A Survey of Cryptographic Approaches to Securing Big-Data Analytics in the Cloud

Sophia Yakoubov, Vijay Gadepally, Nabil Schear, Emily Shen, Arkady Yerukhimovich
MIT Lincoln Laboratory
Lexington, MA 02421
{sophia.yakoubov, vijayg, nabil, emily.shen, arkady}@ll.mit.edu

***Abstract*—The growing demand for cloud computing motivates the need to study the security of data received, stored, processed, and transmitted by a cloud. In this paper, we present a framework for such a study. We introduce a cloud computing model that captures a rich class of big-data use-cases and allows reasoning about relevant threats and security goals. We then survey three cryptographic techniques – homomorphic encryption, verifiable computation, and multi-party computation – that can be used to achieve these goals. We describe the cryptographic techniques in the context of our cloud model and highlight the differences in performance cost associated with each.**

I. Introduction

In today's data-centric world, big-data processing and analytics have become critical to most enterprise and government applications. Thus, there is a need for an appropriate big-data infrastructure that supports storage and processing on a massive scale.

Cloud computing has become the tool of choice for big-data processing and analytics due to its reduced cost, broad network access, elasticity, resource pooling, and measured service [1]. Cloud computing enables consumers to store and analyze their data using shared computing resources while easily handling fluctuations in the volume and velocity of the data. However, cloud computing comes with risks. The shared compute infrastructure introduces many security concerns not present in more traditional computing architectures. The cloud provider and tenants may be untrusted entities who try to tamper with data storage or computation. These concerns motivate the need for a novel framework for analyzing cloud computing security, as well as for the use of cryptographic tools to address cloud computing security goals. In this paper, we propose a computation model of the cloud for big-data applications, and survey existing cryptographic tools using this model.

Our primary contributions are:

- A general computation model that captures a large class of big-data use-cases in the cloud,
- A description of relevant security threats, and
- A survey of cryptographic tools that address these security threats, and their current performance overhead.

This work is sponsored by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, recommendations and conclusions are those of the authors and are not necessarily endorsed by the United States Government.

The remainder of this paper is organized as follows. In Section II, we present a model of big-data analytics in the cloud and introduce genomic sequencing as a motivating application. In Section III describes the security goals and the threat model. Then, in Section IV, we describe several cryptographic techniques that can be used to address these security goals in various cloud deployments. Finally, in Section V, we conclude and describe some directions for future research in the area of secure cloud computation.

II. A Model for Big-Data Analytics in the Cloud

In this section, we present a general computational model of the cloud that allows reasoning about a wide variety of big-data analytics applications. In this model, we categorize cloud compute nodes by their roles in the big-data analytics pipeline. Extending the notation of Bogdanov et al. [2], we define the following types of nodes:

- **I** denotes an *input node* supplying raw data for the application. Input nodes include sensors capturing data and machines used to enter client data.
- **C** denotes a *compute node* whose role is to perform the computation for the application. Compute nodes include *ingestion* nodes, which refine the input data to get it ready for analysis, and *enrichment* nodes, which perform the actual analysis.
- **S** denotes a *storage node* whose role is to store data between computations. Storage nodes store both the original inputs and the computation outputs.
- **R** denotes a *result node* which receives the output of some computation, and either makes automated decisions based on that output or conveys the output to a client.

Additionally, X^+ (where $X \in \{\mathbf{I}, \mathbf{C}, \mathbf{S}, \mathbf{R}\}$) denotes a set of one or more (possibly communicating) nodes of type X.

Figure 1 depicts a cloud architecture for big-data analytics using the above terminology. This model can be used to describe a wide variety of big-data applications.

