HORNS: A Semi-perfectly Secret Homomorphic Encryption System

Mahadevan Gomathisankaran[a], Kamesh Namuduri[b], Akhilesh Tyagi[c]
[a] Dept. of Computer Science and Engineering, University of North Texas, Denton, Texas, USA
[b] Dept. of Electrical Engineering, University of North Texas, Denton, Texas, USA
[c]Dept. of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, USA

Abstract – With the increase in computation power and communication bandwidth it has become cheaper to aggregate the computation capabilities. This new model of computation is known as cloud computing. The security of cloud computing model will determine its universal applicability. In this paper we propose a novel homomorphic encryption system which can become a building block of securing the computations performed by cloud. We introduce the notion of semi-perfect secrecy and prove that our system is secure under this setup.

Key Words – Cloud Computing, Encryption function, Homomorhpic Encryption, Residue Number System.

I. Introduction

Cloud computing refers to the provision of computational resources on demand via a computer network. Cloud computing fundamentally allows for a functional separation between the resources used and the user's computing environment. This has enabled new business models as well as computing applications. DARPA, in its recent budget, has announced that it wants to create a cloud infrastructure which can be used to provide any-time and any-where access to the military users. Success of these new cloud applications depends on the security it can guarantee.

Homomorphic encryption systems allow computations to be performed on the ciphertext directly without having to decrypt them to plaintext. Such systems can form the building block of providing security to cloud computing applications. Recently, Gentry [2] has proposed a homomorphic encryption scheme of the form $c = pq + m$, where c is the ciphertext, m is the plaintext message, p is the secret key, and q is a random number. This encryption function is homomorphic with respect to addition, subtraction and multiplication. The relationship between c and m is that m is the residue of c with respect to modulus p. In other words, the encryption function is the inverse of residue operation. While this approach provides data security it does not exploit the parallelism inherently present in the cloud.

Residue Number System (RNS) is a well-known and a well-studied number theory system [6]. RNS has been used to achieve performance improvement as the arithmetic involves smaller numbers and can be done in parallel. RNS is defined in terms of a set of relatively prime moduli.

Let P denote the moduli set, then, $P = \{p_1, p_2, \dots, p_n\}$ and $\mathrm{GCD}(p_i, p_j) = 1$ for $i \neq j$. The dynamic range M_P of this RNS is given by

$$M_P = \prod_{i=1}^{n} p_i.$$

Any integer in the residue class Z_{M_P} can be represented in the RNS with the n-tuple, $X \to (x_{p_1}, x_{p_2}, \dots, x_{p_n})$, where, $x_{p_i} = x \bmod p_i$. The RNS representation is homomorphic with respect to addition, subtraction, and multiplication. In other words, $X \otimes Y \leftrightarrow (x_{p_1} \otimes y_{p_1}, \dots, x_{p_i} \otimes y_{p_i}, \dots, x_{p_n} \otimes y_{p_n})$.

The primary application of homomorphic encryption is in the field of *Cloud Computing*. In this set-up, the *cloud*, which is untrusted, is given the task of *computing* on a client's confidential data. The client can protect its confidential data from the untrusted *cloud* if it can encrypt its data using a homomorphic encryption function and use the *cloud* to do the computing on the encrypted data.

RNS creates multiple shares of a data and the operations on these shares are homomorphic. These two properties of RNS can be used to design a homomorphic encryption function for *cloud computing*. The application of RNS, so far, has been in the fields of computer arithmetic and digital signal processing.

