A generic transformation from threshold homomorphic encryption to conditional homomorphic encryption

Huang Lin

September 20, 2016

I. Introduction

This write-up mainly aims to provide a solution to the second problem mentioned in the proposal for comprehensive solutions for private association studies on encrypted genomic-phenomic data [1]. Moreover, it also aims to provide a justification for the system architecture proposed in [2] based on my discussion with Jean Louis. It will start with the justification for the system architecture, then we will show why several trivial solutions are not sufficient for the aforementioned problem. After that, we will describe our solution.

A. Justification for system architecture (why trivial solutions would not work?)

A straightforward solution to realize the system design goal in [2] that immediately comes into mind is that one could let each individual directly send the homomorphic encryption of the message further encrypted using a threshold symmetric encryption scheme [3], where the decryption of the ciphertext can only be performed by MU and SPU collectively. This will guarantee that homomorphic operation can only be performed over the ciphertexts when it is agreed by both MU and SPU. However, this solution is not desirable as it is unknown how MU and SPU can perform the update of their secret key shares without the involvement of CI, which implies it is difficult to provide forward secrecy under this framework. Consequently, we assume each individual genomic data is encrypted using public key encryption under different public keys. Here the data has to be encrypted under different public keys in order to guarantee the compartmentation of individual genomic data access by individual patients. This requirement implies CI will generate a new public/secret key pair for each newcomer, which also fits our system model that assumes the individuals might join the system in any time and CI is not supposed to store any private information. Although one could argue that attribute based homomorphic encryption (ABHE) [4] can be applied to realize the access control of different individual genomic data samples, ABHE assumes that CI holds the master key, which contradicts our design goal that CI does not hold any master secret after initialization. This work mainly focuses on the comparison between garbled circuit based and homomorphic encryption based approach. Secret sharing based multi-party computation solution such as Sharemind [5] is not considered in this work because it requires both MU and SPU to store the secret shares of the input data, which contradicts our design requirement that SPU should be responsible for most of the storage and computational task.

B. Why existing solutions are not sufficient?

The main goal of the second problem in [1] is to guarantee that homomorphic operations can only be performed on those ciphertexts agreed by both MU and SPU. The reason why this is a necessary requirement is that when SPU is corrupted by the adversary, it could try to perform homomorphic operations over the ciphertexts which are not chosen by MU. Although we assume an honest-but-curious adversary model, it would be better if we have a technical enforcement over this particular requirement. A straightforward solution to this problem, which was actually our initial proposal in [2], includes the following operations: masking and partial decryption by SPU$\rightarrow$partial decryption and re-encryption by MU$\rightarrow$unmasking by SPU. This solution is not satisfactory mainly because it requires MU to perform decryption and re-encryption operations, which is too computationally expensive for MU. Another possible solution is to apply the multi-key fully homomorphic encryption scheme [6], [7], which allows the cloud server or SPU in our case to transform individual ciphertext encrypted under different public keys to a ciphertext encrypted under a jointed public keys containing all those different public keys. However, this solution is not appropriate for our application scenario mainly due to the fact that the decryption of the transformed ciphertexts in their proposal requires the cooperation of the data owners, which contradicts our requirements that the data owner should be offline after the initialization. Moreover, the application scenario of our system is quite different from theirs in the sense that in our case CI has access to both the data from the data owners and their secret keys while in the case of multi-key encryption the computing server does not have access to the secret keys, and hence our system can exploit this distinct feature to significantly enhance the performance of our proposal.

II. A GENERIC CONDITIONAL HOMOMORPHIC ENCRYPTION CONSTRUCTED FROM THRESHOLD HOMOMORPHIC ENCRYPTION SCHEME

A. Primitive idea

In this section, we propose a generic conditional homomorphic encryption scheme based on two-layer threshold encryption scheme to realize the design goal in the aforementioned problem. The outer layer threshold encryption scheme allows MU and SPU to collectively decide which ciphertexts are allowed to participate in the homomorphic operation, and the inner layer threshold homomorphic encryption scheme allows them to decrypt the final computational results jointly. Compared to the straightforward solution, MU only needs to perform a partial decryption instead of a partial decryption and a re-encryption in this process. The purpose of outer layer threshold encryption aims to enable MU to exert control over which ciphertexts permit homomorphic operations. Although one could realize this goal by encrypting the inner ciphertext under MU's public key, it would be impossible to realize the forward security under mobile adversary to be mentioned for the outer encryption if so.

A functionally equivalent solution would be to construct a threshold homomorphic encryption scheme to realize an access structure which allows MU and SPU to jointly transform different individual ciphertexts into a homomorphic ciphertext encrypted under a unique public key, the secret key of which will be further shared by SPU and MU such that it would allow them to jointly decrypt the final computational result. However, we believe the two-layer encryption framework is superior in terms of performance since it provides us flexibility to choose more efficient outer-layer threshold encryption scheme than the inner-layer homomorphic encryption scheme instead of sticking to the same homomorphic encryption scheme for both layers in this equivalent solution.

B. Basic scheme

Before the introduction of our proposal, we will first introduce the formal description of threshold encryption scheme, which is the building block for our transformation. The following exposition follows the notation from Shoup and Genarro [8] with an adaptation to our application scenario. The following description applies to both the outer threshold encryption **TE** and inner threshold homomorphic encryption **THE** scheme.

