Combining Homomorphic Encryption with Trusted Execution Environment: A Demonstration with Paillier Encryption and SGX

Nir Drucker and Shay Gueron

University of Haifa, Haifa, Israel,
Amazon Web Services Inc., Seattle, WA , USA

Abstract. Cloud database services offer performance and storage advantages that local client platforms do not have, and become very appealing solutions. We list three approaches that address data privacy concerns that are associated with depositing sensitive data on remote platforms. Users can protect their data privacy by locally encrypting it before uploading to remote cloud servers. This prevents the servers from carrying out operations on the data, and also increases the networking overheads. Another approach uses a Trusted Execution Environments (TEE) to protect the data. Examples include OS containers, Virtual Machines or Intel's Software Guard Extension (SGX). This approach relies on the trustworthiness of the TEE for privacy and integrity of operations. The third approach is using Homomorhpic Encryption (HE) schemes. They can allow a remote platform to carry out computations on encrypted data, but are malleable. Adding authentication tags to database entries could solve this problem only if the server is in the user's trust domain. We present here a new combined model. It uses a TEE to guarantee the integrity and correctness of the database code and data, while the data itself is encrypted with some HE scheme. In this way, the malleability protection, achieved through the TEE, is decoupled from the privacy protection that is achieved through the HE. Of course, this comes at some performance costs, but the results of our demonstration, that uses SGX as the TEE and Paillier cryptosystem as the HE, indicate that the proposed combined solution is practical.

Keywords: Secure Guard Extension; Homomorphic Encryption; Trusted Execution Environment; Paillier cryptosystem; Cloud database

1 Introduction

Cloud database services become an appealing solution for handling large amounts of data and computations on behalf of their users. Obviously, a solution to the

Published in http://doi.acm.org/10.1145/3139923.3139933

privacy concerns is required before uploading private data to a remote (untrusted) server. Different types of adversaries are considered: a) attackers from outside the server's Trusted Computing Base (TCB), who can potentially exploit some vulnerabilities in the OS or even in the hypervisor; b) attackers from within the server's TCB, typically having administrator privileges, who can potentially access or modify users' data. Under this model, an insider adversary can craft some manipulations on the computations done on the user's data, and cause incorrect results.

Isolation solutions that are based on secure software can address some threats from attackers from outside the cloud TCB (e.g., OS, hypervisors, BIOS loaders) [1–3]. Other solutions based on secure hardware such as TrustedDB [4] have also been demonstrated.

A different approach was taken in [5], where privacy is achieved by leveraging the power of HE [6], and multiparty computations. In general, there are two types of HE: a) Fully Homomorhpic Encryption (FHE), which allows to perform general-purpose computations over encrypted data. Currently known schemes are too inefficient to be considered practical; b) Partially Homomorhpic Encryption (PHE) which is restricted to supports only one type of operation (e. g., addition or multiplication). These can be practical from the performance viewpoint. An example for a usage where supporting only addition operation suffices, is a rating system where the participants' votes are encrypted with a PHE that supports addition, and uploaded to a database. The ratings can be subsequently determined from the database, without exposing the individual votes to observers who can view it.

Some recent database implementations combine HE and isolation solutions. Monomi [7] allows the untrusted server to handle non-sensitive data, which was encrypted with some HE, while the computations over this data are left for the client. Cipherbase [8] takes the same approach of using HE (when possible) on an untrusted servers, but uses a TEE for computations over sensitive data. These solutions protect data confidentiality, but do not guarantee code and data integrity.

VC3 [9] and M^{2}R [10], offer a distributed MapReduce cloud system solution (a framework for processing problems across large datasets, using a large number of computers). They keep the code and data confidential while providing code and data integrity, by leveraging the capabilities of SGX.

While various solutions handle data privacy, it seems that guaranteeing code and data integrity requires the user to add a third party component to its TCB. For example, solutions that rely on Trusted Platform Module (TPM) devices or secure CPUs (e. g., TrustedDB and Cipherbase) require the users to trust the hardware manufacturer. The VC3 and M^{2}R implementations that use SGX require the users to trust Intel (currently, it is the only attestation service for SGX).