In this paper we design a novel encryption system, HORNS, that exploits the inherent parallelism present in the cloud. HORNS uses residue number system to create multiple ciphertext shares. These shares themselves are homomorphic hence computations on them can be done in parallel. For example, the client can partition the shares and provide different partitions

Corresponding Author: Mahadevan Gomathisankaran, Email: mgomathi@unt.edu

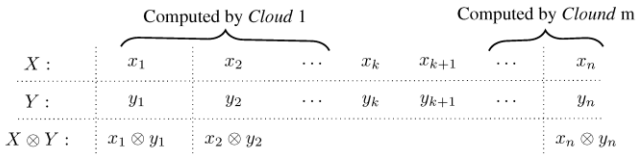

Fig. 1 Example HORNS scheme with multiple competing clouds doing the computations

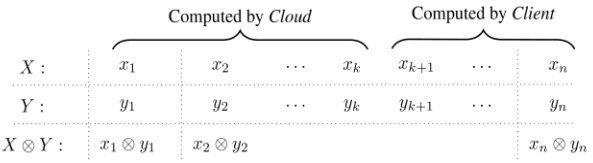

Fig. 2 Example HORNS scheme with client computing some partitions

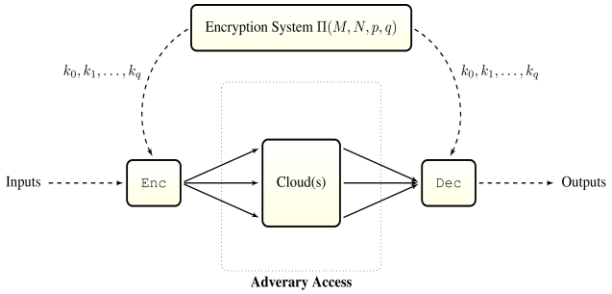

Fig. 3 Adversarial access in the HORNS encryption system

to different clouds, as shown in Figure 1. In such a scenario, all the clouds have to collude to do the attack. Another approach, as shown in Figure 2, is that the client can itself perform the computations with respect some partitions thus reducing the success probability of an attack dramatically.

The advantage of HORNS over other approaches is that it is semi-perfectly secret. In a perfectly secret encryption system the adversary does not learn any partial information about the plaintext from the ciphertext. Similarly, in a semi-perfectly secret encryption system the adversary does not gain any partial information about the plaintext from observing a subset of the ciphertext shares. We will formally define this notion and prove the security in the following sections.

The idea of splitting the information into multiple shares has been studied in the design of private circuits.

II. HORNS Encryption System $\prod(M, N, p, q)$

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. M, N, p, q are the parameters of the encryption system with $M, N, p, q \in \mathbb{N}$ and $M < N$, and $p < q$.

Message Space:

$$\mu = \{m | m \in \mathbb{N}_0, m < M\}$$

Key Space:

$$\kappa = \{ \ (k_0, \dots, k_i, \dots, k_j, \dots, k_q) \quad | \quad \forall 1 \le i < j \le q \ 1 < k_i < k_j < N, \ \forall 0 \le i < j \le q \ \mathrm{GCD}(k_i, k_j) = 1, \ \prod_{z=1}^{q} k_z > N, \prod_{z=1}^{q-p} k_z > k_0 \ge M \ \}$$

Ciphertext Space:

$$\psi = \{(c_1, \dots, c_i, \dots, c_q) | \ \forall \ 1 \le i \le q \ \ 0 \le c_i < k_i\}$$

Note that both key and ciphertext are q-tuples. We refer to each element in this q-tuple as a *share*. For example, the i^{th} share of ciphertext $c = (c_1, \dots, c_i, \dots, c_q)$ is c_i.

Generator Function (GEN):

The key generator function GEN chooses a key at random from the keyspace κ.

Encryption Function (ENC):

For any given message $m \in$ M and key $k = (k_0, \dots, k_q) \in$ K the encryption function proceeds as follows. **ENC** chooses a random number r_m such that $m + r_m k_0 < N$. Let $x_m = m + r_m k_0$ be the intermediate value. Then, $\forall 1 \le i \le q$ **ENC** computes i^{th} share of the ciphertext as $c_i = x_m \bmod k_i$.

Decryption Function (DEC):

The decryption function solves the system of simultaneous congruence equations $x_m \equiv c_i \bmod k_i \quad \forall \ 1 \le i \le q$ to find x_m. Then the message m is given by $m = x_m \bmod k_0$.