- At the beginning of the system, the probabilistic algorithm **Gen** takes the security parameter k as input, and outputs a public/secret key pair (PK_i, SK_i) for each user i. It also outputs secret key shares $\{\mathrm{SK}_{U_j(i)}\}_{j=1}^{n}$ for each computing server U_j responsible for the decryption.
- The encryption algorithm **Enc** encrypts a message $m \in \{0,1\}^k$ under the public key PK_i of user i to generate ciphertext $\mathrm{C}_i(m)$.
- The share decryption algorithm **ShareDec** decrypts a ciphertext $\mathrm{C}_i(m)$ with the secret key $\mathrm{SK}_{U_j(i)}$ to generate the decryption share $\mathrm{C}_j(i)(m)$.
- The **Combine** algorithm takes as input the decryption shares $\{\mathrm{C}_j(i)(m)\}$ and the public key PK_i. This algorithm outputs the original message m if and only if $\{j\} = [1, n]$, i.e., the decryption shares for all the computing servers are available.

For the **THE** scheme, there is an extra homomorphic operation algorithm **THE.HomoOp** that allows SPU to perform homomorphic operations over the ciphertext. The **THE.HomoOp** algorithm might consist of **Add** or **Add** together with **Mult** algorithms dependent on the concrete instantiation of the homomorphic encryption scheme. The chosen plaintext security of threshold encryption scheme can be found in Appendix.V-A.

The transformation from a threshold encryption scheme to a conditional homomorphic encryption scheme is formally described as follows:

- At the beginning of the system, CI runs the probabilistic algorithm **TE.Gen** to output a public/secret key pair (PK_i, SK_i) for each user i. For each user i, **TE.Gen** also outputs secret key shares $\mathrm{SK}_{MU(i)}$, $\mathrm{SK}_{SPU(i)}$ for MU and SPU respectively. It also runs **THE.Gen** generates a public/secret key pair (PK_H, SK_H) for homomorphic operation, and outputs $\mathrm{SK}_{MU(H)}$, $\mathrm{SK}_{SPU(H)}$ as secret key shares for MU and SPU.
- CI runs the encryption algorithm **Enc** that encrypts a message $m_i \in \{0,1\}^k$ under the public key PK_i of user i to generate ciphertext as $\mathrm{C}_i(m_i)$=$\{$**TE.Enc**$_{\mathrm{PK}_i}(K_i)$,**SEnc**$_{K_i}$(**THE.Enc**$_{\mathrm{PK}_H}(m_i)$)$\}$. Here a key encapsulation mechanisms is applied where K_i is the symmetric key used in symmetric encryption scheme **SEnc**.
- In **HomoShareDec** algorithm, both SPU and MU run the partial decryption algorithm **TE.ShareDec** which takes the original ciphertext $\mathrm{C}_i(m_i)$ along with the respective secret key $\mathrm{SK}_{SPU(i)}$ or $\mathrm{SK}_{MU(i)}$ to generate the partial ciphertext. Then SPU executes **TE.Combine** algorithm to recover the message encrypted under PK_i, i.e., K_i. K_i will be further used for the decryption of **THE.Enc**$_{\mathrm{PK}_H}(m_i)$. This algorithm will output all the ciphertexts $\{\mathrm{C}_H(m_i)\}$=$\left\{\textbf{THE.Enc}_{\mathrm{PK}_H}(m_i)\right\}$ that permit homomorphic operations.
- After running homomorphic algorithm **THE.HomoOp** over the cipertext $\{\mathrm{C}_H(m_i)\}$ to generate ciphertext for the final computational result $\mathrm{C}_H(m)$, SPU and MU will again execute the partial decryption algorithm **THE.ShareDec** algorithm to generate the partial decryption. Then MU will run **THE.Combine** algorithm to decrypt the computational result m.

The security of the proposed transformation is straightforward to argue. Assuming the outer-layer threshold encryption scheme **TE** and **SEnc** used in key encapsulation mechanism are chosen-plaintext secure, they together can guarantee SPU can only perform homomorphic operations over $\mathrm{C}_H(m_i)$ if it is agreed by both SPU and MU. Finally, the semantic security of **THE** can guarantee that the final decryption result is known to MU only if SPU and MU both agree.

C. Concrete instantiation: A threshold somewhat homomorphic encryption scheme

The following is a threshold somewhat homomorphic encryption scheme which is based on FV scheme proposed by Fan and Vercauteren [9]. This scheme can serve as the underlying threshold homomorphic encryption **THE** invoked in the aforementioned transformation. However, **THE** can also be the threshold additive homomorphic encryption based on BCP encryption [10] described in Appendix V-B. We have borrowed some notations from the original description of FV scheme [9].

