Input Verifiability in Delegation of Computation with Embedded Proofs

Tanvi Sharma, Rohit Singh
Department of Computer Science and Engineering
Career Point University, Kota, India

***Abstract* - Emerging paradigm of pay-per-use through cloud computing has led to outsourcing of many business-critical functions to cloud service providers. Delegation of Computation is useful concept, only until the results of computation can be verified. Such verifiability has been achieved through interactive proof systems, leading to high communication cost. Other methods involve cryptographic primitives which impose high computation cost, rendering it unsuitable for mobile clients. We propose here a protocol with reduced communication cost for light-weight devices, utilizing cryptographic primitives to generate proofs. The proof-generating function is incorporated into the delegated function to make it fool-proof.**

***Index Terms* - verifiable delegation of computation, interactive proofs**

I. INTRODUCTION

Surprisingly powerful protocols for verifiable computation were discovered within the computer science theory community several decades ago, in the form of interactive proofs (IPs), interactive arguments (IAs) and probabilistically checkable proofs (PCPs)[1,2]. In these protocols, the prover P solves a problem using her (possibly vast) computational resources, and tells V the answer. P and V then have a conversation, i.e., they engage in a randomized protocol involving the exchange of one or more messages. During this conversation, P's goal is to convince V that the answer is correct. Results quantifying the power of IPs, IAs, and PCPs represent some of the most celebrated results in all of computational complexity theory[1,2], but until recently they were mainly of theoretical interest, far too inefficient for actual deployment. In fact, the main applications of these results have traditionally been in negative applications – showing that many problems are just as hard to approximate as they are to solve exactly.

However, the surging popularity of cloud computing has brought renewed interest in positive applications of protocols for verifiable computation. A typical motivating scenario is as follows. A business processes billions or trillions of transactions a day. The volume is sufficiently high that the business cannot or will not store and process the transactions on its own. Instead, it offloads the processing to a commercial cloud computing service. The offloading of any computation raises issues of trust: the business may be concerned about relatively benign events like dropped transactions, buggy algorithms, or uncorrected hardware faults, or the business may be more paranoid and fear that the cloud operator is deliberately deceptive or has been externally compromised. Either way, each time the business poses a query to the cloud, the business may demand that the cloud also provide a guarantee that the returned answer is correct. This is precisely what protocols for verifiable computation accomplish, with the cloud acting as the prover in the protocol, and the business acting as the verifier.

Cloud computing has a lot to offer in terms of computational power for lightweight and mobile computing devices. The pay-per-use is a prominent feature leading to growing popularity, as companies and users reduce their computing assets and turn to weaker computing devices; thereby delegating a number and variety of computations to cloud service providers. Yet the monetary benefits related to such services raise a concern about trust. The major risk here is that the business critical computations are being performed remotely by untrusted parties that may be error-prone or even malicious. This motivates exploring methods for delegating computations reliably: a weak client delegates his computation to a powerful server. After the server returns the result of the computation, the client should be able to verify the correctness of that result using considerably less resources than required to actually perform the computation from scratch. Users may benefit a lot by outsourcing the computational intensive functions to the cloud, the data can well be secured through encryption, but can the computation be performed over encrypted data? Even if it is practically feasible, can the user trust that the results returned by the cloud are correct? Verifiable delegation of computation (VDoC) is a possible solution.

VDoC is a paradigm where a delegator (commonly a lightweight device) delegates data and function to be computed over it to a worker (such as cloud service provider). For security purposes this data may be encrypted. The worker returns the result of computation along with a proof that the result is correct. Delegator processes the proof and accordingly accepts or rejects the result. The major requirement over here is that cost of verification process should be much less than actual computation. A few protocols, implementations and results in the field of VDoC have been produced. Most of the work in this field has been directed towards efficient verification of result of a computation, or towards secure computation thereby using cryptographic primitives. Yet, in the light of monetary benefits related to outsourced computations, clouds have a motivation to provide imprecise results or some previously calculated results. Hence, we should steer the direction of research towards ensuring that the computation is indeed done over the provided input. Existing schemes which accomplish this rely on the soundness of cryptographic primitives involved, or

they depend on heavy communication between delegator and worker. Thatswhy, effort to decrease the communication costs is also a possible research area.

