigma}_j, \mathsf{com}^{\mathsf{msk},\sigma}_j)$ from that used by the party j in the σ-th **Round 1**, and (ii) the party i's signature verification for $(\sigma, \mathsf{com}^{\mathsf{seed},\sigma}_j, \mathsf{com}^{\mathsf{msk},\sigma}_j)$ passes in the σ-th **Round 2**.

  Any adversary $\mathcal{A}$ that can trigger this abort with non-negligible probability implies an efficient attacker against the security of the digital signature scheme. Using the difference lemma, the joint output distributions in the two hybrids are computationally indistinguishable.

- Hybrid_{10}. Same as the previous one, except that the simulator additionally aborts if there exist two honest parties $i, j \in \mathcal{H}$ such that (i) H^{σ}_i contains different h^{σ}_j from that used by the party j in the σ-th **Round 2**, and (ii) the party i's signature verification for (σ, h^{σ}_j) passes in the σ-th **Round 3**.

  Any adversary $\mathcal{A}$ that can trigger this abort with non-negligible probability implies an efficient attacker against the security of the digital signature scheme. Using the difference lemma, the joint output distributions in the two hybrids are computationally indistinguishable.

[1] Note that the transition in this hybrid is independent of the probability that the simulator aborts due to *the conditions in the previous hybrid* so that the established indistinguishability between any two previous adjacent hybrids will not be skewed by the transition. As a result, the simulator still aborts in this hybrid with the *same* negligible probability due to these conditions. This requirement simplifies the probability analysis, and it is also maintained in the subsequent hybrids by carefully arranging their orders.

- Hybrid_{11}. Same as the previous one, except that the simulator additionally aborts if there exist two honest parties $i, j \in \mathcal{H}$ such that (i) $\mathsf{syncMsg}_i^\sigma \neq \mathsf{syncMsg}_j^\sigma$ in the σ-th **Round 3**, and (ii) the party i's signature verification for $(\sigma, \mathsf{syncMsg}_i^\sigma)$ passes in the σ-th **Round 4**.

  Any adversary $\mathcal{A}$ that can trigger this abort with non-negligible probability implies an efficient attacker against the security of the digital signature scheme. Using the difference lemma, the joint output distributions in the two hybrids are computationally indistinguishable.

  Note that the sync message $\mathsf{syncMsg}_i^\sigma$ is locally defined by the party i. For any two distinct honest parties $i, j \in \mathcal{H}$ such that $\mathsf{syncMsg}_i^\sigma \neq \mathsf{syncMsg}_j^\sigma$, they cannot both survive the assertion in the σ-th **Round 4** conditioned on that the simulator does not abort. Otherwise, there must be at least $t - |C|$ distinct honest parties are required to sign each sync message, contradicting the assumption $2(t - |C|) > N - |C|$.

  That is, for the honest parties that do not abort and need to send messages to the server in the σ-th **Round 4**, their sync messages must be identical if the simulator does not abort. The transcripts in this identical sync message, which is denoted by $\mathsf{syncMsg}^\sigma$, are authenticated so that $\mathcal{A}$ cannot forge the entries of the honest parties. From $\mathsf{syncMsg}^\sigma$, the simulator can define $\mathcal{U}_2^\sigma$ and $\mathcal{U}_3^\sigma$ that are consistent for the surviving honest parties *before* sending any message to $\mathcal{A}$ in this round.

- Hybrid_{12}. Same as the previous one, except that the simulator additionally aborts if (i) there exists some honest party $i \in \mathcal{U}_3^\sigma \cap \mathcal{H}$ such that $\mathcal{A}$ queries $\mathsf{H}(b_i^\sigma)$ before the simulator sends any message in the σ-th **Round 4**, or (ii) there exists some honest party $i \in \mathcal{H} \setminus \mathcal{U}_3^\sigma$ such that $\mathcal{A}$ queries $\mathsf{H}(b_i^\sigma)$.

  In these two hybrids, only the commitment $\mathsf{com}_i^{\mathsf{seed},\sigma}$ depends on the concrete value of b_i^σ. Any adversary $\mathcal{A}$ can trigger this abort with non-negligible probability must be able to guess the committed b_i^σ with non-negligible probability before seeing the valid opening string. Such an adversary implies an efficient attacker against the hiding property of the commitment scheme. Using the difference lemma, the joint output distributions in the two hybrids are computationally indistinguishable.