III. Semi-Perfect Secrecy

A perfectly secret encryption system is defined as one in which no adversary can learn any partial information about the message from any ciphertext. This condition can be expressed as

$P[\mu = m \mid \psi = c] = P[\mu = m]$.

Similarly, we define an encryption system as $(p, q)-$secret if no adversary can learn any partial information about the message from any p shares out of q of the ciphertext. For example, a perfectly secret system will be $(q, q)-$secret as per our definition. When $p < q$, such a system will be *semi*-perfectly secret.

Theorem 1: *The HORNS encryption system is* $(p, q)-$*secret.*

Proof: The adversarial access to the HORNS system is shown in Figure 3. The intuition behind the proof is as follows. We allow the adversary to observe any p

shares out the q ciphertext shares. In order to show that the adversary can not infer any partial information about the plaintext from these p shares we need to show that there is no correlation between them. In otherwords every possible value of the p shares should be representing any given plaintext and the vice-versa. In order to prove this theorem, we will construct the following experiment. This approach of proving the security is similar to the one proposed by Ishai et al. in [4].

A key $k \in \kappa$ is chosen at random. A message $m \in \mu$ is chosen at random. p out of q shares of the ciphertext are chosen at random. Then we show that the given p shares of cipher text can be generated from the chosen message m with the given key k. In other words, any message and key pair can generate any given p shares of the ciphertext.

Let S be the set of all p indices of the shares which were set randomly and $\bar{S}$ be the set of indices where not present in S. To prove (p, q) −secrecy we have to show that there exists a valid intermediate value $x_m = m + k_0 r_m$ such that $x_m \equiv c_i \bmod k_i \; \forall \, i \in S$, and $x_m < \prod_{z=1}^{q} k_z$.

The intermediate value x_m can be obtained by solving the following system of simultaneous congruence relations.

$$x_m \equiv c_i \bmod k_i \;\; \forall \, i \in S$$
$$x_m \equiv m \bmod k_0$$

Since the moduli in this system are co-prime with respect to each other, there exists a unique solution x_m (Chinese Remainder Theorem). The solution to this system x_m is bounded by the product of the moduli $k_0 \cdot \prod_{i \in S} k_i$.

Now we have to show that $k_0 \cdot \prod_{i \in S} k_i < \prod_{z=1}^{q} k_z$. From the properties of κ, k_1 to k_q are increasingly ordered and $\prod_{z=1}^{q-p} k_z > k_0$, we observe that

$$\prod_{i \in S} k_i \leq \prod_{z=q-p+q}^{q} k_z$$
$$\Rightarrow k_0 \cdot \prod_{i \in S} k_i \leq k_0 \cdot \prod_{z=q-p+1}^{q} k_z$$
$$\Rightarrow k_0 \cdot \prod_{i \in S} k_i < \prod_{z=1}^{q-p} k_z \cdot \prod_{z=q-p+1}^{q} k_z$$
$$\Rightarrow k_0 \cdot \prod_{i \in S} k_i < \prod_{z=1}^{q} k_z$$

IV. Homomorphism

The encryption system $\Pi(M, N, p, q)$ is homomorphic with respect to addition, subtraction and multiplication, provided messages and their intermediate values lie within the domain. Since the ciphertext c_i is residue of the intermediate value x_m with respect to the modulus k_i, the ciphertexts derive their homomorphic properties from the residue number system.

The system $\Pi(M, N, p, q)$ uses *random* r_m to convert a message to its intermediate value. This could cause issues with overflow as $x_{m_1} \otimes x_{m_2}$ may fall outside the valid range of 0 to $N - 1$. Redundancy can be used for overflow detection. Let m_1 and m_2 be the two messages such that $m_1 \otimes m_2$ to be performed in their encrypted form. Let message m_1 be encrypted to two different ciphertexts c_1 and c_2' by choosing two different intermediate values x_{m_1} and x_{m_2}'. Let message m_2 be encrypted to ciphertext c_2. Then, both $c_1 \otimes c_2$ and $c_1' \otimes c_2$ should yield the same result $m_1 \otimes m_2$ when decrypted. But if there is an overflow then the decrypted values of these ciphertexts will not match.