- **Gen**(parms): Run the FV.SH.**SecretKeyGen**(λ) and FV.SH.**PublicKeyGen(SK)** to output key pair for homomorphic operation as (PK_H, SK_H)=
$$\left(\left(\left[-\left(a_{\mathrm{H}} \cdot s_{\mathrm{H}} + e_{\mathrm{H}}\right)\right]_q, a_{\mathrm{H}}\right), s_{\mathrm{H}}\right)$$
, and the evaluation key **rlk** will be generated as in FV.SH.**EvaluateKeyGen**. The secret share of the decryption key for SPU and MU will be distributed as ($\mathrm{SK}_{SPU(H)}$, $\mathrm{SK}_{MU(H)}$)=$\left(\mathbf{s}_{\mathrm{H}} - \mathbf{s}_{\mathrm{MU(H)}}, \mathbf{s}_{\mathrm{MU(H)}}\right)$, where $\mathbf{s}_{\mathrm{MU(H)}}$ by running FV.SH.**SecretKeyGen**(λ).
- **Enc**(PK_H, $\mathbf{m}_i$): Run FV.SH.**Enc**(PK_H, $\mathbf{m}_i$) to generate $\mathrm{C}_H(\mathbf{m}_i)$=
$$\left(\left[\mathbf{p}_{\mathrm{H0}}\mathbf{u} + \mathbf{e}_1 + \Delta\mathbf{m}_i\right]_q, \left[\mathbf{p}_{\mathrm{H1}}\mathbf{u} + \mathbf{e}_2\right]_q\right)$$
.
- **Add**($\mathrm{C}_H(\mathbf{m}_1)$, $\mathrm{C}_H(\mathbf{m}_2)$): Given two ciphertexts $\mathrm{C}_H(\mathbf{m}_1)$ and $\mathrm{C}_H(\mathbf{m}_2)$, the algorithm runs FV.SH.**Add**($\mathrm{C}_H(\mathbf{m}_1)$, $\mathrm{C}_H(\mathbf{m}_2)$) on these two ciphertexts to output the ciphertext C_{add}=$[\mathrm{C}_H(\mathbf{m}_1)+\mathrm{C}_H(\mathbf{m}_2)]_q$.
- **Mult**($\mathrm{C}_H(m_1)$, $\mathrm{C}_H(m_2)$, **rlk**): Given two ciphertexts $\mathrm{C}_H(m_1)$ and $\mathrm{C}_H(m_2)$, run FV.SH.**Mult**($\mathrm{C}_H(m_1)$, $\mathrm{C}_H(m_2)$, **rlk**) to output C_{mult}.
- **ShareDec**($\mathrm{SK}_{SPU(H)}$,$\mathrm{SK}_{MU(H)}$,$\mathrm{C}_H(\mathbf{m})$): Given SPU's private decryption key $\mathrm{SK}_{SPU(H)}$=$\mathbf{s}_{\mathrm{H}} - \mathbf{s}_{\mathrm{MU(H)}}$, and a ciphertext $\mathrm{C}_H(\mathbf{m})$=($\mathrm{C}_H(\mathbf{m})[0]$, $\mathrm{C}_H(\mathbf{m})[1]$)=
$$\left(\left[\mathbf{p}_{\mathrm{H0}}\mathbf{u} + \mathbf{e}_1 + \Delta\mathbf{m}\right]_q, \left[\mathbf{p}_{\mathrm{H1}}\mathbf{u} + \mathbf{e}_2\right]_q\right)$$
, SPU first computes
$$\begin{aligned}&\mathrm{C}_{\mathrm{SPU(H)}}(\mathbf{m})\\ &= \left[\mathrm{SK}_{SPU(H)} \cdot \mathrm{C}_H(\mathbf{m})[1] + \mathbf{e}_{SPU}\right]_q\\ &= \left[\left(\mathbf{s}_{\mathrm{H}} - \mathbf{s}_{\mathrm{MU(H)}}\right) \cdot \left(\mathbf{p}_{\mathrm{H1}}u + \mathbf{e}_2\right) + \mathbf{e}_{SPU}\right]_q\end{aligned}$$
. Given MU's private decryption key $\mathrm{SK}_{MU(H)}$=$\mathbf{s}_{MU(H)}$, and the ciphertext $\mathrm{C}_H(\mathbf{m})$, MU execute the similar procedure as SPU, and computes
$$\begin{aligned}&\mathrm{C}_{\mathrm{MU(H)}}(\mathbf{m})\\ &= \left[\mathrm{SK}_{MU(H)} \cdot \mathrm{C}_H(m)[1] + \mathbf{e}_{MU}\right]_q\\ &= \left[\mathbf{s}_{\mathrm{MU(H)}} \cdot \left(\mathbf{p}_{\mathrm{H1}}u + \mathbf{e}_2\right) + \mathbf{e}_{MU}\right]_q\end{aligned}$$
. In the above procedure the re-randomization noises e_{SPU}, e_{MU} are sampled from χ_{err} as in the basic scheme.
- **Combine**($\mathrm{C}_{SPU(H)}(\mathbf{m})$,$\mathrm{C}_{MU(H)}(\mathbf{m})$,$\mathrm{C}_H(\mathbf{m})$): Given the partial decryption shares from MU and SPU and the original ciphertext $\mathrm{C}_H(\mathbf{m})$=($\mathrm{C}_H(\mathbf{m})[0]$,$\mathrm{C}_H(\mathbf{m})[1]$), MU computes
$$\begin{aligned}&\left[\left\lfloor t/q\left[\mathrm{C}_H(\mathbf{m})[0] + \mathrm{C}_{MU(H)}(\mathbf{m}) + \mathrm{C}_{SPU(H)}(\mathbf{m})\right]_q\right\rfloor\right]_t\\ &= \left[\left\lfloor t/q[\mathrm{C}_H(\mathbf{m})[0] + \mathbf{s}_{\mathrm{H}} \cdot \left(\mathbf{p}_{\mathrm{H1}}u + \mathbf{e}_2\right) + \mathbf{e}_{MU} + \mathbf{e}_{SPU}]_q\right\rfloor\right]_t\\ &= \left[\left\lfloor t/q[\mathrm{C}_H(\mathbf{m})[0] + \mathbf{s}_{\mathrm{H}} \cdot \mathrm{C}_H(\mathbf{m})[1] + \mathbf{e}_{MU} + \mathbf{e}_{SPU}]_q\right\rfloor\right]_t\\ &= \mathbf{m}\end{aligned}$$

Theorem 1: The final decryption step in **Combine**($\mathrm{C}_{SPU(H)}(\mathbf{m})$,$\mathrm{C}_{MU(H)}(\mathbf{m})$,$\mathrm{C}_H(\mathbf{m})$) can successfully recover the underlying message m when the parameters is chosen as suggested in [9].