- Hybrid_{13}. Same as the previous one, except that the simulator additionally aborts if there exist two distinct honest parties $i, j \in \mathcal{U}_3^\sigma \cap \mathcal{H}$ such that $\mathcal{A}$ queries $\mathsf{H}(\mathsf{mak}_{i,j}^\sigma)$.

  Conditioned on that the simulator does not abort, $\mathsf{mak}_{i,j}^\sigma = \mathsf{KA.Agree}(\mathsf{msk}_i^\sigma, \mathsf{mpk}_j^\sigma) = \mathsf{KA.Agree}(\mathsf{msk}_j^\sigma, \mathsf{mpk}_i^\sigma)$ is well-defined for the two parties. Also, only the agreed key $\mathsf{mak}_{i,j}^\sigma$ depends on the concrete values of msk_i^σ and msk_j^σ in these two hybrids. Any adversary $\mathcal{A}$ can trigger this abort with non-negligible probability must be able to guess $\mathsf{mak}_{i,j}^\sigma$ with non-negligible probability, i.e., implying an efficient attacker against the security of the key agreement scheme. Using the difference lemma, the joint output distributions in the two hybrids are computationally indistinguishable.

- Hybrid_{14}. Same as the previous one, except that the simulator (i) sets a random $\boldsymbol{p}_i^\sigma \leftarrow \mathbb{Z}_q^d$ for every honest party $i \in \mathcal{H}$ in the σ-th **Round 2**, and (ii) programs $\mathsf{H}(b_i^\sigma)$ for every honest party $i \in \mathcal{U}_3^\sigma \cap \mathcal{H}$ before sending any message to $\mathcal{A}$ in the σ-th **Round 4** such that the following consistency holds:

$$\mathsf{H}(b_i^\sigma) = \boldsymbol{p}_i^\sigma - \boldsymbol{v}_i^\sigma - \sum_{j \in \mathcal{U}_2^\sigma} \Delta_{i,j} \cdot \mathsf{H}(\mathsf{mak}_{i,j}^\sigma).$$

  Conditioned on that the simulator does not abort, we claim that the joint output distributions in the two hybrids are identical. On the one hand, for $i \in \mathcal{U}_3^\sigma \cap \mathcal{H}$, $\boldsymbol{p}_i^\sigma$ is uniform from the view of $\mathcal{A}$ until receiving messages from the simulator in the σ-th **Round 4** due to (i) the never queried $\mathsf{H}(b_i^\sigma)$ in the previous hybrid, or (ii) the uniform sampling in this hybrid. Since $\mathsf{H}(b_i^\sigma)$ was not queried by $\mathcal{A}$ before the simulator sends any message in the σ-th **Round 4**, the programming must succeed so that, when the opening string is reconstructed, the consistency holds in both two hybrids from the view of $\mathcal{A}$. On the other hand, for $i \in \mathcal{H} \setminus \mathcal{U}_3^\sigma$, no programming is performed to maintain the consistency. However, since $\mathcal{A}$ never queries $\mathsf{H}(b_i^\sigma)$, this inconsistency cannot be observed by $\mathcal{A}$, and the resulting $\boldsymbol{p}_i^\sigma$ in the previous hybrid is as uniform as that in this hybrid.

  It is worth noting that the identical distribution of $\boldsymbol{p}_i^\sigma$, which is a part of the view of $\mathcal{A}$ and affects the abort probability of the simulator, ensures that the transition in this hybrid does not skew the previous computational indistinguishability. Using the law of total probability, the joint output distributions in the two hybrids are identical.

- Hybrid_{15}. Same as the previous one, except that, for every honest party $i \in \mathcal{U}_3^\sigma \cap \mathcal{H}$, the simulator instead programs

$$\mathsf{H}(b_i^\sigma) = \boldsymbol{p}_i^\sigma - \mathsf{sum}_{\mathsf{honest},i}^\sigma - \sum_{j \in \mathcal{U}_2^\sigma \setminus (\mathcal{U}_3^\sigma \cap \mathcal{H})} \Delta_{i,j} \cdot \mathsf{H}(\mathsf{mak}_{i,j}^\sigma),$$

  where $\{\mathsf{sum}_{\mathsf{honest},i}^\sigma\}_{i \in \mathcal{U}_3^\sigma \cap \mathcal{H}}$ are uniform conditioned on $\sum_{i \in \mathcal{U}_3^\sigma \cap \mathcal{H}} \mathsf{sum}_{\mathsf{honest},i}^\sigma = \sum_{i \in \mathcal{U}_3^\sigma \cap \mathcal{H}} \boldsymbol{v}_i^\sigma$.