Binary Message Space:

When the message space is binary i.e., $\mu = \{0,1\}$, the encryption system exhibits the following homomorphisms.

$$a \text{ AND } b \quad \leftrightarrow \quad a_i \times b_i$$
$$a \text{ XOR } b \quad \leftrightarrow \quad a_i + b_i$$

Thus every binary operation in the message space can be achieved by operating on the individual share in the ciphertext space. Any arbitrary Boolean function represented in its canonical product of sum form can be implemented in HORNS.

V. Research Issues

In order to apply HORNS for encryption, the issues of confidentiality, integrity, and cloud collusion need to be addressed. These research issues in HORNS will be illustrated by the following example.

Let a and b represent two numbers that need to be added to produce the result, $c = a + b$, by a *cloud*. Let $P = \{p_1, p_2, \ldots, p_n\}$ be the moduli set that defines the RNS and M_P be its range such that $-\frac{M_P}{2} \leq a, b < \frac{M_P}{2}$. In the residue class Z_{M_P}, the range $[0, \frac{M_P}{2})$ represents positive numbers and the range $[\frac{M_P}{2}, M_P)$ represents the negative numbers. A negative number, say $-x$, is encoded as $M_P - x$, a representation analogous to 2′s complement. The client generates n shares of $a \rightarrow (a_{p_1}, a_{p_2}, \ldots, a_{p_n})$ and $b \rightarrow$

$(b_{p_1}, b_{p_2}, \dots, b_{p_n})$. The client requests the *cloud* to perform modular additions over p_i on the individual shares $c_{p_i} = a_{p_i} + b_{p_i}$ independently. The client reconstructs c from the n shares, c_{p_i}, it receives from the *cloud*.

Overflow and Sign Detection:

If $a, b \geq \frac{M_P}{4}$, then the result $c = a + b > \frac{M_P}{2}$ which will imply the result is a negative number while it is not.

Confidentiality and Cloud Collusion:

The *cloud* is given access to the data shares a_{p_i}, b_{p_i} and the modulus p_i. The client can employ several mechanisms so that the *cloud* does not get access to the moduli set P. For example, the client can partition the shares and provide different partitions to different *clouds*, as shown in Figure 1. In such a scenario, all the *clouds* have to collude to reconstruct the moduli set P. Another approach, as shown in Figure 2, is that the client can itself perform the computations with respect some partitions. In any case, a *cloud* should not be given access to all the moduli in P. Even with such a restriction, the confidentiality cannot be fully guaranteed.

The *cloud* can predict M_P as it most likely will be in the neighborhood of system word size. Then it can predict a as $a = x \cdot p_i + a_{p_i}$ such that $a < M_P$. The probability that the *cloud* can find a correctly is p_i/M_P. With the cloud given access to k moduli, this probability can be increased significantly to $\prod_{i=1}^{k} p_i / M_P$. Thus, to prevent the *cloud* from inferring the data the moduli need to be protected from it. But, RNS requires moduli to be given access to the cloud for computations. The solution is to transform the moduli and let the *cloud* operate on this transformed domain.

Integrity:

The *cloud* can provide a random result c without performing the actual addition $a + b$. In other words, the client should be able to detect if the result does not correspond to the actual computation requested.

VI. Possible Solutions

The following are the solutions we propose for the research issues presented in the previous section.