Proof 1: The final equality holds in the above formula mainly due to the fact that the inherent noise v in $f_H \cdot C_H(m) + e_{SPU} + e_{MU}$ with respect to m and f_H[1] is sufficiently small to satisfy the noise bounds mentioned for YASHE' in [9]. The

[1]The interested readers are referred to [9] for the definition of inherent noise.

aforementioned noise can be re-written as $v_H + e_{SPU} + e_{MU}$, where v_H is the inherent noise of $C_H(m)$ with respect to m and f_H. We have

$$\begin{aligned}
&\|v\|_\infty \\
&= \|v_H + e_{SPU} + e_{MU}\|_\infty \\
&\leq \|v_H\|_\infty + \|e_{SPU}\|_\infty + \|e_{MU}\|_\infty \\
&\leq \|v_H\|_\infty + 2B_{err} \\
&< \|v_H\|_\infty + 2\delta t B_{key} B_{err} + \delta t B_{key} r_t(q) \\
&< \|v_H\|_\infty + \Delta/2
\end{aligned}$$

The second inequality in the above formula holds because B_{err} is the upper bound for e_{err}. The fourth inequality holds since $\delta t B_{key} > 1$ and $\delta t B_{key} r_t(q) > 0$ in [9]. $\Delta/2$ in $\|v_H\|_\infty + \Delta/2$ is the upper bound for the inherent noise in the basic ciphertext in YASHE'. In other words, the additional noise due to the **ShareDec** algorithm in our scheme is smaller than a homomorphic addition of a basic ciphertext. This implies that as long as we choose the parameter such that it tolerates an extra homomorphic addition on top of the predefined function for the original homomorphic evaluation, which will suffice the noise bound requirement, and hence conclude the proof.

The security of the proposed threshold homomorphic encryption scheme can be summarized as the following theorem.

***Theorem** 2:* Let d be a positive integer, q and $t < q$ be two moduli, w be a fixed positive integer, and let χ_{key} and χ_{err} be distributions on R. The proposed threshold encryption scheme is CPA secure under the $RLWE_{d,q,\chi_{\text{err}}}$ assumption, the $DSPR_{d,q,\chi_{\text{key}}}$ assumption, and the assumption that the scheme remains IND-CPA secure even when the evaluation key evk is known to the adversary.

***Proof** 2:* The proof of this theorem is almost identical to that of Theorem 6 of [11] except that the adversary need to provide partial decryption keys $\text{SK}_{SPU(H)}$ and $\text{SK}_{MU(H)}$ for either SPU or MU respectively. However, this can be done by asking the simulator to generate an invalid but correctly distributed secret key SK_H, and then the decryption key of either SPU or MU can be generated as in real practice. This guarantees the correct distribution of the decryption key of SPU or MU, which is sufficient for the successful simulation of the security game, and hence conclude the proof by choosing the parameter such that it tolerates an extra $n-1$ homomorphic additions.

D. Optimization

Optimization for $\sum_{i=1}^{n} p_i g_i a_i$ This section describes the optimized algorithm we use for the following computation: given message vectors $P = \{< p_1, \cdots, p_n >, G =< g_1, \cdots, g_n >, A =< a_1, \cdots, a_n > | p_i \in \{1,2\}, g_i \in \{1,2,3\}, a_i \in \{1,2\}\}$, generate the encryption of these vectors and then compute the encryption of $R = \sum_{i=1}^{n} p_i g_i a_i$, and run the threshold decryption algorithm introduced in the above section to decrypt the final result. The algorithm mainly consists of the following steps:

1) **Setup**: Choose t to be a prime such that $t \equiv 1 \pmod m$ s.t. the prime field $\mathbb{F}_t$ contains the primitive $m-$th root of unity ζ, and hence $\Phi(x) = \prod_{i=0}^{\varphi(m)-1} (x - \zeta^{2i+1})$. We have $Z_t[x]/\Phi_m(x) \cong \bigotimes_{i=0}^{\varphi(m)-1} Z_t[x]/(x - \zeta^{2i+1})$, where $\otimes$ denotes direct product and $\cong$ means "is isomorphic to". We note t is required to satisfy the condition $-t/2 < R < t/2$ such that the final result is correct when it is mod by t. In the FV scheme, m is usually chosen to be the positive integer power of 2, i.e., $m = 2^z, z \in Z$.
2) **Packed encryption**(V): This algorithm takes a vector $V \in (Z_t)^{\varphi(m)}$ as input, and generates the respective packed encryption. We first view V as an element in $\bigotimes_{i=0}^{\varphi(m)-1} Z_t[x]/(x - \zeta^{2i+1})$, our first step is to transform it to the corresponding element in $Z_t[x]/\Phi_m(x)$ under the isomorphism between $\bigotimes_{i=0}^{\varphi(m)-1} Z_t[x]/(x - \zeta^{2i+1})$ and $Z_t[x]/\Phi_m(x)$. The map used in this step is basically the compose map in [12], which can be achieved at the cost of $O(\varphi(m)\log(\varphi(m)))$ by using discrete FFT algorithm.
   Then, we can run the aforementioned **Enc**(PK_H, $T(V)$) algorithm to generate the respective ciphertext $\text{C}_H(T(V))$.
3) **Packed homomorphic evaluation**($\text{C}_\text{H}(\text{T}(\text{V}_1))$, $\text{C}_\text{H}(\text{T}(\text{V}_2))$, $\text{C}_\text{H}(\text{T}(\text{V}_2))$): Run **Mult**($\text{C}_\text{H}(\text{T}(\text{V}_1))$, **Mult**($\text{C}_\text{H}(\text{T}(\text{V}_2))$, $\text{C}_\text{H}(\text{T}(\text{V}_3))$)) to generate $\text{C}_\text{H}(\text{T}(\text{V}_1)\cdot\text{T}(\text{V}_2)\cdot\text{T}(\text{V}_3))$. The SPU then applies the following maps $f^{(i)}(X) = f(X^i) \bmod \Phi_m(X), i \in Z^*_{\varphi(m)}$ to generate ciphertexts $\text{C}^{(\text{i})}_\text{H}(\text{T}(\text{V}_1)\cdot\text{T}(\text{V}_2)\cdot\text{T}(\text{V}_3))$. The SPU will also apply the same map to the respective secret key. The MU will perform the similar operations for its partial ciphertext and secret key. The computational cost of this step is almost negligible since each map is dependent on the permutation of the coefficients of the polynomial. The SPU and MU will then jointly choose a random element $\Upsilon = \left(0, \gamma_1, \cdots, \gamma_{\varphi(m)-1}\right) \in \bigotimes_{i=0}^{\varphi(m)-1} Z_t[x]/(x - \zeta^{2i+1})$, and then compute the respective coefficient representation by computing $T(\Upsilon) = V_m^{-1} \cdot \Upsilon$, which will be then used to mask