  Conditioned on that the simulator does not abort, the joint output distributions in the two hybrids are identical. We define

$$\mathsf{sum}_{\mathsf{honest},i}^\sigma := \boldsymbol{v}_i^\sigma + \sum_{j \in \mathcal{U}_3^\sigma \cap \mathcal{H}} \Delta_{i,j} \cdot \mathsf{H}(\mathsf{mak}_{i,j}^\sigma),$$

which is uniform from the view of $\mathcal{A}$ since it never queries $\mathsf{H}(\mathsf{mak}^{\sigma}_{i,j})$ for any $j \neq i$ and $j \in \mathcal{U}_3^{\sigma} \cap \mathcal{H}$ under the condition. The above definition satisfies the condition on the sum given that $\sum_{i,j \in \mathcal{U}_3^{\sigma} \cap \mathcal{H}} \Delta_{i,j} \cdot \mathsf{H}(\mathsf{mak}^{\sigma}_{i,j}) = \mathbf{0}$. Using the law of total probability, the joint output distributions in the two hybrids are identical.

- Hybrid_{16}. Same as the previous one, except that the simulator additionally aborts if (i) there exist some honest party $i \in \mathcal{H}$ and some corrupt party $j \in \mathcal{U}_3^{\sigma} \cap \mathcal{C}$ (or, $j \in (\mathcal{U}_2^{\sigma} \setminus \mathcal{U}_3^{\sigma}) \cap \mathcal{C}$) such that T_i^{σ} contains different b_j^{σ} (or, msk_j^{σ}) from that committed by the party j in the σ-th **Round 1**, and (ii) the party i does not abort in the σ-th **Round 5**.

  Any adversary $\mathcal{A}$ that can trigger this abort with non-negligible probability implies an efficient attacker against the binding property of the commitment scheme. Using the difference lemma, the joint output distributions in the two hybrids are computationally indistinguishable.

- Hybrid_{17}. Same as the previous one, except that the simulator additionally aborts if (i) there exist two honest parties $i \in \mathcal{H}$ and $j \in (\mathcal{U}_3^{\sigma} \cap \mathcal{H}) \setminus \{i\}$ (or, $j \in (\mathcal{U}_2^{\sigma} \setminus \mathcal{U}_3^{\sigma}) \cap \mathcal{H}$) such that T_i^{σ} contains different b_j^{σ} (or, msk_j^{σ}) from that chosen by the simulator on behalf of the party j in the σ-th **Round 1**, and (ii) the party i does not abort in the σ-th **Round 5**.

  Any adversary $\mathcal{A}$ that can trigger this abort with non-negligible probability implies an efficient attacker against the binding property of the commitment scheme[2]. Using the difference lemma, the joint output distributions in the two hybrids are computationally indistinguishable.

- Hybrid_{18}. Same as the previous one, except that the simulator additionally aborts if (i) there exists some corrupt party $j \in \mathcal{U}_2^{\sigma} \cap \mathcal{C}$ such that syncMsg contains a *malformed* com_j^{σ}, which is not obtained from $\mathcal{A}$'s invocation of $\mathsf{COM.Commit}^{O}(\cdot,\cdot)$, and (ii) all honest parties do not abort in the σ-th **Round 5**.

  By Lemma 1, the probability that the simulator aborts due to this condition is negligible (i.e., at most the negligible probability in the lemma times a polynomial number of the opening strings of the corrupt parties' commitments). Using the difference lemma, the joint output distributions in the two hybrids are computationally indistinguishable.