We propose a new combined model that decouples the properties for trust and for privacy. The model uses a TEE (e. g., Intel SGX) to secure the code

* This work was done prior to joining Amazon.

and data integrity, and a PHE scheme (e. g., Paillier cryptosystem) for encrypting the data. In this way, an adversary that tries to exploit the malleability of PHE is blocked by the TEE. At the same time, the PHE guarantees the data confidentiality independently of the trustworthiness of the TEE. We show, by example, a combined solution that has practical performance.

2 Preliminaries and notation

2.1 Paillier cryptosystem

Paillier cryptosystem [11] is a PHE scheme that supports addition. Let $n = pq$ be a modulus with two equal size prime factors p and q, and let $k \in \mathbb{Z}$ and $m_1, m_2 \in \mathbb{Z}_n^*$. Then,

$$D\big(E\left(m_1\right) \cdot E\left(m_2\right) \pmod{n^2}\big) = m_1 + m_2 \pmod{n} \tag{1}$$

$$D\big(E\left(m_1\right)^k \pmod{n^2}\big) = m_1 \cdot k \pmod{n} \tag{2}$$

This property can be expressed as follows: given only the public key and the encryptions $ENC(m_1)$, $ENC(m_2)$ of the messages m_1, m_2, respectively, it is possible to compute $ENC(m_1 + m_2)$. It is also possible to compute $ENC(k \cdot m_1)$, for some constant k. For completeness, we briefly describe the Paillier cryptosystem.

Key generation: denote the Euler totient function by $\phi(n) = (p-1)(q-1)$ and the Charmichael function by $\lambda(n) = lcm(p-1, q-1)$, verify that $gcd(n, \phi(n)) = 1$ else choose a different modulus. Choose an integer $g \in \mathbb{Z}_{n^2}^*$ uniformly at random, such that n divides the order of g in $\mathbb{Z}_{n^2}^*$. Verify this condition with $\mu = \left[L\left(g^{\lambda(n)} \pmod{n^2}\right)\right]^{-1} \pmod{n}$, where $L(\mu) = (\mu - 1)$ `div` n. Set the Paillier public (encryption) key to (n, g), and the private (decryption) key to $(\lambda(n), \mu)$. If $g = n + 1$, then $\lambda(n) = \phi(n)$, and $\mu = \phi(n)^{-1} \pmod{n}$, and the key generation is simplified.

Encryption: to encrypt a message $m \in \mathbb{Z}_n$, select a random value $r \in \mathbb{Z}_n^*$, and compute the ciphertext: $E(m) = c = g^m r^n \pmod{n^2}$.

Decryption: to decrypt the ciphertext $c \in \mathbb{Z}_{n^2}^*$ compute $D(c) = m = L(c^{\lambda(n)} \pmod{n^2}) \cdot \mu \pmod{n}$.

2.2 SGX

Intel's SGX [12–16] is a TEE that protects an implementation from software threats at any privilege level (BIOS, OS, Hypervisor, and user level applications). Moreover, SGX assumes that the system memory is outside its TCB, all the memory reads and writes are encrypted, and integrity and replay protected, using a dedicate hardware unit. We describe SGX briefly.

The basic primitive of SGX is called an "enclave". It is a "container" holding some code, data, and metadata, which realizes some (software) functionality. When shipped, the content of the enclave is in the clear, and can therefore be audited publicly. SGX technology can instantiate an enclave by loading it securely, verifying its cryptographic identity, and locking it in a protected memory region. It also guarantees the isolation of the enclave, during run-time, from any other software on the system (including other enclaves).