II. Related Work

There exists a vast literature for the notion of verifiable computation, ranging from works for arbitrary computations [3,4, 5, 6, 7, 8, 9, 10, 11] to works for specific classes of computations [12, 13, 14, 15]. In verifiable computation, a client wants to delegate a computationally heavy task to a remote server while being able to verify the result in a very efficient way. In the definition proposed by Gennaro, Gentry, and Parno [6] (and later adopted in several works, e.g., [7, 9, 12, 13]), to delegate the computation of f(D), the client has to compute an encoding τ_D of D, which depends on the function f. However, if we want to choose f after outsourcing D, the computation of τ_D is no longer possible. Alternatively, one could keep the entire input D locally and then compute τ_D from D and f, which would yield a running time proportional to the input size. In other work (e.g.,[4.16]) the efficiency requirement for a client is to run in time poly(n, log T), when delegating a function f that runs in time T and takes inputs of size n. The notion of multi-function verifiable computation proposed in [9 and 17] is close to our model, in that a client can delegate the computation of many functions on the same input D, while being able to efficiently verify the results.

Another interesting line of work in this area recently proposed efficient systems for verifiable computation [18,19,20,21]. The proposed solutions also work in a model where the client needs to know the input of the computation, and it also has to engage in an interactive protocol with the server in order to verify the results. In contrast, our work considers a completely non-interactive setting in which the proof is transferred from the server to the client in a single round of communication, similar to [22].

III. Proposed Protocol and Scheme

A. Protocol

We propose to send the challenge to the worker in the form of a choice vector. The vector selects only one out of all input data provided. The worker is required to compute an extra function over this selected data point. The extra functionality is built into the delegated function. We employ a special cryptographic primitive (described in further section) so that the result of extra functionality can be verified by a single step calculation by the delegator. If this result is verified, it implies that the delegated function was indeed computed over the provided input data.

Formally, the steps are:

Phase I: Delegation

Step 1: Input Preparation

For data input be the vector of inputs $\langle d_1, d_2, \dots, d_n \rangle$, and choice be the vector of bits such that only one of them is 1, all other 0's. Size of choice is also n. Now, encrypt bits of choice using Encryption method E. Let choice be $\langle b_1, b_2, \dots, b_n \rangle$ which after encryption becomes $\langle c_1, c_2, \dots, c_n \rangle$, that is $Enc(b_i) \rightarrow c_i, i = 1,2, \dots, n$

Step 2: Send Input

Send ($\langle d_1, d_2, \dots, d_n \rangle$, $\langle c_1, c_2, \dots, c_n \rangle$,*key*) to Worker

Phase II: Computation

Step 1: Worker computes $f'(\langle d_1, d_2, \dots, d_n \rangle) \rightarrow result$. f' has additional functionality, that is, besides computing $f(\langle d_1, d_2, \dots, d_n \rangle)$, it also computes $g(\langle d_1, d_2, \dots, d_n \rangle, \langle c_1, c_2, \dots, c_n \rangle) \rightarrow proof$ calculated as: $proof = \sum_{i=1}^{n} (d_i * c_i \bmod key)$.

Step 2: Worker sends result and proof to the delegator.

Phase III: Verification

Delegator verifies $proof \bmod s = d_k \bmod s$ where $b_k = 1$, if yes then accept result, else reject.

B. Cryptographic Primitives

For the purpose of the proposed protocol we suggest a simple homomorphic encryption primitive. The basic idea is to use probabilistic encryption for security. We suggest a method to randomize a plaintext from a ring Zn to obtain an integer ciphertext. Hence, addition and multiplication within the ring are homomorphic to ordinary integer addition and multiplication operations. The method is similar to the somewhat homomorphic scheme of [23] with symmetric keys. But we modify it to make it simple yet secure.