- Hybrid_{19}. Same as the previous one, except that the simulator is no longer given the inputs of honest parties and interacts with the functionality $\mathcal{F}_{\mathsf{SecVerAgg}}$ to complete the simulation. More specifically,

  - Whenever some honest party i played by the simulator aborts, the simulator sends (abort, i) to $\mathcal{F}_{\mathsf{SecVerAgg}}$.
  - Note that syncMsg is identical for all parties in $\mathcal{H}^{\sigma}_{\mathsf{survive}} \subseteq \mathcal{U}_3^{\sigma} \cap \mathcal{H}$, the set of the honest parties that are still played by the simulator just *before* sending any message to $\mathcal{A}$ in the σ-th **Round 4**. At this time, the simulator learns $\{h_i^{\sigma}\}_{i \in \mathcal{U}_3^{\sigma}}$ from syncMsg and checks, for each honest party $i \in \mathcal{H}^{\sigma}_{\mathsf{survive}}$, the consistency

    $$\sum_{i \in \mathcal{U}_3^{\sigma}} h_i^{\sigma} = \mathsf{LHH.Hash}(\boldsymbol{p}^{\sigma,i}),$$

    where $\boldsymbol{p}^{\sigma,i}$ is the the masked aggregate result in H_i^{σ} received by the party i in the σ-th **Round 3**. Let $\mathcal{H}^{\sigma}_{\mathsf{pass}} \subseteq \mathcal{H}^{\sigma}_{\mathsf{survive}}$ denote the set of the honest parties whose $\boldsymbol{p}^{\sigma,i}$ pass this consistency check. There are three cases for each $\sigma \in [\ell]$:

    1. **Case (i): $\mathcal{H}^{\sigma}_{\mathsf{pass}} = \varnothing$.** After $\mathcal{H}^{\sigma}_{\mathsf{survive}}$ and $\mathcal{H}^{\sigma}_{\mathsf{pass}}$ are determined, the simulator sends $(\mathsf{aggregate}, \sigma, j, \widetilde{\boldsymbol{v}}_j)$ to $\mathcal{F}_{\mathsf{SecVerAgg}}$ on behalf of each corrupt party $j \in \mathcal{U}_3^{\sigma} \cap \mathcal{C}$, where $\widetilde{\boldsymbol{v}}_j$ is uniform. Then, the simulator sends $(\mathsf{ready}, \sigma, \mathcal{U}_3^{\sigma})$ to $\mathcal{F}_{\mathsf{SecVerAgg}}$ on behalf of the corrupt server, receives back $(\sigma, \boldsymbol{a})$, and programs H in the σ-th **Round 4** using

       $$\sum_{i \in \mathcal{U}_3^{\sigma} \cap \mathcal{H}} \boldsymbol{v}_i^{\sigma} := \boldsymbol{a} - \sum_{j \in \mathcal{U}_3^{\sigma} \cap \mathcal{C}} \widetilde{\boldsymbol{v}}_j.$$

       Finally, for each honest party $i \in \mathcal{H}^{\sigma}_{\mathsf{out}} \subseteq \mathcal{H}^{\sigma}_{\mathsf{survive}}$, where $\mathcal{H}^{\sigma}_{\mathsf{out}}$ is the set of the honest parties surviving the assertion in the σ-th **Round 5**, the simulator sends $(\mathsf{deliver}, \sigma, i, \boldsymbol{a}^{\sigma,i})$ to $\mathcal{F}_{\mathsf{SecVerAgg}}$, where

       $$\boldsymbol{a}^{\sigma,i} := \boldsymbol{p}^{\sigma,i} - \sum_{j \in \mathcal{U}_3^{\sigma}} \mathsf{H}(b_j^{\sigma}) - \sum_{j \in \mathcal{U}_3^{\sigma}, k \in \mathcal{U}_2^{\sigma} \setminus \mathcal{U}_3^{\sigma}} \Delta_{j,k} \cdot \mathsf{H}(\mathsf{KA.Agree}(\mathsf{msk}_k^{\sigma}, \mathsf{mpk}_j^{\sigma}))$$