A user audits some piece of code (enclave) and determines that it is crafted to execute what he desires. Then, in order to trust an instance of that enclave that runs on a remote platform (and hand it secret information), the user carries out the following protocol. He first communicates with the enclave, still without trusting it. Both parties run a key exchange protocol (e. g., a Diffie-Hellman key exchange), and agree on a shared secret key K_{shared}. Next, the user needs to verify the authenticity of the enclave instance, and that its running environment is a legitimate SGX platform. To this end, he challenges the enclave to prove its trustworthiness, using a "Remote Attestation" protocol that is supported by SGX, and some trusted attestation servers (currently, only Intel has and maintains such a server, but this will change in future versions of SGX). The enclave dispatches the SGX instruction `EREPORT` that generates an authenticated "REPORT". This REPORT is some data structure with information that uniquely identifies the enclave. It also includes a 64 bytes field for arbitrary user data (defined below) that the enclave provides as part of the input to the `EREPORT` instruction. To complete the attestation, the server application uses two dedicated enclaves (currently provided and signed by Intel) that have special capabilities and purposes, as follows.

- The Provisioning Enclave (PE). It generates a private signing key K_{priv} in a special way that enables some external service (currently provided via an Intel server) to sign the matching public key K_{pub}, and return a signed certificate. This procedure is called "platform provisioning". Using K_{priv}, together with the certificate (on K_{pub}), the platform can subsequently prove its cryptographic identity to an external entity. The proof is facilitated by the Quoting Enclave (QE). For privacy reasons, the signing key is generated by the PE, and is not embedded in the processor hardware.
- The Quoting Enclave (QE). It is the only entity that can access and use K_{priv}, to sign a "REPORT" of any other enclave that runs on the same platform.

The enclave fills the reserved bytes with a hash of K_{shared}, generates the authenticated REPORT. It sends it to the QE that verifies it and signs it (only if it is valid). Finally, the signed REPORT is sent by the enclave application to the user, who can establish trust in the (signed) REPORT by contacting Intel Attestation Server (IAS) (that can verify the signature) and validating the hash of K_{shared}. This verification chain proves to the user that the enclave instance that runs on the remote platform is indeed the vetted software, and that it is running under the SGX supervision.

3 Combined Trusted Model

HE schemes are considered malleable, in the following sense: by applying the homomorphic operation, an adversary can modify a ciphertext in a way that it would still decrypt into a valid plaintext. To illustrate the problem, we provide a simplified (fictional) example. A company `Comp` uses cloud service `CloudDB` to store a database that consists a table $Salaries(ID, department, E_{K_{PHE}}(salary))$. The table stores employee ID's, and their respective department, as plaintext, while their respective (confidential) salary is encrypted with the Paillier scheme. `CloudDB` can carry out computations on the database, using the public modulus n on behalf of `Comp` . An employee at `CloudDB` (with the right privileges) can modify any row in that table, say $R = (id, dep, c)$. An arbitrary change in c will most likely decrypt into an illegitimate plaintext. However, by squaring $c' = c^2 \pmod{n^2}$ and replacing R by $R' = (id, dep, c')$, it is possible to double the salary of employee id.

This type of threat is not mitigated by adding authentication tags to the database on the remote platform. Only the user who owns the authentication key can generate new authentication tags. Thus, the untrusted server will not be able to execute modification queries (e. g., UPDATE, INSERT, DELETE) and complex math queries (such as SUM or AVEREGE) while keeping the data authenticated. In order to solve this problem, the user should hand the authentication key to the server, and for that, he must trust it. It follows that every solution that involve HE on cloud servers, should include also a TEE or a Trusted Proxy (TP) as defined in [17].

We now address a different threat. The CEO of `Comp` wishes to check the total costs of the different departments, and queries the server for the total salaries of the members in each department. The summation is computed on the cloud, using the PHE properties. An attacker on the remote server environment can change the encrypted results $E_{K_{PHE}}(dep\_sum)$ that the CEO would see. For example, by first reading a random row $R = (id, dep, c)$, where $c = E_{K_{PHE}}(s)$ is the encryption of a valid salary s, and then, manipulate the code (or the network traffic) to modify the summation result into $E_{K_{PHE}}(dep\_sum + k \cdot s) = c^k \cdot E_{K_{PHE}}(dep\_sum) \pmod{n^2}$ which is higher than the real total value. Note that the CEO cannot validate the correctness of the result, which would be valid and also pass integrity checks.