Overflow and Sign Detection:

Redundant RNS can be used for overflow and sign detection. The idea is to do the computation on multiple RN systems and compare the results. For example, let RNS_p and RNS_q be the two different RN systems used for computation. RNS_p is defined by the set of moduli $P = \{p_1, p_2, \dots, p_n\}$ and RNS_q is defined by $Q = \{q_1, q_2, \dots, q_n\}$. Let $M_p = \prod_{i=1}^{n} p_i$ and $M_q = \prod_{i=1}^{n} q_i$. Let Y_p and Y_q be converted results of RNS computation in RNS_p and RNS_q respectively. Then results are valid if and only if $Y_p = Y_q$ or $M_p - Y_p = M_q - Y_q$. This solution can detect if an overflow has occurred but not correct it. While earlier research has focused on improving the performance [3] by sharing the moduli between the redundant systems, we propose to increase the redundancy by not sharing the moduli. This can increase the security as will be shown later.

Modulus Confidentiality:

In order to do computations in RNS the modulus has to be provided to the *cloud*. But, this can in-turn reduce the security of the system as the cloud can infer the range M_p if it can acquire all the moduli of the RNS by some means. In order to prevent such a possibility we want to design the HORNS in such a way that the cloud should be able to operate on the data without having to know the actual modulus. This is similar to the data confidentiality requirement hence can be achieved in a similar way by adding *confusion* to the modulus. One way to add confusion is to transform the modulus by multiplying it with a random noise, $r_{p_i} \in R_P$, to the modulus $p_i \in P$, in such way that the computations are performed using the modulus $p_i' = p_i \cdot r_{p_i}$. The result from the cloud, which operates using modulus, p_i' can be converted back to modulus p_i as shown in Lemma 1.

Lemma 1: $(x \bmod p_i') \bmod p_i = x \bmod p_i$

Proof: Let $c = x \bmod p_i$ then, for some integer $x = k \cdot p_i + c$. Then for some integer l,

$$k \cdot r_{p_i} = k \cdot r_{p_i} + l \cdot r_{p_i} - l \cdot r_{p_i}$$
$$\Rightarrow k = r_{p_i} \cdot \frac{k - l}{r_{p_i}} + l$$
$$\Rightarrow x = p_i \cdot r_{p_i} \cdot \frac{k - l}{r_{p_i}} + l \cdot p_i + c$$
$$\Rightarrow x \bmod p_i' = l \cdot p_i + c$$
$$\Rightarrow (x \bmod p_i') \bmod p_i = c$$

Montgomery Representation Variations:

In modular arithmetic, multiplications are expensive. This is so since the size of the result doubles in multiplication. If many repeated multiplications occur, as in modular exponentiation needed in RSA, the size of the result gets out of hand very quickly. The result size can be kept in check if modular reduction is performed after each multiplication. Modular reduction with its trial division is an expensive operation.

Montgomery [5] came up with an interesting scheme to maintain the multiplication products with sizes in check. For modular arithmetic modulo N, another modulus/radix $R > N$ is chosen such that R and N are coprime. R is chosen to be a power of 2 say 2^k so that multiplication and division by R on a typical machine are just a left shift or right shift by k bit positions. Somewhat similar to Lemma 1, modulus operations performed in R still maintain the modular reductions in N valid. The Montgomery representation converts each a into aR. Addition works fine as is. For multiplication of aR and bR, the size doubles. Intuitively a division by R such that the modular reduction with respect to N can be carried out later will restore the operand size. Montgomery proposed the following reduction (Montgomery reduction) for a value $T-$ result of a multiplication: $(m \leftarrow (T \bmod R) \cdot N' \bmod R$ and $t \leftarrow \frac{T+mN}{R}$. The value t retains all the bits relevant for modular reduction with respect to N and is yet of the right size. N' is chosen such that $RR^{-1} - NN' = 1$.

Intuitively, we would like most arithmetic within the cloud to proceed in Montgomery domain with radix R (or moduli r_i). With appropriately chosen Montgomery moduli, the computation can be efficient. In the current form of Montgomery reduction, though, the reduction expression needs both N and its inverse N'. Hence if the reduction were to be performed within the untrusted cloud node, both the moduli and its multiplicative inverse need to be revealed. This goes counter to our preceding argument for modulus confidentiality.

The following are the open research questions within the context of Montgomery reduction.