the partial decryption results from both SPU and MU. In our implementation, the masking of the plaintext slots happens in the final step of the partial decryption as the permutation of partial decryption result has to be executed before the masking in order to guarantee the correctness of the decryption results.

4) **Packed decryption**($\mathrm{C_H}(\Upsilon) + \sum_{i=0}^{\varphi(m)-1} \mathrm{C}_H^{(i)}\left(T(V_1)\cdot T(V_2)\cdot T(V_3)\right)$): Run **ShareDec**($SK_{SPU(H)}$, $SK_{MU(H)}$, $\mathrm{C_H}(\Upsilon) + \sum_{i=0}^{\varphi(m)-1} \mathrm{C}_H^{(i)}\left(T(V_1)\cdot T(V_2)\cdot T(V_3)\right)$) to generate a polynomial $A \in {Z_t[x]}/{\Phi_m(x)}$. The first component of the evaluation representation of A will be $\sum_{i=0}^{\varphi(m)-1} V_{1i}\cdot V_{2i}\cdot V_{3i}$. However, the above decryption result will be in the form of coefficient representation, hence we need to further compute $V_m \cdot A$ to generate the respective evaluation representation, the first component of which will be the desired result.

We assume n is[2] a multiple of $\varphi(m)$. Then we can sequentially run **Packed encryption** over each blocks of the input messages, and run **Packed homomorphic evaluation** over each corresponding blocks of encryption. The resultant encryption will be decrypted through running **Packed decryption**.

Optimization for comparison circuit: Our somewhat homomorphic encryption based solution does not directly apply the non-interactive comparison approach [13] due to the prohibitive degree of multiplication it requires. Instead we consider replace the homomorphic encryption scheme used in the non-interactive secure comparison protocol with somewhat homomorphic encryption scheme and compare the performances of the respective solution.

E. Realizing forward secrecy: proactive secret sharing based solution

The first naive solution proposed in [2] offers certain level of forward secrecy in the sense that secret keys stolen during the homomorphic operation in a particular period is useless in the subsequent periods since during the re-encryption phase MU can update the public/secret key pair for the homomorphic operation. This is a useful property especially considering the fact that the storage of genomic data is usually meant to be long-term. In this section, we will show how to apply proactive secret sharing scheme [14], [15] to strengthen the basic scheme such that it is secure against a particular adversary model named mobile adversary. Although our basic adversary model assumes that it at most corrupts only one party, i.e., either MU on SPU at one specific time, it does not rule out the possibility that a mobile adversary that corrupts one party in a certain period of time, then later the other party in a different period of time. In fact, the longer genomic data is stored, the more likely it is subject to such kind of adversary.

The proactive secret sharing scheme (PSSS) aims to provide a key update mechanism against this adversary. In PSSS, we assume two parties P_1 and P_2 which received two secret shares s_1 and s_2 of a secret s from a dealer such that $s_1 + s_2 = s$. In the PSSS protocol, they are supposed to refresh their secret shares without the assistance of the dealer. Each party P_i will first reboot and re-initialize their system by a trusted agent (e.g., from a read-only device) [14]. Then $P_i, i \in [1, 2]$ will serve as a dealer to generate the secret shares of 0 to be r_{i1} and r_{i2} and send $r_{ij}, j \neq i$ to the other party P_j in a secure channel. P_i will update his secret share to be $s'_i = s_i + \sum_{j=1}^{2} r_{ji}$ after receiving secret randomness from the other party, and erase all variables except s'_i. It is easy to verify that the updated secret shares remains valid secret shares for the original secret s due to the linearity of secret sharing scheme and also the fact that the added randomness is just secret shares of 0. This scheme can defend against the mobile adversary since we assume at most one party in each period is corrupted by the adversary and also the fact the update is performed over secure channel. For a more formal proof of the security of PSSS protocol, the readers can refer to [15]. We note that the original protocol assume the adversary at most corrupt $t < n/2$ parties involved in the protocol in each period mainly due to the fault tolerance requirement. However, this requirement is unnecessary in our design since we assume an honest-but-curious adversary model.