The malleability problem of HE can be mitigated by using a TEE. However, solutions that use a TEE currently rely totally on that TEE, for both integrity and confidentiality. Here, we suggest a new combined model that leverages the capabilities of both TEE and PHE. In this model, the TEE guarantees the code and data integrity, and privacy is protected by PHE. This approach decouples the integrity from the confidentiality. Although this seems like a very slow solution, we show later that the resulting performance is still reasonable. We chose here a specific combination of a TEE and PHE, namely SGX and Paillier encryption.

Our goal was to choose a TEE with the smallest possible user's TCB, and to this end, SGX is a suitable candidate. It includes only Intel in the user's TCB, but excludes the OS, hypervisor, BIOS, and physical devices. Our combined model

considers four entities: 1) A user, who is the owner of the confidential data; 2) An application (an enclave in our case) that the user can audit and vet. Then, the user can trust the enclave if it is securely loaded on the remote platform, and the user receives an attestation to that fact (see Section 2.2); 3) An untrusted application (`uApp`) whose role is only to launch the enclave, and connect it to the server's OS; 4) A database that holds confidential information;

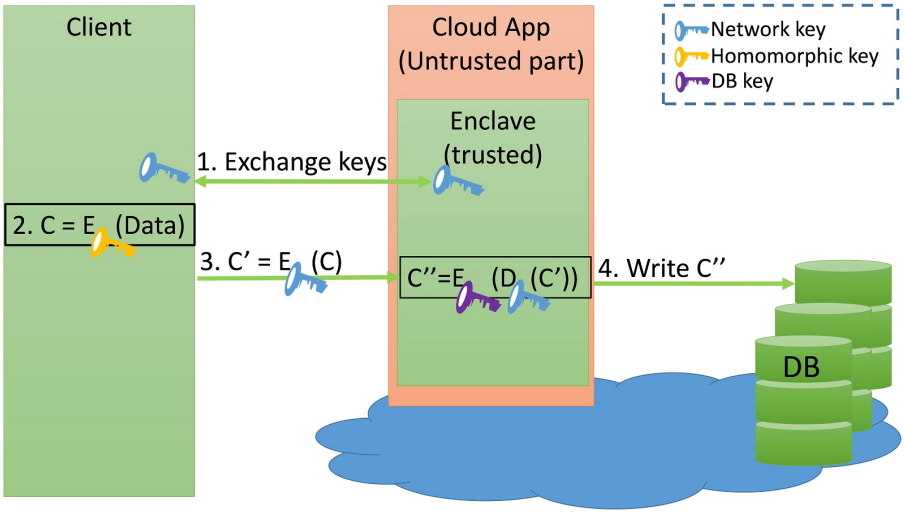

Fig. 1. Adding data to a cloud DB. The user establishes a secure channel with the enclave, where both parties agree on the common network key (blue). Subsequently, the user encrypts his data with some PHE (yellow key), and re-encrypts it with the network key. The enclave decrypts the received messages, using the network encryption key, re-encrypts it (purple key) and stores the data in the DB.

Figure 1 illustrates the flow of uploading data to a cloud database. The flow use the untrusted application `uApp`, only to launch the (already vetted) enclave that exchanges a network key ($K_{network}$). Note that the user does not need to trust `uApp`to handle his data, or even to correctly pass it to the trusted enclave (a demonstration of a Transport Layer Security (TLS) protocol that runs from an enclave is shown in [18]). Finally, the user follows the Remote Attestation protocol (Section 2.2) to determine the trustworthiness of the specific enclave instantiation.

Uploading data: The user encrypts the data with a PHE scheme, obtaining the ciphertext $c = E_{K_{PHE}}(data)$. He encrypts it to $c' = E_{K_{network}}(c)$, and sends over the network. Only the enclave (but not `uApp`) can decrypt c' into c. At this point, the enclave needs to store the data (which is encrypted with PHE) in the database. It re-encrypts it (or at least adds an authentication tag), in order to address the malleability of the PHE ciphertext.