1. Current Montgomery representation maintains a value t within one subtraction of the correct value. If we allow this size to be larger, can we get away with revealing less of N and N'? There is likely to be a trade-off in computation efficiency of the modular arithmetic and its privacy. Bajard et al. [1] merge RNS and Montgomery field in an interesting manner.
2. Another option is not to perform Montgomery reduction at the cloud node. For efficiency, up to certain number of multiplications, say k, no reduction is performed. This allows the operands to grow into k times as many bits as needed minimally. At these $k-$multiplication epochs, a synchronization with a trusted processor is forced wherein the Montgomery reduction and/or modular reduction is performed.

Cloud Collusion:

One way to protect the confidentiality of modulus is to distribute the computations to different clouds as shown in Figure 1. In this section, we devise a strategy for allocating a subset of the moduli to each *cloud* in such a way that it will minimize the impact on security due to collusion. Let $P = \{p_1, p_2, \dots, p_n\}$ be the moduli set and $|p_i| > S_P^n$, where S_P^n is the minimum size of modulus. Let M_P be the range of this RNS. Let k be the number of moduli given to a cloud for execution. Then, the maximum probability that any *cloud* can infer the data is S_P^k/M_P and $M_P \approx S_P^n$.

Let $f(n,k)$ represent the probability of success for any *cloud* with k out of n moduli to infer the data. Then:

$$f(n,k) = \frac{1}{S_P^{n-k}}$$

A simple way to distribute the moduli to the *clouds* is to create disjoint subsets of P and distribute it to all the clouds. In this simple approach, collusion among two clouds will exponentially increase the success probability, or in other words:

$$f(n, k_1 + k_2) = \frac{1}{S_P^{n-(k_1+k_2)}}$$

This can be achieved by the following redundant scheme in which each cloud will be given $k = q + l$ moduli, where q moduli are distinct and non-overlapping, and l moduli are redundant and overlapping with the moduli assigned to other clouds.

Thus, if two clouds collude, the number of moduli they can gather is not $2k$, instead, it is less than $2k$. Of course, if all clouds collude, they will have all the moduli required to break the RNS system.

Proposition 1: To recreate the RNS system completely, a collusion of at least $n/q + l$ clouds is needed. As long this doesn't happen, the RNS system is secure.

Proof: Consider a scenario, where y clouds collude to recreate RNS system. The collusion gives rise to $y \cdot q$ distinct moduli and $y \cdot l$ redundant moduli. In best case scenario, $y \cdot l$ moduli do not overlap with $y \cdot q$ moduli. The group of y clouds will recreate all the moduli if $n = y \cdot l + y \cdot q$. In other words, $y \geq \frac{n}{q+l}$ in order to recreate the RNS system.

Proposition 2: Assuming that $\geq \frac{n}{q+l}$, the probability of finding $(n - y \cdot q)$ distinct moduli from the $y \cdot l$ redundant moduli is given by $\frac{\binom{y \cdot q}{n-y \cdot q}}{y \cdot \binom{n-q}{l}}$.

Proof: The numerator gives the number of ways of selecting $(n - y \cdot q)$ moduli from $y \cdot q$ distinct moduli and the denominator specifies the number of

ways of selecting any l number of moduli from $(n-q)$ moduli and repeating this step for y times.

Integrity:

The integrity requirement of HORNS is to verify that the result produced by the *cloud* is indeed valid, in other words, the *cloud* has not tampered the moduli or moduli in generating the result. The solutions of overflow detection and confidentiality together provide the required integrity protection.

VII. Cloud Computing Framework

In this section, we provide a conceptual *cloud computing* framework based on HORNS. Given a program P to be dispatched to an untrusted could, there are many options for transforming it. Moreover, there are at least two possible trust models. (1) Cloud is completely untrusted. Hence any trusted operations need to be performed by the client node itself. (2) Cloud contains a kernel trusted processor. Particularly, for each group of k untrusted cores there could be one trusted core. The trusted core could have TPM [7] or Arc3D [8] like architecture. The main point to note is that this node necessarily would have significantly lower throughput than the computation cores due to its cryptographic overhead. Hence, in this model, we must carefully design the throughput load of the trusted core to be some fraction of the throughput of untrusted computing cores (say $\left(\frac{1}{10}\right)^{th}$ to $\left(\frac{1}{100}\right)^{th}$). We next describe program transformations.