In our PSSS based solution, we will apply the above approach to periodically update the secret key pairs $\mathrm{SK}_{MU(i)}, \mathrm{SK}_{SPU(i)}$ after a period consisting of a predefined number of interactions between the MU and SPU ends. At the end of each period, we will add the secret shares of 0 to update $\mathrm{SK}_{MU(i)}$, $\mathrm{SK}_{SPU(i)}$ to $\mathrm{SK}'_{MU(i)}$, $\mathrm{SK}'_{SPU(i)}$. Similarly, we can also update $\mathrm{SK}_{MU(H)}$, $\mathrm{SK}_{SPU(H)}$ to $\mathrm{SK}'_{MU(H)}$, $\mathrm{SK}'_{SPU(H)}$. Since $\mathrm{SK}'_{MU(i)}$, $\mathrm{SK}'_{SPU(i)}$ remains valid secret shares of SK_i, and hence these secret keys can still be used for selecting which ciphertext to perform homomorphic operation. Similarly, the fact that $\mathrm{SK}'_{MU(H)}$, $\mathrm{SK}'_{SPU(H)}$ remains valid secret shares of SK'_H guarantees that the final decryption results can be successfully decrypted.

There is a caveat for the secret key update in the threshold SHE scheme. If we assume that χ distribution used in SHE scheme is Gaussian distribution as described in [11], then the distributed random shares for 0 must also be sampled from Gaussian distribution. Consequently, the renewed secret key will still be sampled from Gaussian distribution except with different expected value and variance due to the property of sum of normally distributed random variables [**?**]. The B-bounded requirement of the renewed secret key can still be satisfied by rejecting samples with norm larger than B such that the Gaussian distribution is truncated as required [11].

[2]If n is not, we can pad enough 0's in P (G or A)such that it would be the next multiple of $\varphi(m)$

III. A CONSTRUCTION BASED ON PROXY HOMOMORPHIC RE-ENCRYPTION SCHEME

In this section, we first propose a proxy homomorphic re-encryption (PHRE) scheme, and then we show how to build our system on top of it. Our re-encryption scheme is also based upon YASHE' proposed by Bos et al [11]. It bears certain similarity to the recently proposed NTRUReEncrypt scheme [16], however, since their construction is based upon Stehle and Steinfeld's NTRU scheme [17], which does not support full homomorphic operation, their construction cannot be directly used in our system. The formal definition of proxy homomorphic re-encryption can be found in the appendix. The proposed proxy homomorphic re-encryption scheme is shown as follows:

- **KeyGen**(parms): This algorithm takes security parameter parms as input, and it will run YASHE'.**KeyGen** as a subroutine. It will output secret/public key pair for user A as $\mathrm{SK}_A = f_A$, PK_A=h_A, where they both distributed as in YASHE'.**KeyGen**.
- **ReKeyGen**(SK_A, SK_B): On input the secret keys SK_A=f_A and SK_B=f_B, the re-encryption key generation algorithm **ReKeyGen** computes the re-encryption key between users A and B as $\mathrm{RK}_{A\to B}$=$\mathrm{SK}_A \cdot \mathrm{SK}_B^{-1}$=$f_A \cdot f_B^{-1}$.
- **Enc**(PK_A, M): On input the public key PK_A and a message $M \in \mathcal{M}$, it runs YASHE'.**Enc**$(\mathrm{PK_A}, \mathrm{M})$ to generate the ciphertext as C_A=$[\lfloor q/t \rfloor M + e + hs]_q \in R$.
- **ReEnc**($\mathrm{RK}_{A\to B}$, C_A): On input a re-encryption key $\mathrm{RK}_{A\to B}$ and a ciphertext C_A, the re-encryption algorithm samples $e' \leftarrow \chi_{err}$, and outputs the re-encrypted ciphertext as

$$C_B = [C_A \cdot \mathrm{RK_{A\to B}} + \mathrm{e}]_q \in R$$

.
- **Add**($\mathrm{C}_H(m_1)$, $\mathrm{C}_H(m_2)$): Given two ciphertexts $\mathrm{C}_H(m_1)$ and $\mathrm{C}_H(m_2)$, the algorithm runs YASHE'.**Add** on these two ciphertexts to output the ciphertext C_{add}=$[\mathrm{C}_H(m_1)$+$\mathrm{C}_H(m_1)]_q$.
- **Mult**($\mathrm{C}_H(m_1)$, $\mathrm{C}_H(m_2)$, **evk**): Given two ciphertexts $\mathrm{C}_H(m_1)$ and $\mathrm{C}_H(m_2)$, run YASHE'.**Mult** to output $cmult$.
- **Dec**(SK, C): On input the secret key SK and a ciphertext C, this algorithm runs YASHE'.**Dec**(SK,C) to output the underlying message.

Similar to that of [16], the correctness of decryption mainly consists of the correct decryption of the ciphertext generated by **Enc**(PK_A, M) and **ReEnc**($\mathrm{RK}_{A\to B}$, C_A). The correct decryption for the ciphertext output from **Enc**(PK_A, M) follows from that of the underlying YASHE' scheme [11]. The decryption correctness for the re-encrypted ciphertext can be stated as the following theorem:

Theorem 3: For any key pair sequence $\{(\mathrm{PK}_i, \mathrm{SK}_i)\}_{i\in[1,n]}$ output by **KeyGen**(parms), and an ciphertext $C_1(m)$ generated by **Enc**(PK_1, m), where $m \in \mathcal{M}$ and a collection of re-encryption keys $\{\mathrm{RK_{i\to i+1}}\}_{1\le i\le n}$ generated by **ReKeyGen**(SK_i, SK_{i+1}), the following equality holds:

$$\mathbf{Dec}(\mathrm{SK}_n, \mathbf{ReEnc}(\mathrm{RK}_{n-1\to n}, \cdots, \mathbf{ReEnc}(\mathrm{RK}_{1\to 2}, C_1(m)))) = m$$