As an example, consider an application in which the cloud stores and correlates sensitive genomics data with the goal of identifying a specific biological sequence, as described by Kepner et al. [3]. In this scenario, the cloud stores and correlates billions of reference genomic sequences. An agency

978-1-4799-6233-4/14/$31.00 ©2014 IEEE

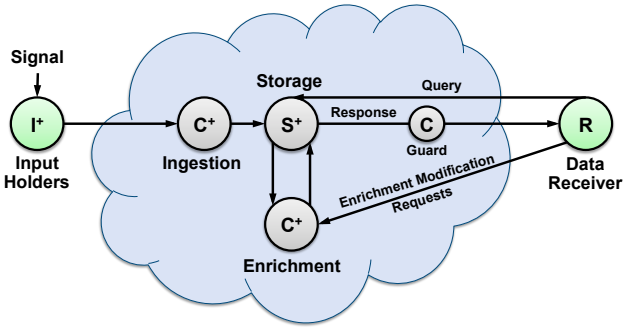


Fig. 1: A cloud architecture for big-data analytics. Here **I** refers to an input node, **C** refers to a compute node, **S** refers to a storage node, **R** refers to a result node, and X^+ refers to one or more nodes of type X (where $X \in \{\mathbf{I}, \mathbf{C}, \mathbf{S}, \mathbf{R}\}$).

such as the National Institutes of Health (NIH) may be responsible for maintaining the reference database and inserting new reference datasets as they are characterized. Each dataset has metadata computed periodically by the cloud using a set of enrichment algorithms and user criteria. A scientist can then analyze a sample genomic sequence using the correlation between the sample and the cloud-stored reference sequences, performing computations such as those in the standard sequence alignment tool BLAST [4].

We can map this scenario onto the cloud model shown in Figure 1 in the following manner. An *input* node in this example is a genomic sequencing tool that obtains reference datasets. Note that there may be multiple input nodes corresponding to multiple sequencers. These input nodes send the sequence data to a set of *ingest* nodes, which are compute nodes responsible for parsing the sequences and organizing them into files and/or databases. The ingest nodes then send the persistent files and databases to the *storage* nodes. The *enrichment* compute nodes periodically access the stored data to perform additional computations. The enrichment process is typically offline (often batch) processing on the data to update the associated metadata based on user needs. These enriched data sequences and correlations are stored again by the storage nodes. In this application, the *result* node would be a doctor or medical researcher who wishes to correlate a patient's genomic sequence with reference sequences. This data receiver sends a query to the cloud and receives the appropriate data from the storage nodes. This data additionally passes through a compute node acting as a guard, which checks that the data receiver has the proper authorization to see the data requested. The data receiver can also make requests to modify the enrichment process, based on his analytics needs. In this application, the NIH may wish to guarantee the privacy of reference datasets containing sensitive information. Furthermore, the queries executed by doctors or researchers may also be sensitive.

As another example, consider an imagery analysis big data system. Imagery analysis consists of collecting, processing, and analyzing large numbers of images collected from a variety of sensors (e.g., satellite, aerial, smartphone) to extract meaningful information. Unlike the genomic sequencing application, imagery analysis has a heterogeneous set of input nodes, which can include, among other things, distributed sensor networks and image capture devices from aircraft and other platforms. These sensors may be diverse not only in nature but also in geographic location. All of the data gathered by the sensors is ingested into a database and stored on the storage nodes. Enrichment nodes are responsible for user- and mission-specific requests such as correlating imagery data by location, metadata, or description. Many images collected have specific security concerns. The enriched data can be accessed by field operatives and analysts, but only if they have the proper security level or need to know, as checked by the guard compute node. Using such a system, analysts may, for instance, look at imagery of remote areas to determine ground movements.

The taxonomy of cloud nodes types in this model allows us to reason about a wide variety of different cloud applications. For instance, the type and level of protection required may vary depending on which type of node is involved and on the needs of the application. In the next section, we describe a threat model for categorizing the types of threats present in the cloud.

III. Data Security in the Cloud

Cloud computing introduces risks to any sensitive data it touches. These risks largely arise from the need to entrust data protection to a third party cloud provider. Different nodes in the environment may be controlled or administered by different untrusted parties, and could be vulnerable to attacks from other cloud tenants, malicious insiders or external adversaries. When data owners release control of their data to a cloud environment, they require guarantees that their data remains appropriately protected. Today, these guarantees are typically legal promises that the cloud provider makes to the owner as outlined in a service level agreement (SLA). Cryptography allows data owners to protect their data proactively instead of relying solely on legal agreements that are difficult to monitor or enforce.