Encryption

Step 1: Choose an integer y equivalent to plaintext x modulo secret key s, that is $x \equiv y \bmod s$

Step 2: Select a random number r>key

Step 3: Compute $c = y + key * r$

Step 4: Output c as ciphertext

Parameter values: To enable proper modular operations, we maintain key>s and r>key.

C. Examples

Example 1

We present here how to encrypt a bit using the proposed scheme, under the parameters s = 19, $\mu = 3$. Let key = 469 (a number of length μ^2 bits). Suppose the plaintext is 1, that is x = 1. Selecting $x \equiv y \bmod s$, we have y=153. Let the random number be 412.

Thus, encryption of 1 is, c = y + key*r = 153 + 469*412 = 193381.

The correctness can be seen as, $x = c \bmod key \bmod s = 193381 \bmod 469 \bmod 19 = 153 \bmod 19 = 1$

Example 2

We construct a toy example of the VDoC protocol, taking number of inputs n=4 and the parameter μ = 5. Suppose the input data vector is ⟨21,4,17,9⟩, and the choice vector is ⟨0,1,0,0⟩.

Input preparation: We encrypt the choice vector with key=29342513, under s=21, as ⟨796472263254181, 957786953793533, 828249031132577, 536273656014965⟩

Proof Computation: The worker knows key = 29342513, it computes $proof = \sum_{i=1}^{n}((d_i * c_i) \bmod key) = (21 * 796472263254181) \bmod 29342513 + (4 * 957786953793533) \bmod 29342513 + (17 * 828249031132577) \bmod 29342513 + (9 * 536273656014965) \bmod 29342513$
$= 1764 + 256 + 0 + 756 = 2776$

Verification: The delegator computes $proof \bmod s = 2776 \bmod 21 \equiv 4$, which is same as the input number at which position the bit 1 was sent in choice vector.

D. Benefits

The proposed scheme and protocol have several benefits over other VDoC protocols:

- Tamper-proof : The extra functionality is built into the delegated function. Hence, the worker cannot cheat by tampering with the proof function. No rejection problem: Most of the interactive VDoC schemes suffer from the rejection problem, which means that the worker might know whether the result was accepted or rejected and take counter-steps for future. But, in our proposal the interaction of the delegator and worker is limited. The worker never comes to know whether the result was accepted or rejected.
- Low communication cost: We have only three step communication protocol, which means lesser communication cost.
- There are some issues of concern around our proposal, which could be explained as follows:
- Overhead of modification of functions – Our proposal requires modification in the function itself. A question about the overhead incurred may arise. We counter this by stating that many VDoC protocols use FHE, hence the functions have to be modified for homomorphic operations. These modifications are much costlier than our suggestion.
- Long input size – including the choice vector with the input data raises concern about the length of input. Our protocol can be modified to have a shorter choice vector.
- Lightweight device as delegator – Such delegator requires lightweight operations. The input preparation can be done in collaboration with a third party. The scheme is secure even if same choice vector is used many times. Thus, amortized computation cost is very low.

E. Mathematical analysis

Length of input sent to the worker depends on the actual data to be input for the delegated function. Say the largest d_i is of length μ bits. Then, any m_i is of maximum length μ bits. To have proper modulo operations, we have to select key of length μ^2 bits. Thus, making every c_i μ^4 bits long. For an input of μ bits, an extra input of μ^4 bit length is sent. Communication overhead can be computed as the ratio of lengths of actual input and the input sent to worker.

Length of input sent to worker $= n * (\mu + \mu^4) + \mu^2$

Communication overhead = $\frac{n*(\mu+\mu^4)+\mu^2}{n*\mu} = O(\mu^3)$

Thus, communication overhead is cubic in bit length of input.

Computation overhead includes cost of proof calculation on behalf of worker. This computational cost is governed by most costly operation, that is, multiplication of 2n numbers d_i and $c_i \bmod key$. The bit lengths are μ and μ^2 respectively. Hence, cost of multiplication is $O(\mu^3)$ bit operations or O(n) integer operations. Thus, computation overhead is linear in input size.