Proof 3: Let $C_n(m)$ denote the ciphertext **ReEnc**$(\mathrm{RK}_{n-1\to n}, \cdots, \mathbf{ReEnc}(\mathrm{RK}_{2\to 1}, C_1(m)))$. We have $C_n(m) = f_1 \cdot f_n^{-1} \cdot C_1(m) + \sum_{i=2}^{n-1} f_i \cdot f_n^{-1} \cdot e_{i-1} + e_{n-1}$, where $f_i = \mathrm{SK_i}$ and e_i is the noise introduced by the re-encryption with re-encryption key $\mathrm{RK}_{i\to i+1}$. For the final decryption to be successful, the inherent noise v of $C_n(m)$ with respect to $\mathrm{SK}_n = f_n$ and m should be sufficiently small. In the following, we argue that the noise is upper bound by that of ciphertext for the homomorphic addition of the original ciphertext with $n-1$ basic ciphertext.

We observe that

$$\begin{aligned}
&\|v\|_\infty \\
&= \left\| v_1 + \sum_{i=2}^{n-1} f_i \cdot e_{i-1} + f_n \cdot e_{n-1} \right\|_\infty \\
&< \|v_1\|_\infty + \left\| \sum_{i=2}^{n-1} f_i \cdot e_{i-1} \right\|_\infty + \|f_n \cdot e_{n-1}\|_\infty \\
&< \|v_1\|_\infty + \sum_{i=2}^{n-1} B_{key} B_{err} + B_{key} B_{err} \\
&< \|v_1\|_\infty + (n-1) B_{key} B_{err} \\
&< \|v_1\|_\infty + (n-1) 2\delta t B_{key} B_{err} \\
&< \|v_1\|_\infty + (n-1)\Delta/2
\end{aligned}$$

The second inequality in the above formula holds since f_i is bounded by B_{key}. By applying the same argument in the proof of Theorem II-C, we conclude that the additional noise is indeed bounded by what can be introduced through $n-1$ homomorphic addition, and hence conclude the proof.

Our system can be built upon the proposed PHRE scheme as follows:

- At the beginning of the system, CI first generates public/secret key pair (PK_i, SK_i)=(h_i, f_i)$\in R \times R_q^\times$ for each user i, and (PK_{MU}, SK_{MU})=(h_{MU}, f_{MU}) $\in R \times R_q^\times$ by running the PHRE.**KeyGen** algorithm. It will also run the PHRE.**KeyGen**

algorithm to output key pair for homomorphic operation as (PK_H, SK_H)=(h_H, f_H)$\in R \times R_q^{\times}$, and the evaluation key γ_H. For each user U_i, CI will execute **ReKeyGen**(SK_{U_i}, SK_H) and **ReKeyGen**(SK_H, SK_A) to generate $rk_{U_i \to H}$=$f_{U_i} \cdot f_H^{-1}$ and $rk_{H \to MU}$=$f_H \cdot f_{MU}^{-1}$, which will be then delivered to MU and SPU respectively.

- CI then runs PHRE.**Enc**(h_i, m_i) to generate $C_i(m_i)$=$[\lfloor q/t \rfloor m_i + e + h_i \cdot s]_q \in R$, where h_i is the public key for user U_i.
- Given the re-key $rk_{U_i \to H}$=$f_{U_i} \cdot f_H^{-1}$ hold by MU, and a ciphertext $C_i(m_i)$=$[\lfloor q/t \rfloor m + e + h_i \cdot s]_q$, MU executes **ReEnc**($rk_{U_i \to H}$, $C_i(m)$) to generate $C_H(m_i)$, which will then delivered to SPU to perform homomorphic operations by running PHRE.**Add** and PHRE.**Mult** algorithms to output $C_H(m)$.
- Given $C_H(m)$=$[\lfloor q/t \rfloor m + e + h_H \cdot s]_q$ and $rk_{H \to MU}$=$f_H \cdot f_{MU}^{-1}$, SPU executes **ReEnc**($rk_{H \to MU}$, $C_H(m)$) to generate $C_{MU}(m)$, which will be then delivered to MU. MU will then use SK_{MU} to decrypt $C_{MU}(m)$ by running **Dec**(SK_{MU}, $C_{MU}(m)$).

IV. Performance evaluation: finding the optimal trade-off between communication and computational cost of somewhat homomorphic encryption scheme

The trade-off between the communication and computational cost of SHE can be demonstrated through adjusting the degree of homomorphic multiplication operations allowed by the SHE scheme. In the secure multiplication protocol, the extreme case would be directly treating the SHE scheme as an additive homomorphic encryption (AHE) scheme and run the interactive secure multiplication protocol based on it. On the other end of spectrum, the experiment can be performed by using SHE scheme which accommodates two level multiplication operations. We can also employ SHE scheme with one multiplication in conjunction with the interactive secure multiplication protocol (based on the respective SHE scheme serving as AHE) to see which among these three solutions is the optimal one.

The higher the levels of multiplication operations allowed by SHE, the higher the computational cost is. On the other hand, although SHE serving as AHE is computationally less expensive even compared with traditional additive homomorphic encryption scheme such as DGK encryption [18], the communication cost is higher. Therefore, the proposed experimentation approach will help find out the optimal trade-off between these two types of costs.

References

[1] Huang Lin. Proposal for comprehensive solutions for private association studies on encrypted genomic-phenomic data. Feb,2016.

[2] Jean Louis Raisaro, Erman Ayday, Paul J. McLaren, Amalio Telenti, and Jean-Pierre Hubaux. Private association studies on encrypted genomic-phenomic data. 2016.