To understand the protections offered by cryptography in the cloud, we consider security with respect to three traditional security goals:

- *Confidentiality:* All sight sensitive data (computation input, output, and any intermediate state) remains secret from any potentially adversarial or untrusted entities.
- *Integrity:* Any unauthorized modification of modification sensitive data is detectable. Furthermore, the outputs of any computation on sensitive data are correct (i.e., consistent with the input data).
- *Availability:* Data owners (e.g., the output recipients described in the previous section) are ensured access to their data and compute resources.

Since availability is typically addressed in today's cloud environments through non-cryptographic means, we focus only on the confidentiality and integrity of cloud computation and storage. The measures necessary to achieve confidentiality

and integrity depend heavily on how the cloud is deployed, who controls which parts of the cloud, and the trust that exists between these entities. We consider the following three scenarios:

a) Untrusted cloud: One scenario is when the data owners do not trust the cloud or any of the cloud nodes to maintain the confidentiality or integrity of data or computations outsourced to the cloud. Thus, client-side protections are necessary to ensure that confidentiality and integrity are maintained in the face of an adversarial cloud. This scenario will commonly correspond to the public cloud deployment model.

b) Trusted cloud: A second scenario, common in government use-cases, is when the cloud is deployed in an air-gapped environment completely isolated from any outside networks and adversaries. Clients can put their data in the cloud and have assurance that it will remain confidential against outside adversaries. However, even in the isolated environment, some nodes may be corrupted (e.g., due to malware or insiders). These corrupted nodes can't exfiltrate any private data, but they could attempt to violate data and computation integrity. This scenario will commonly correspond to the private cloud deployment model.

c) Semi-trusted cloud: A third scenario that may be particularly relevant to real-world deployments of cloud resources is a *semi-trusted* cloud. In this setting, we neither require the client to fully trust the cloud, nor do we assume that the entire cloud is untrusted. Instead, we assume that some parts of the cloud may be under the control of an adversary at any given time, but that a sufficient fraction of the resources will remain adversary-free. This scenario is consistent with a cloud provider who is trusted to attempt to maintain security, but not to succeed in guarding against every internal or external threat. This scenario may correspond to the hybrid, public, or private cloud deployment models.

To reason about security in the cloud setting, we follow a standard cryptographic approach to modeling the adversary. We model the threat as an adversary that can control cloud nodes of his or her choosing. There are two types of adversaries typically considered in the literature, defined by the capabilities of the parties they corrupt:

- A party corrupted by an *honest-but-curious (HBC)* adversary carries out all computations and protocols exactly as an honest party would. However, the adversary tries to learn additional information by combining the observations of its set of corrupted parties. In particular, an adversary corrupting multiple parties can learn information that no single party could learn individually.
- A party corrupted by a *malicious* adversary may deviate arbitrarily from the prescribed protocols (e.g., by sending malformed messages, actively colluding with other malicious parties, etc.) in an effort to violate the confidentiality or integrity of the data or computations.

The design goal for secure cloud computing is to preserve the confidentiality and integrity of data in the presence of such adversaries. The threat model and the chosen architecture dictate the appropriate solutions for a given situation.

IV. Cryptographic Techniques

In this section, we survey three cryptographic techniques that are particularly applicable to achieving secure big-data analytics in the cloud: *homomorphic encryption* (HE), *verifiable computation* (VC), and *secure multi-party computation* (MPC). We note that many other cryptographic techniques can be used to help secure cloud computing, including functional encryption, identity-based encryption and attribute-based encryption. However, we focus on the techniques we believe are the most promising and relevant for securely delegating computation to a cloud.

The three cryptographic techniques each address one of the scenarios described in Section III: untrusted, trusted, and semi-trusted cloud. For each cryptographic technique, we describe how to use it in the genome sequencing example from Section III, as well as the functionality and security guarantees it provides. Specifically, we characterize each cryptographic technique in terms of the security properties it provides (confidentiality and/or integrity), the adversaries (honest-but-curious or malicious) it protects against, and whether it requires interaction between parties. Figure 2 summarizes the three techniques with respect to these properties. Figure 3 summarizes approximate efficiency cost of each of the techniques across a wide range of computations, depicting the multiplicative performance overhead incurred over unsecured computation.