RNS Coding: The protected computation of program P is threaded into k threads. All the key variables to be protected are split into k moduli using the RNS schema of HORNS. Each thread would have identical control flow, but different data. In fact, data-parallel model of NVidia Fermi GPU, CUDA, works well for such a schema. In GPU terminology, each original thread becomes a $k-$way Warp. The threads within a Warp are forced to sync either for Montgomery reduction, or for HORNS moduli reduction, or for a branch.

Threading Control: The preceding discussion highlights the fact in a secure cloud environment, some of the thread scheduling flexibility must be given to the client. A client should be able to specify scheduling parameters loosely. How this specification must be incorporated into Cloud protocols, and what degree of scheduling specifications cane be entrusted with the client is a topic for cloud computing research.

Root of trust at Cloud: It would be more efficient to have a trusted node at cloud. How would such a node be rooted in trust is still an open question.

Validation: There are many possible validation mechanisms. To name a few, the redundancy schema of Section VI could deploy multiple moduli sets for each thread. The results from all the k threads can be validated at predetermined validation points by a trusted processor. Along a similar validation schema, a $(k+1)$ st hidden modulus could be selected. The corresponding residue can be kept as a secret with the trusted processor. When the k results from the k threads corresponding to the k moduli come back, their consistency with respect to the hidden residue can be validated with Chinese Remainder Theorem (CRT). There are many other in-between validation schema.

VIII. Cloud Computing Framework

Cloud computing refers to the provision of computational resources on demand via a computer network. With the increase in computation power and communication bandwidth it has become cheaper to aggregate the computation capabilities provision it as and when required over the network. The security of cloud computing model will determine its universal applicability. In this paper we design a novel encryption system, HORNS, that exploits the inherent parallelism present in the cloud. HORNS uses residue number system to create multiple ciphertext shares. We have defined a new notion of secrecy known as *semi-perfect secrecy* that is application to HORNS. We have also proven that, in Theorem 1, HORNS is semi-perfectly secret. We have discussed the research issues involved in applying HORNS in a Cloud Computing framework.

REFERENCES

[1] BAJARD, J.-C., DIDIER, L.-S., and KORNERUP, P. An RNS montgomery modular multiplication algorithm. IEEE TRANSACTIONS ON COMPUTERS 47, 7 (1998), 766–776.

[2] GENTRY, C. Computing arbitrary functions of encrypted data. Commun. ACM 53, 3 (2010), 97–105.

[3] GREGORY, R. T., and MATULA, D. W. Base conversion in residue number systems. Residue number system arithmetic: modern applications in digital signal processing (1986), 22–30.

[4] ISHAI, Y., SAHAI, A., and WAGNER, D. Private circuits: Securing hardware against probing attacks. In Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara,

California, USA, August 17-21, 2003, Proceedings (2003), vol. 2729 of Lecture Notes in Computer Science, Springer, pp. 463–481.
[5] MONTGOMERY, P. L. Modular multiplication without trial division. Mathematics of Computation 44, 170 (April 1985), 519–521.
[6] SODERSTRAND, M. A., JENKINS, W. K., JULLIEN, G. A., and TAYLOR, F. J., Eds. Residue number system arithmetic: modern applications in digital signal processing. IEEE Press, Piscataway, NJ, USA, 1986.
[7] TRUSTED COMPUTING GROUP. TPM Main Specification Level 2 Version 1.2, Revision 103.
[8] GOMATHISANKARAN, M. and TYAGI, A. "Architecture support for 3D obfuscation," Computers, IEEE Transactions on , vol.55, no.5, pp.497,507, May 2006