[3] Keith M Martin, Rei Safavi-Naini, Huaxiong Wang, and Peter R Wild. Distributing the encryption and decryption of a block cipher. *Designs, Codes and Cryptography*, 36(3):263–287, 2005.

[4] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013*, pages 75–92. Springer, 2013.

[5] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security-ESORICS 2008*, pages 192–206. Springer, 2008.

[6] Chris Peikert and Sina Shiehian. Multi-key fhe from lwe, revisited. 2016.

[7] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM, 2012.

[8] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *EUROCRYPT'98*, pages 1–16. Springer, 1998.

[9] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption.

[10] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *Advances in Cryptology-ASIACRYPT 2003*, pages 37–54. Springer, 2003.

[11] Joppe W Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding*, pages 45–64. Springer, 2013.

[12] Rachel Player Kim Laine. Simple encrypted arithmetic library - seal (v2.0). Technical report, September 2016.

[13] Miran Kim and Kristin Lauter. Private genome analysis through homomorphic encryption. *BMC medical informatics and decision making*, 15(Suppl 5):S3, 2015.

[14] Christian Cachin. Lecture notes on proactive security. https://lpd.epfl.ch/site/_media/education/sdc_proactive.pdf?id=education%3Asecure_distributed_computing&cache=cache, 2009.

[15] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in CryptologyCRYPT095*, pages 339–352. Springer, 1995.

[16] David Nuñez, Isaac Agudo, and Javier Lopez. Ntrureencrypt: An efficient proxy re-encryption scheme based on ntru. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 179–189. ACM, 2015.

[17] Damien Stehlé and Ron Steinfeld. Making ntruencrypt and ntrusign as secure as standard worst-case problems over ideal lattices. *IACR Cryptology ePrint Archive*, 2013:4, 2013.

[18] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *Information security and privacy*, pages 416–430. Springer, 2007.

V. Appendix

A. Definition for chosen plaintext attack security

The following game defines the chosen plaintext security for threshold homomorphic scheme. Chosen plaintext attack (CPA) secure means that any polynomial time adversary has a negligible advantage in winning the following game:

- The adversary chooses to corrupt either MU or SPU.

- The key generation algorithm is run. The private key of the corrupted party either MU or SPU is given to the adversary, while the other private key is given to the honest party, and kept secret from the adversary. The adversary of course receives the public key as well.
- The adversary chooses two messages m_0 and m_1 (of the same length). These are given to an encryption oracle that chooses $b \in \{0,1\}$ at random, and gives the target ciphertext ϕ'= Enc(PK, m_b) to the adversary.
- At the end of the game, the adversary outputs $b' \in \{0,1\}$. The adversarys advantage is defined to be the absolute difference between 1/2 and the probability that $b' = b$.

B. Threshold additive homomorphic encryption based on BCP encryption [10]

- **Gen**(parms): Let PK_H=$(n, g, h = g^x)$ represent the public key of the BCP cryptosystem. Then, the strong private key is the factorization of $n = pq$ (p, q are safe primes), and the weak private key SK_H is $x \in \left[1, n^2/2\right]$. The partial decryption key for MU and SPU will be generated as $\text{SK}_{MU} = x_{MU}$, $\text{SK}_{SPU} = x_{SPU}$ such that $x_{MU} + x_{SPU} = x$. Furthermore, let g be of order $(p-1)(q-1)/2$. Then by selecting a random $a \in Z*_{n^2}$, it can easily be computed as $g = -a^{2n}$.
- **Enc**(PK_H, m_i): Choose random $r \in [1, n/4]$, the encryption of message m_i under public key PK_H is $\text{C}_H(m_i)$=$(\text{C}_1, \text{C}_2) = \left(g^r \mod n^2, h^r\,(1+mn) \mod n^2\right)$.
- **Add**($\text{C}_H(m_1)$, $\text{C}_H(m_2)$): This algorithm is identical to that of BCP encryption.
- **ShareDec**($\text{SK}_{SPU(H)}$,$\text{SK}_{MU(H)}$,$\text{C}_H(m)$): Given $\text{C}_H(m_i)$=(C_1, C_2), $\text{SK}_{MU(H)}$ and $\text{SK}_{SPU}(H)$, The partial decryption of MU and SPU can proceed as follows: $\text{C}_{MU(H)}(m)$=$\left(\text{C}_1{}^{\text{SK}_{MU}}, \text{C}_2\right)$=$\left(g^{r\cdot x_{MU}} \mod n^2, h^r\,(1+mn) \mod n^2\right)$, $\text{C}_{SPU(H)}(m)$=$\left(\text{C}_1{}^{\text{SK}_{SPU}}, \text{C}_2\right)$=$\left(g^{r\cdot x_{SPU}} \mod n^2, h^r\,(1+mn) \mod n^2\right)$.
- **Combine**($\text{C}_{SPU(H)}(m)$,$\text{C}_{MU(H)}(m)$): Given $\text{C}_{MU(H)}(m)$=$\left(\text{C}_1{}^{\text{SK}_{MU}}, \text{C}_2\right)$ and $\text{C}_{SPU(H)}(m)$=$\left(\text{C}_1{}^{\text{SK}_{SPU}}, \text{C}_2\right)$, the decryption of the underlying message proceeds as follows:

$$\Delta\left(\frac{\text{C}_2}{\text{C}_1{}^{\text{SK}_{MU}} \cdot \text{C}_1{}^{\text{SK}_{SPU}}}\right) = m$$

where $\Delta(u) = \frac{(u-1) \mod n^2}{n}$, for all $u \in \left\{u < n^2 | u = 1 \mod n\right\}$.