A. Homomorphic Encryption

Suppose that to lower operating costs, the NIH were to store and process its data on a public cloud, such as the Amazon Elastic Compute Cloud (EC2) [5]. The NIH does not own this cloud or trust the cloud to provide security, but wants to outsource their big-data analytics while maintaining the confidentiality of their data. This corresponds to the untrusted cloud model from Section III. One potential approach is for the NIH to encrypt its data and allow the cloud to perform computation over the encrypted data, without giving the cloud the decryption key. Figure 4a illustrates this approach.

Homomorphic encryption is a type of encryption that allows functions to be computed on encrypted data without decrypting it first. That is, given only the encryption of a message, one can obtain an encryption of a function of that message by computing directly on the encryption. More formally, let $E_k(m)$ be an encryption of a message m under a key k. An encryption scheme is homomorphic with respect to a function f if there is a corresponding function f' such that the $D_k(f'(E_k(m)) = f(m)$, where D_k is the decryption algorithm under key k.

A *fully* homomorphic encryption (FHE) scheme enables arbitrary functions to be computed over encrypted data. FHE is considered the "holy grail" of confidential outsourced computation because it allows *any* computation to be performed over multiple encryptions without decryption. Moreover, a *single*

Cryptographic technique	Adversary type	Confidentiality	Integrity	Requires interaction
Homomorphic Encryption (HE)	Malicious	Y	N	N
Verifiable Computation (VC)	Malicious	N	Y	N
HE + VC	Malicious	Y	Y	Y
Multi-Party Computation (MPC)	Honest-but-curious or malicious	Y	Y	Y

Fig. 2: Comparison of cryptographic approaches showing the type of adversary the approach handles, the security guarantees provided, and whether computation requires interaction between parties. We include the combination of homomorphic encryption (HE) and verifiable computation (VC) separately to highlight the fact that these two techniques can be combined to offer both confidentiality and integrity.

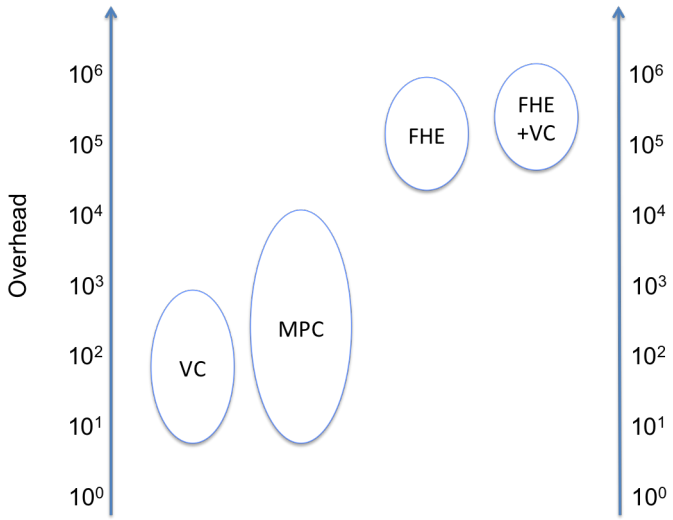


Fig. 3: A graphical depiction of the multiplicative performance overheads over unsecured computation incurred by homomorphic encryption (HE), verifiable computation (VC), and multi-party computation (MPC).

entity in possession of the encryption (such as a cloud node) can perform this computation, without the need for interaction with the data owner or other entities. Alternatively, *somewhat* homomorphic encryption (e.g., El Gamal encryption [6], under which the multiplication of two ciphertexts produces an encryption of the product of the plaintexts) supports restricted classes of functions.

Gentry introduced the first fully homomorphic encryption scheme in 2009 [7]. This was a revolutionary cryptographic achievement, but the scheme was far too inefficient for any practical use. Since 2009, several works (e.g., [8]–[12]) have improved Gentry's technique, significantly reducing the running time. However, fully homomorphic encryption remains prohibitively slow for most use-cases. For example, HElib, a library developed by IBM that provides state-of-the-art implementation of homomorphic encryption, currently performs a matrix-vector multiplication for a 256-entry integer vector in approximately 26 seconds [13], [14].