       and $(\{b_j^{\sigma}\}_{j \in \mathcal{U}_3^{\sigma}}, \{\mathsf{msk}_k^{\sigma}\}_{k \in \mathcal{U}_2^{\sigma} \setminus \mathcal{U}_3^{\sigma}})$ is included in T_i^{σ}. Other honest parties in $(\mathcal{U}_3^{\sigma} \cap \mathcal{H}) \setminus \mathcal{H}^{\sigma}_{\mathsf{out}}$ aborts.
    2. **Case (ii): $\exists i, j \in \mathcal{H}^{\sigma}_{\mathsf{pass}} : \boldsymbol{p}^{\sigma,i} \neq \boldsymbol{p}^{\sigma,j}$.** The simulator aborts in this case.
    3. **Case (iii): $(\mathcal{H}^{\sigma}_{\mathsf{pass}} \neq \varnothing) \wedge (\forall i, j \in \mathcal{H}^{\sigma}_{\mathsf{pass}} : \boldsymbol{p}^{\sigma,i} = \boldsymbol{p}^{\sigma,j})$.** Let $\boldsymbol{p}^{\sigma}$ denote this identical $\boldsymbol{p}^{\sigma,i}$ for $i \in \mathcal{H}^{\sigma}_{\mathsf{pass}}$. The simulator is identical to that in **Case (i)** except that it defines $\widetilde{\boldsymbol{v}}_j$ for each corrupt party $j \in \mathcal{U}_3^{\sigma} \cap \mathcal{C}$ as follows:
       First, the simulator runs the straight-line extractor $\mathsf{COM.Extract}^{O}$ on the corrupt parties' commitments

       $$\{\mathsf{com}_j^{\mathsf{seed},\sigma}\}_{j \in \mathcal{U}_3^{\sigma} \cap \mathcal{C}}, \{\mathsf{com}_j^{\mathsf{msk},\sigma}\}_{j \in (\mathcal{U}_2^{\sigma} \setminus \mathcal{U}_3^{\sigma}) \cap \mathcal{C}}$$

[2] The reduction uses the fact that, before sending T_i^{σ} to the party i, $\mathcal{A}$ is given the commitment of the honest party $j \neq i$ and its opening string.

to extract the committed values in these commitments, i.e., $\{b_j^\sigma\}_{j\in\mathcal{U}_3^\sigma\cap C}$ and $\{\mathsf{msk}_j^\sigma\}_{j\in(\mathcal{U}_2^\sigma\setminus\mathcal{U}_3^\sigma)\cap C}$. If the extractor fails, the simulator immediately aborts all honest parties still played by it by sending $(\mathsf{abort},\cdot)$ to $\mathcal{F}_{\mathsf{SecVerAgg}}$ and terminates. Then, the simulator extracts the sum of the inputs of the corrupt parties in $\mathcal{U}_3^\sigma\cap C$:

$$\sum_{j\in\mathcal{U}_3^\sigma\cap C}\boldsymbol{v}_j^\sigma := \left(\boldsymbol{p}^\sigma - \sum_{j\in\mathcal{U}_3^\sigma\cap\mathcal{H}}\boldsymbol{p}_j^\sigma - \sum_{j\in\mathcal{U}_3^\sigma\cap C,k\in\mathcal{U}_2^\sigma\cap\mathcal{H}}\Delta_{j,k}\cdot\mathsf{H}(\mathsf{KA.Agree}(\mathsf{msk}_k^\sigma,\mathsf{mpk}_j^\sigma))\right) - \sum_{j\in\mathcal{U}_3^\sigma\cap C}\mathsf{H}(b_j^\sigma) - \sum_{j\in\mathcal{U}_3^\sigma\cap C,k\in(\mathcal{U}_2^\sigma\setminus\mathcal{U}_3^\sigma)\cap C}\Delta_{j,k}\cdot\mathsf{H}(\mathsf{KA.Agree}(\mathsf{msk}_k^\sigma,\mathsf{mpk}_j^\sigma)).$$

Finally, the simulator samples random $\{\widetilde{\boldsymbol{v}}_j\}_{j\in\mathcal{U}_3^\sigma\cap C}$ conditioned on that $\sum_{j\in\mathcal{U}_3^\sigma\cap C}\widetilde{\boldsymbol{v}}_j = \sum_{j\in\mathcal{U}_3^\sigma\cap C}\boldsymbol{v}_j^\sigma$.