Verification procedure consists only of modulo divisions. Time complexity of verification is $O(\mu^4)$ bit operations or O(1) integer operations.

Cost of input preparation: Since Delegator has to prepare the input; such cost should also be considered as computation overhead on Delegator's part. The input preparation involves encryption of n numbers. Hence, the cost is $O(\mu^4)$ bit operations or O(n) integer operations. This cost is linear in input size, so it should be reduced in case of large input dataset. Reduction can be achieved by selecting a choice vector for only first k data-items.

The proposed scheme derives its security from the security of the involved encryption scheme. The key is known to the worker, but the random number involved in encryption is not known. Also, the number s is not known. Hence, the worker cannot decrypt the choice vector only by knowing the key. Under the assumption that μ is known s can be guessed correctly with a negligible probability of $\approx \frac{b \ln 2}{2^{(b-1)+(b^2-1)}}$, where b =μ. Let μ=8 bit, we get probability of guessing s correctly = $(8*0.693)/2^{70}$ = $4.69*10^{-21}$, which is almost zero.

IV. IMPLEMENTATION RESULTS

The runtimes recorded for various steps of the protocol are shown in tables below. The parameter μ is varied as increasing value, and corresponding growth of the run time is noted. The recorded runtime in nanoseconds is shown in Table 1 for different values of n. It can be observed that Encryption process requires much more time than proof generation or verification.

Table 1 Runtime in nanoseconds for various values of n and μ

μ	n = 4			n = 5			n = 6		
	Encryption	***Proof Generation***	***Verification***	***Encryption***	***Proof Generation***	***Verification***	***Encryption***	***Proof Generation***	***Verification***
4	1087551	27461	80571	1318700	34099	83588	2971754	34401	82985
5	2288263	28365	83381	2739397	43152	90922	3573467	43756	99704
6	2633177	34703	137000	3546007	58926	148769	5197854	68500	172909
7	3491388	52205	148467	3842338	64275	155408	5542466	77553	152993
8	4406934	54016	155105	4817331	66991	154502	6518063	86606	144846
9	5866558	57939	170796	6888627	70612	178945	8496241	90058	109540
10	9660614	59447	207311	11644701	72423	193430	13859936	98163	189507

To study the impact of both parameters on runtimes, we plot the results as shown in Fig 1, 2 and 3. Cost of encryption increases with both μ and n. This is emphasized by the plot shown in Fig 1. It plots the encryption time on y-axis, value of n on x-axis, for different values of μ. The growth is polynomial in n, and same for every value of μ. Cost of encryption can be reduced if same choice vector is used for many inputs, only proof computation and verification will be performed every time. Thus, the per–session cost of input preparation will be reduced. Also, we can reduce the size of choice vector thereby reducing number of operations in input preparation (and hence time of encryption) and proof generation. Cost of proof generation is very small, thereby making it unobjectionable for worker. As we can observe cost of verification varies only according to value of μ and does not depend on n for many continuous values of n. The growth can be seen in Fig 2, which is a plot of runtime for proof generation against values of n on x-axis. The three different plots correspond to different values of μ. The runtime for verification follows a step growth as can be seen in Fig 3. The sharp increase for every value of μ appears after n=6. The sudden increase can be attributed to implementation. But, the major observation is that the values are almost same for every μ, which implies that it doesn't depend on μ.