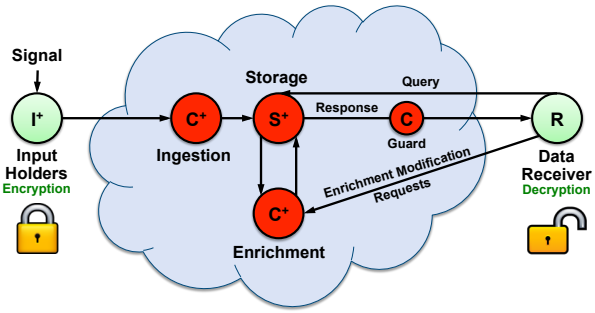
In addition to its inefficiency, homomorphic encryption has other limitations. For instance, homomorphic encryption requires that all sensors and the eventual recipients of the results share a key to encrypt the inputs and decrypt the results, which may be difficult to arrange if they belong to different organizations. Also, homomorphic encryption does not allow for computation on data encrypted using different keys (without incurring additional significant overhead), thus making it impossible for sensors to allow different access to data they contribute to the computation. Some of these limitations can be addressed by other tools such as attribute-based encryption [15] and functional encryption [16]; however, this discussion is beyond the scope of this paper.

Note that homomorphic encryption only guarantees data confidentiality, not integrity. However, it can be combined with verifiable computation (described in Section IV-B) to provide both guarantees. The combination of homomorphic encryption and verifiable computation enables secure computation even on a completely untrusted cloud.

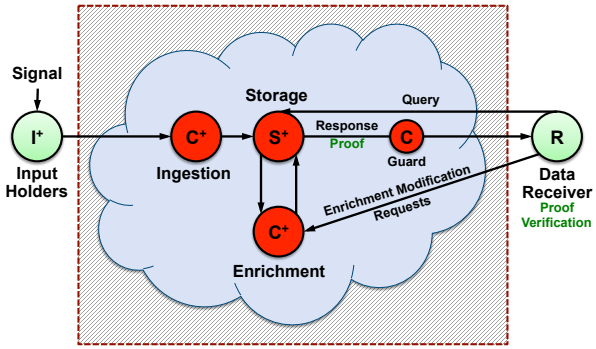
B. Verifiable Computation

Suppose the NIH is willing to expend a significant amount of hardware resources to securely deploy their own private cloud. Specifically, they are willing to set their cloud up in a secure enclave, isolating it completely from the outside world through the use of an air-gap, preventing any data from leaving the cloud. Such a setup would automatically guarantee the confidentiality of data without the need for additional cryptographic protections. However, even in an isolated enclave, we may not want to assume that all parties in the cloud are honest. It may still be possible for an adversary to corrupt machines through malware or supply chain attacks and attempt to compromise the integrity of computations. This corresponds to the trusted cloud model from Section III. An alternate scenario requiring strong integrity protection without confidentiality is cloud computing on public data (e.g., statistical analysis of public census data).

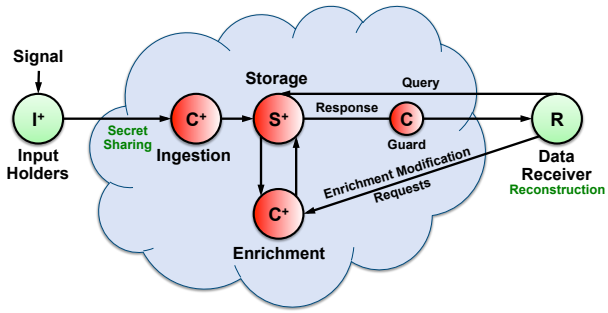
This setting calls for techniques that can guarantee the integrity of the data and the computations without incurring the performance costs associated with achieving confidentiality. The appropriate cryptographic tool for this scenario is *verifiable computation* (VC), which allows the data owner to check the integrity of the computation. In a verifiable computation scheme, the data owner gives his or her data, along with a specification of the computation desired, to some (usually more powerful) entity whom we call the *prover*. The prover then outputs the result of the specified computation, along with some "convincing argument" or "proof" that this output is in fact correct. Such "convincing arguments" typically take the form of one of two cryptographic objects: probabilistically checkable proofs (PCPs) [17], and succinct non-interactive



(a) Homomorphic encryption. Cloud nodes are not trusted to protect confidentiality. Input holders encrypt data before it enters the cloud, and data receivers decrypt the data after it leaves the cloud. The locks denote encryption and decryption.