Summation queries: The user establishes a secure channel with the enclave and submits a query request. The enclave reads and authenticates the relevant data from the DB. Then it performs the required calculations by means of homomorphic operations. The result is re-encrypted with the network key, and sent back to the user who can decrypt the data with both keys (the network key and the homomorphic private key), to obtain the result.

4 Demonstration and results

The combined model uses both a TEE and a PHE, and this affects the performance. To estimate the cost, we implemented three different models that use SGX as the TEE and/or Paillier cryptosystem as the PHE, as follows.

- A PHE model. The user encrypts the data with Paillier encryption, prior to uploading it to the server. The server does not use a TEE.
- An SGX model. The server uses SGX for isolation and for code integrity. The user uploads his data to an SGX enclave, using a secure/confidential channel.
- The combined model. The user encrypts the data with Paillier encryption prior to uploading it to the server. The server uses SGX to ensure isolation and code integrity.

The experiment. We simulated two different users. They operate similarly, except that one uses homomorphic encryption, and the other does not. We also developed a simulation of three types of servers, as described above. To simplify our demonstration, we skipped the TLS implementation inside the enclave (as done in [18]), which would provide the enclave with a network key and after attestation, a database key. We simply embedded these two keys in the user and the enclave simulators. The database was a file that holds a table with the information (no compression or optimization was used). A CMAC tag was appended to each row of the table, to protect it from unauthorized modification. Furthermore, to protect the whole database from an attacker who deletes or injects (e. g., duplicates) rows, we added an index table and stored it in the enclave. Since SGX has limits on the overall size of an enclave, we used an appropriately sized index table (a larger index table could be stored outside the enclave, but with additional complication). We used a database with 1000 IDs. By the SGX architectural definitions, an enclave cannot read an external file, and it must use an external API of an assisting application (`uApp`). When a desired row is fetched, the enclave verifies its ID and the validity of the authentication tag.

We designed our demonstration to support summation queries of the from "SELECT SUM(salary) FROM Salaries WHERE id¿=*low* AND id¡=*high*". Paillier encryption allows such queries to be carried out on the ciphertext. We also leveraged the method of [19], where the ciphertext that is stored on the server is already converted to a Montgomery friendly format.

In our experiments, we queried the database with different ID ranges, starting from $low = 0$ up to $high \in (0, 1000]$, incremented in steps of 25. We used

an Intel®desktop of the 7^{th} Intel®Core^{TM} Generation (Microarchitecture Codename "Kaby Lake"), where the Intel®Turbo Boost Technology was turned off (i. e., , the frequency was fixed). The Intel®Hyper-Threading Technology, and the Enhanced Intel Speedstep®Technology were disabled. The OS was Ubuntu 64 bits. Each measurement was run 100 times (warm-up), followed by 50 iterations that were clocked and averaged. To minimize the effect of background tasks running on the system, we repeated each measurement and record the average result.

Figure 2 shows the querying performance, measured in millions of processor cycles. As seen, it grows linearly with the number of summed entries. The SGX model and the PHE model have roughly the same performance. The combined model is (only) $1.7x$ slower.

5 Conclusion

This paper presented a combined model for handling information on remote servers. It combines the capabilities of a TEE for code and data integrity, and the capabilities of PHE for data privacy. Previous solutions choose one of these methods, or use both as orthogonal components (e. g., using PHE only on non-sensitive data). Our approach decouples the privacy and integrity functionalities, to allow addressing the malleability of PHE with TEE features and protecting confidentiality by the encryption, while enjoying the benefits that homomorphic schemes can offer.

We designed an experiment that is based on SGX as the instantiation of the TEE, and Paillier cryptosystem as the instantiation of the PHE. Our results compare the runtime of the combined model to: a) only SGX, where integrity and privacy are bundled under the same TCB; b) only PHE, where data privacy is protected, but malleability can be leveraged. We show that our model is only 1.7x slower than #a and #b, and can be viewed as a practical solution.