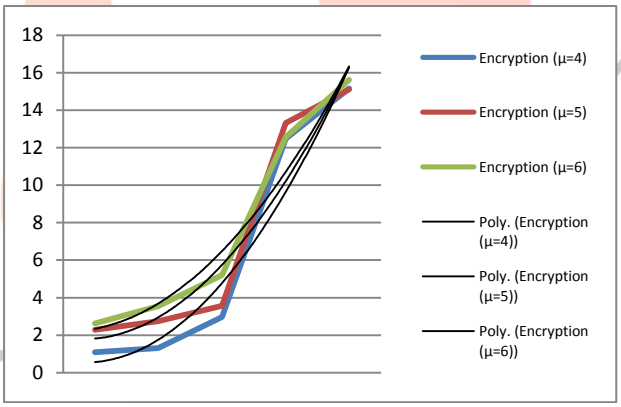

Fig 1 Plot of Encryption time vs value of n for different values of μ

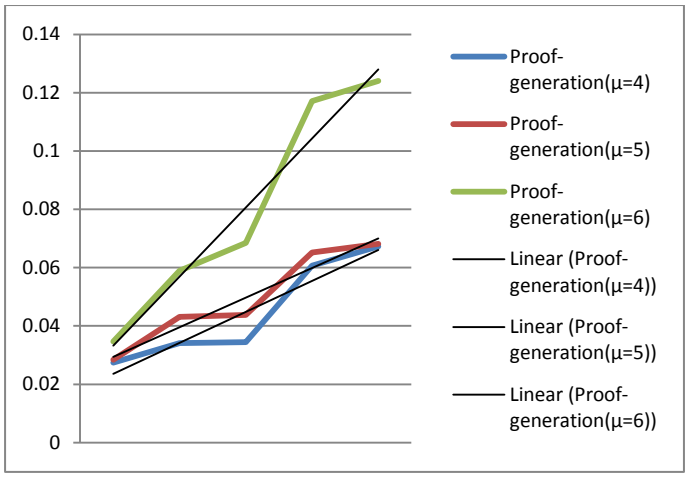

Fig 2 Plot of Proof-generation time vs value of n for different values of μ

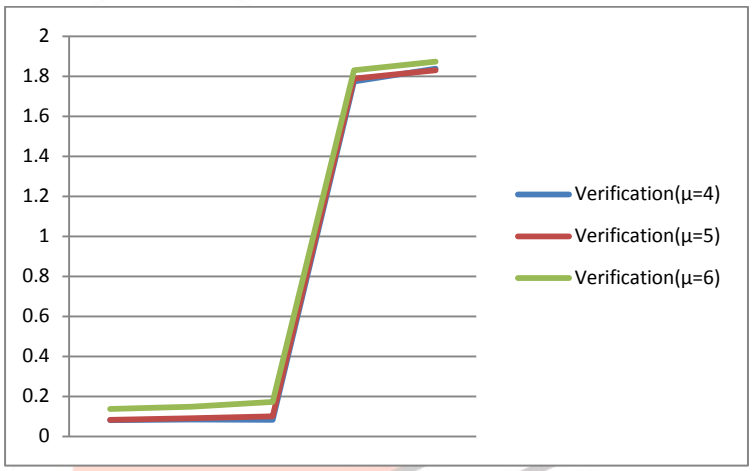

Fig 3 Plot of Verification time vs value of n for different values of μ

V. Conclusion

Our proposal presents a new method of input verifiability for delegated computation, applicable in cloud computing and mobile cloud computing. Based on the interactive methods of proofs and arguments, we have reduced the inherent communication cost of the interactive protocols. A major issue with the soundness of any interactive delegation schemes is the Verifier Rejection problem. In such schemes, it is required that the cheating worker does not learn whether the verifier accepts or rejects previous proofs from the prover. Our proposal aims at removing this problem by limiting the interaction between the delegator and verifier. Many schemes use FHE as a primitive in their protocol, while practical feasibility of a FHE scheme is still under consideration. We have used the homomorphic encryption for hiding the choice vector, in a novel way to achieve its benefits to make it tamper-proof while keeping away from the drawbacks of infeasibility. Apart from these, the benefits of low computation cost of verification and scope of amortizing the total cost of tag preparation and verification make our protocol attractive for practical applications.

Future research can be directed towards studying the implementation of the proposed protocol in real application environment.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. J. ACM, 45(3):501-555, May 1998.

[2] S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of np. J. ACM, 45(1):70-122, Jan, 1998.

[3] J. Kilian. "Improved efficient arguments." In Proceedings of CRYPTO, 1995.