(b) Verifiable computation. Cloud nodes are not trusted to protect integrity. The compute nodes provide proofs of correct computation, and the data receiver verifies the proof. The dashed line denotes physical isolation from outside networks.



(c) Secure multi-party computation. The cloud is semi-trusted. The input holders secret-share the data among the compute nodes, who perform multi-party computation on the shares. The data receiver reconstructs the output.

Fig. 4: Homomorphic encryption, verifiable computation, and multi-party computation mapped onto our big-data cloud architecture. Shaded red nodes are untrusted or adversarial; partially shaded nodes are semi-trusted.

arguments of knowledge (SNARKs) [18]. Note that it must be easier to verify the proof than to perform the computation; if verification of the computation is as hard as the computation itself, there would be no reason for the data owner to outsource the computation to the prover in the first place. Figure 4b illustrates verifiable computation on the cloud.

Recent implementations include Pinocchio [19] and SNARKs for C [20]. In Pinocchio, verification takes 10 milliseconds, but the prover takes a prohibitively long time to run (144.4 seconds to construct a proof for a computation with 277,745 nested multiplications). In SNARKs for C, verification takes longer (up to 5 seconds), but the prover is more efficient (31 seconds to construct a proof for a computation with 262,144 nested arithmetic operations). These techniques are still much too slow for most big-data analytics applications.

C. Multi-Party Computation

Suppose the NIH decides that, having built their own cloud, they now do not have to view the cloud as entirely adversarial. They are willing to place some trust in its security; however, they do not fully trust that their cloud is free of adversaries, since it is difficult to eliminate the risk of malware or a malicious insider taking control of some of the cloud nodes. This corresponds to the semi-trusted cloud model from Section III.

Secure multi-party computation (MPC) is suited to take advantage of the semi-trusted cloud setting. MPC leverages the presence of honest parties, without necessarily knowing which parties are honest, to achieve confidentiality and integrity of the data and computation. Multi-party computation offers weaker security guarantees than FHE, but can be much more efficient. In MPC, no single party learns anything about the data, but if sufficiently many parties are corrupted by an adversary and pool their information, they can break confidentiality. The relative efficiency of MPC, as well as the applicability of the semi-trusted cloud model to the real world, make it a promising candidate for use in more practical secure cloud computation. Figure 4c illustrates MPC on the cloud.

Secure computation was originally introduced by Yao [21] and Goldreich, Micali and Wigderson [22] and extended to the multi-party case by Chaum, Crépeau, and Damgård [23] and Ben-Or, Goldwasser and Wigderson [24]. Most MPC schemes use a *threshold* adversary model, which limits the total number of nodes that can be corrupted by an adversary at any given time to t out of the n total participating parties. Such schemes share the input data among the participating nodes in such a way that no set of fewer than t shares reveals anything about the input data. By computing on those input shares, the nodes can produce shares of the output that the receiver can reconstruct to obtain the actual output. Because the cloud nodes only see individual shares of the data, they do not learn anything about the data or the computation output. Some seminal multi-party computation schemes based on this secret-sharing paradigm include [23] and [24] which allow $t < \frac{n}{2}$ honest-but-curious or $t < \frac{n}{3}$ malicious corruptions respectively, and [25] which allows $t < n$ malicious corruptions.

The research community has produced a number of practical MPC implementations. Most famously, Danish farmers used MPC in auctions to agree on the price of sugar beets [26]. The VIFF library [27] runs several MPC protocols and takes 2.1 seconds to evaluate a single AES [1] block [28]. Bogdanov et al. developed Sharemind [29], which is secure against $t = 1$ honest-but-curious corruption. Sharemind takes 0.24 seconds to evaluate an AES block. Burkhart et al. developed Sepia [30] to be secure against $t < \frac{n}{2}$ honest-but-curious corruptions. Finally, Ejgenberg et al. are currently developing an efficient general purpose library called Secure Computation Application Programming Interface (SCAPI) [31], that aims to implement many of the efficient MPC protocols.