We claim that the joint output distributions in the two hybrids are computationally indistinguishable. Note that all aborted parties behave identically in the two hybrids. First, we consider this distribution conditioned on that the simulator does not abort due to either the conditions in the previous hybrid or the condition in this hybrid. For any fixed $\sigma\in[\ell]$:

- **Case (i).** The joint distribution of the honest parties' outputs and the corrupt parties' views is identically distributed in the two hybrids *by the end of **Aggregate** phase* since (i) the simulator also uses the sum $\sum_{i\in\mathcal{U}_3^\sigma\cap\mathcal{H}}\boldsymbol{v}_i^\sigma$, and (ii) the simulator sets the honest parties' outputs as per the previous hybrid. It is left to consider the cheating flags maintained in $\mathcal{F}_{\mathsf{SecVerAgg}}$ and to be used in **Verify** phase. For each $i\in\mathcal{H}_{\mathsf{out}}^\sigma$, $\mathsf{Cheat}[\sigma,i]=0$ if and only if

$$\boldsymbol{a}^{\sigma,i} := \boldsymbol{a} + \left(\boldsymbol{p}^{\sigma,i} - \sum_{j\in\mathcal{U}_3^\sigma\cap\mathcal{H}}\boldsymbol{p}_j^\sigma\right) - \sum_{j\in\mathcal{U}_3^\sigma\cap C}\left(\widetilde{\boldsymbol{v}}_j + \mathsf{H}(b_j^\sigma) + \sum_{k\in\mathcal{U}_2^\sigma\setminus(\mathcal{U}_3^\sigma\cap C)}\Delta_{j,k}\cdot\mathsf{H}(\mathsf{KA.Agree}(\mathsf{msk}_k^\sigma,\mathsf{mpk}_j^\sigma))\right) = \boldsymbol{a},$$

  where ":=" comes from that $\{b_j^\sigma\}_{j\in\mathcal{U}_3^\sigma\cap\mathcal{H}}$ given to the party i are the same as those chosen by the simulator and used in the programming on $\mathsf{H}(b_j^\sigma)$ for each $j\in\mathcal{U}_3^\sigma\cap\mathcal{H}$ (otherwise the simulator will abort). For any $i\in\mathcal{H}_{\mathsf{out}}^\sigma$, $\mathsf{Cheat}[\sigma,i]=0$ happens with probability $1/q^d$ due to the randomness of $\{\widetilde{\boldsymbol{v}}_j\}_{j\in\mathcal{U}_3^\sigma\cap C}$.
- **Case (iii).** We consider the following two sub-cases.

  On the one hand, if $\mathcal{A}$ never uses malformed commitments on behalf of the corrupt parties, the straight-line extractor always succeeds. The reason is that $\mathcal{A}$ cannot open the corrupt parties' commitments to other values than those being committed in the σ-th **Round 1** (otherwise the simulator will abort) and thus there will be only one record in the list maintained by O for each commitment. Like **Case (i)**, the joint output distribution by the end of **Aggregate** phase is identical. One can also check that, for $i\in\mathcal{H}_{\mathsf{out}}^\sigma\cap\mathcal{H}_{\mathsf{pass}}^\sigma$, $\boldsymbol{a}^{\sigma,i}=\boldsymbol{a}$ always holds due to (i) the successful straight-line extraction, (ii) the party-wise consistent $\boldsymbol{p}^\sigma$ (by the case definition) and $\{\mathsf{mpk}_j^\sigma\}_{j\in\mathcal{U}_3^\sigma\cap C}$ (by the consistent syncMsg for $\mathcal{H}_{\mathsf{survive}}$), and (iii) the binding $(\{b_j^\sigma\}_{j\in\mathcal{U}_3^\sigma\cap C},\{\mathsf{msk}_k^\sigma\}_{k\in(\mathcal{U}_2^\sigma\setminus\mathcal{U}_3^\sigma)\cap C})$ and $(\{b_j^\sigma\}_{j\in\mathcal{U}_3^\sigma\cap\mathcal{H}},\{\mathsf{msk}_k^\sigma\}_{k\in(\mathcal{U}_2^\sigma\setminus\mathcal{U}_3^\sigma)\cap\mathcal{H}})$ (otherwise the simulator will abort). In this sub-case, $\mathsf{Cheat}[\sigma,i]=0$ happens with probability 1 for $i\in\mathcal{H}_{\mathsf{out}}^\sigma\cap\mathcal{H}_{\mathsf{pass}}^\sigma$; $\boldsymbol{p}^{\sigma,i}\neq\boldsymbol{p}^\sigma$ ensures that $\mathsf{Cheat}[\sigma,i]=0$ happens with probability 0 for $i\in\mathcal{H}_{\mathsf{out}}^\sigma\setminus\mathcal{H}_{\mathsf{pass}}^\sigma$.