[4] S. Micali. "*CS proofs*." In Proceedings of the IEEE Symposium on Foundations of Computer Science, 1994.

[5] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: interactive proofs for muggles. In STOC, pages 113–122, 2008.

[6] R. Gennaro, C. Gentry, and B. Parno. *"Non-interactive verifiable computing: Outsourcing computation to untrusted workers.*" In Tal Rabin, editor, CRYPTO, volume 6223 of Lecture Notes in Computer Science, pages 465-482. Springer, 2010.

[7] K.-M. Chung, Y. Kalai, and S. P. Vadhan. "*Improved delegation of computation using fully homomorphic encryption.*" In T. Rabin, editor, Advances in Cryptology – CRYPTO 2010, volume 6223 of Lecture Notes in Computer Science, pp 483–501, Springer.

[8] B. Applebaum, Y. Ishai and E. Kushilevitz. "From Secrecy to Soundness: Efficient Verification via Secure Computation". Automata, Languages and Programming. Lecture Notes in Computer Science Volume 6198, 2010, pp 152-163.
[9] B. Parno, M. Raykova and V. Vaikuntanathan. "*How to Delegate and Verify in Public: Verifiable Computation from Attribute-based Encryption.*" IACR Cryptology ePrint Archive, 2011:597. 2011.
[10] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. *Quadratic span programs and succinct NIZKs without PCPs*. In Eurocrypt '13: Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2013. Also in Cryptology ePrint Archive, Report 2012/215, http://eprint.iacr.org/2012/215.
[11] B. Parno, C.Gentry, J.Howell and M.Raykova. "*Pinocchio: Nearly Practical Verifiable Computation.*" IEEE Symposium on Security and Privacy, 2013.
[12] S. Benabbas, R. Gennaro, and Y. Vahlis. *Verifiable delegation of computation over large datasets*. In P. Rogaway, editor, CRYPTO 2011, volume 6841 of LNCS, pages 111{131, Santa Barbara, CA, USA, Aug. 14-18, 2011. Springer, Berlin, Germany.
[13] D. Fiore and R. Gennaro. *Publicly verifiable delegation of large polynomials and matrix computations, with applications.* In 2012 ACM Conference on Computer and Communication Security. ACM Press, October 2012. Full version available at http://eprint.iacr.org/2012/281.
[14] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation, 2013.
[15] D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis. *Algebraic (trapdoor) one way functions and their applications*. In TCC 2013, volume 7785 of LNCS, pages 680-699. Springer, Berlin, Germany, 2013.
[16] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. *Delegating computation: interactive proofs for muggles*. In R. E. Ladner and C. Dwork, editors, 40th ACM STOC, pages 113-122, Victoria, British Columbia, Canada, May 17-20, 2008. ACM Press.
[17] S.G. Choi, J.Katz, R. Kumaresan and C. Cid. "*Multi-User Non-Interactive Verifiable Computation*." 2012.
[18] Srinath Setty, Richard McPherson, Andrew J. Blumberg, and Michael Walfish. *Making Argument Systems for Outsourced Computation Practical (Sometimes)*. In Network and Distributed System Security Symposium (NDSS), Feb. 2012.
[19] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish. *Taking proof-based verified computation a few steps closer to practicality.* In USENIX Security, pages 253–268, Aug. 2012.
[20] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish. *Resolving the conflict between generality and plausibility in verified computation.* In European Conference on Computer Systems (EuroSys), pages 71–84, Apr. 2013.
[21] Victor Vu, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. IEEE Symposium on Security and Privacy, Oakland 2013.
[22] M. Backes, D.Fiore and R.M. Reischuk. "*Verifiable Delegation of Computation on Outsourced Data.*" IACR Cryptology ePrint Archive, 2013:469, 2013.
[23] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. "Fully homomorphic encryption over the integers", Proceedings of Eurocrypt-10*, Lecture Notes in Computer Science, vol 6110,. Springer*, pp 24-43, 2010.