V. Conclusion and Future Directions

In this paper we presented a computation model for big-data analytics in the cloud and surveyed several cryptographic techniques that can be used to secure these analytics in a variety of settings. While these techniques give a good starting point for secure cloud computing, further research is needed to turn them into practical solutions that can achieve secure cloud computing in the real world.

One particularly interesting future research direction is to design and develop secure multi-party computation techniques tailored specifically for a private semi-trusted cloud setting. This setting allows engineers to design the private cloud together with the cryptographic techniques necessary to protect it. This allows leveraging details of the cloud deployment, such as a fast internal network or trusted hardware, to optimize the cryptographic protocols for use in this particular cloud. Since most current research on MPC is general in nature and does not take advantage of such deployment-specific details, this approach may finally overcome the lack of efficiency of all current MPC protocols and lead to practical solutions for data security in the cloud.

Acknowledgment

The authors would like to thank Sasha Berkoff, as well as the HPEC reviewers, for their helpful feedback on this paper.

References

[1] P. Mell and T. Grace, "The NIST definition of cloud computing," NIST Special Publication 800-145, 2011.
[2] D. Bogdanov, L. Kamm, S. Laur, and P. Pruulmann-Vengerfeldt, "Secure multi-party data analysis: End user validation and practical experiments," 2014.
[3] J. Kepner, D. Ricke, and D. Hutchinson, "Taming biological big data with D4M," *Lincoln Laboratory Journal*, 2013.
[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, 1990.
[5] "Amazon elastic compute cloud," 2014, http://aws.amazon.com/ec2/.
[6] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology*, 1985.
[7] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[1] AES stands for the Advanced Encryption Standard which is a standard block-cipher. The time to compute a one block AES encryption is a commonly used benchmark for testing efficiency of MPC protocols. This computation consists of 10 rounds, each of which consists of several shifts and arithmetic operations on 16 bytes.

[8] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *CRYPTO*, 2012.
[9] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS*, 2012.
[10] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *EUROCRYPT*, 2012.
[11] Z. Brakerski, C. Gentry, and S. Halevi, "Packed ciphertexts in LWE-based homomorphic encryption," in *Public Key Cryptography*, 2013.
[12] C. Gentry, S. Halevi, C. Peikert, and N. P. Smart, "Field switching in BGV-style homomorphic encryption," *Journal of Computer Security*, 2013.
[13] S. Halevi and V. Shoup. (2014) HElib - an implementation of homomorphic encryption. [Online]. Available: https://github.com/shaih/HElib
[14] ——, "Algorithms in HElib," *IACR Cryptology ePrint Archive*, 2014.
[15] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *EUROCRYPT*, 2005.
[16] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *TCC*, 2011.
[17] S. Arora and S. Safra, "Probabilistic checking of proofs: A new characterization of NP," *J. ACM*, 1998.
[18] S. Micali, "CS proofs," *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 1994.
[19] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, 2013.
[20] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive arguments for a von Neumann architecture," Cryptology ePrint Archive, Report 2013/879, 2013.
[21] A. C.-C. Yao, "Protocols for secure computations (extended abstract)," in *FOCS*, 1982.
[22] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, 1987.
[23] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols (abstract)," in *CRYPTO*, 1987.
[24] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *STOC*, 1988.
[25] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *CRYPTO*, 2012.
[26] P. Bogetoft, D. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. Nielsen, J. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft, "Secure multiparty computation goes live," in *Financial Cryptography and Data Security*, 2009.
[27] "Viff, the virtual ideal functionality framework," 2014, viff.dk.
[28] I. Damgård and M. Keller, "Secure multiparty AES," in *Financial Cryptography*, 2010.
[29] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A Framework for Fast Privacy-Preserving Computations," in *Proceedings of the 13th European Symposium on Research in Computer Security - ESORICS'08*, 2008.
[30] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "Sepia: Privacy-preserving aggregation of multi-domain network events and statistics," in *Proceedings of the 19th USENIX Conference on Security*, 2010.
[31] Y. Ejgenberg, M. Farbstein, M. Levy, and Y. Lindell, "SCAPI: The secure computation application programming interface," *IACR Cryptology ePrint Archive*, 2012.