  On the other hand, if $\mathcal{A}$ uses malformed commitments in the σ-th execution, all honest parties will abort in the σ-th **Round 5** from the view of $\mathcal{A}$. In the previous hybrid, this abort is due to the assertion in the σ-th **Round 5** (otherwise the simulator will abort). In this hybrid, this abort is due to the simulator sending $(\mathsf{abort},\cdot)$ to $\mathcal{F}_{\mathsf{SecVerAgg}}$ (since the straight-line extractor fails). In this sub-case, the joint output distribution in the two hybrids is **trivially** identical since all honest parties abort before **Verify** phase and the joint output distribution until the previous **Aggregate** phase is identical.

Now, we put these two cases together and, for non-triviality, consider that $\mathcal{A}$ never uses malformed commitments so that $\mathcal{H}_{\mathsf{out}}^\ell\cap\mathcal{H}\neq\varnothing$ for **Verify** phase. For each honest party $i\in\mathcal{H}_{\mathsf{out}}^\ell\subseteq\cdots\subseteq\mathcal{H}_{\mathsf{out}}^1$, there must exist a set $\mathcal{I}_i\subseteq[\ell]$ such that $\forall\sigma\in\mathcal{I}_i: i\in\mathcal{H}_{\mathsf{out}}^\sigma\setminus\mathcal{H}_{\mathsf{pass}}^\sigma$ and $\forall\sigma\in[\ell]\setminus\mathcal{I}_i: i\in\mathcal{H}_{\mathsf{out}}^\sigma\cap\mathcal{H}_{\mathsf{pass}}^\sigma$. On the one hand, if $\mathcal{I}_i=\varnothing$, the party i always outputs $(\mathsf{success},i)$ in this hybrid since $\mathcal{I}_i=\varnothing$ implies, for every $\sigma\in[\ell]$, $i\in\mathcal{H}_{\mathsf{out}}^\sigma\cap\mathcal{H}_{\mathsf{pass}}^\sigma\neq\varnothing$ and $\mathsf{Cheat}[\sigma,i]=0$. In the previous hybrid, $\mathcal{I}_i=\varnothing$ also makes the party i output $(\mathsf{success},i)$ due to the definition of $\mathcal{H}_{\mathsf{pass}}^\sigma$ and the linearity of the LHH scheme. On the other hand, it is left to consider $\mathcal{I}_i\neq\varnothing$. By union bound, the party i in this hybrid outputs $(\mathsf{success},i)$ with probability at most $(1/q^d)^{|\mathcal{I}_i|}$. In the previous hybrid with $\mathcal{I}_i\neq\varnothing$, the party i outputs $(\mathsf{success},i)$ with probability at most $1/q$, which is taken over the random coefficients. The statistical distance between the two joint distributions in **Verify** phase is bounded by $1/q$, which is negligible. In other words, conditioned on that the simulator does not abort, $\mathcal{A}$ can only distinguish this hybrid from the previous one with negligible advantage.

Then, we bound the probability that the simulator aborts. We have seen from the previous hybrids that the simulator aborts due to the conditions in these hybrids with negligible probability, and this probability is not skewed by the transition in this hybrid. In this hybrid, the simulator additionally aborts due to **Case (ii)**. Note that **Case (ii)** happens with negligible probability; otherwise $\mathcal{A}$ implies an efficient attacker against the collision resistance of the LHH scheme. Overall, the simulator still aborts with negligible probability.

Using the law of total probability, the joint output distributions in the two hybrids are computationally indistinguishable.

- Hybrid_{20}. Same as the previous one, except that the abort conditions for the simulator are removed. This hybrid is computationally indistinguishable from the previous one since it is known from the previous hybrid that the simulator aborts only with negligible probability. This simulator is essentially the simulator $\mathcal{S}$ in the ideal execution.

The above hybrid argument completes this proof. □

References

[BGG95] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography and application to virus protection. In *27th ACM STOC*, pages 45–56, Las Vegas, NV, USA, May 29 – June 1, 1995. ACM Press.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1175–1191, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 431–444, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[GLL+21] Xiaojie Guo, Zheli Liu, Jin Li, Jiqiang Gao, Boyu Hou, Changyu Dong, and Thar Baker. Verifl: Communication-efficient and fast verifiable aggregation for federated learning. *IEEE Trans. Inf. Forensics Secur.*, 16:1736–1751, 2021.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[Lin16] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. https://eprint.iacr.org/2016/046.

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 316–337, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.