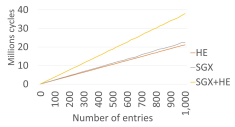

Fig. 2. Comparison of a the performance of summation queries in the different cloud database approaches: a) Using only PHE; b) Using only SGX; c) Using both SGX and PHE. The horizontal axis shows the number of summed entries. The vertical axis shows the number of millions of cycles, required to perform the query (lower is better). See text.

Acknowledgments

This research was supported by the ISRAEL SCIENCE FOUNDATION (grant No. 1018/16), by the BIU Center for Research in Applied Cryptography and Cyber Se-

curity in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, by the Blavatnik Interdisciplinary Cyber Research Center (ICRC) at the Tel Aviv University, and by the PQCRYPTO project, which was partially funded by the European Commission Horizon 2020 research Programme, grant #645622,

References

1. Chen, X., Garfinkel, T., Lewis, E.C., Subrahmanyam, P., Waldspurger, C.A., Boneh, D., Dwoskin, J., Ports, D.R.: Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. SIGARCH Comput. Archit. News **36**(1) (March 2008) 2–13
2. McCune, J.M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., Perrig, A.: Trustvisor: Efficient tcb reduction and attestation. In: 2010 IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA, USA, IEEE, IEEE (May 2010) 143–158
3. Li, Y., McCune, J., Newsome, J., Perrig, A., Baker, B., Drewry, W.: Minibox: A two-way sandbox for x86 native code. In: USENIX ATC 14, Philadelphia, PA, USENIX (2014) 409–420
4. Bajaj, S., Sion, R.: Trusteddb: A trusted hardware-based database with privacy and data confidentiality. IEEE Transactions on Knowledge and Data Engineering **26**(3) (March 2014) 752–765
5. Fournet, C., Kohlweiss, M., Danezis, G., Luo, Z.: ZQL: A compiler for privacy-preserving data processing. In: Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13), Washington, D.C., USENIX (2013) 163–178
6. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing. STOC '09, New York, NY, USA, ACM (2009) 169–178
7. Tu, S., Kaashoek, M.F., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. In: Proceedings of the 39th international conference on Very Large Data Bases. PVLDB'13, Trento, Italy, VLDB Endowment (2013) 289–300
8. Arasu, A., Blanas, S., Eguro, K., Kaushik, R., Kossmann, D., Ramamurthy, R., Venkatesan, R.: Orthogonal security with cipherbase. In: CIDR, –, CIDR (2013) 1–10
9. Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., Russinovich, M.: Vc3: Trustworthy data analytics in the cloud using sgx. In: 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, IEEE, IEEE (May 2015) 38–54
10. Dinh, T.T.A., Saxena, P., Chang, E.C., Ooi, B.C., Zhang, C.: M2r: Enabling stronger privacy in mapreduce computation. In: 24th USENIX Security Symposium (USENIX Security 15), Washington, D.C., USENIX (2015) 447–462
11. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In Stern, J., ed.: Advances in Cryptology — EUROCRYPT '99, Berlin, Heidelberg, Springer (may 1999) 223–238
12. –: Intel ® Software Guard Extensions Programming Reference (October 2014) https://software.intel.com/en-us/isa-extensions/intel-sgx.
13. Anati, I., Gueron, S., Johnson, S., Scarlata, V.: Innovative technology for cpu based attestation and sealing. https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing (2013)
14. Hoekstra, M., Lal, R., Pappachan, P., Phegade, V., Del Cuvillo, J.: Using innovative instructions to create trustworthy software solutions. In: Proceedings of the

2Nd International Workshop on Hardware and Architectural Support for Security and Privacy. HASP '13, NY, NY, USA, ACM (2013) 1–11
15. Johnson, S., Scarlata, V., Rozas, C., Brickell, E., Mckeen, F.: Intel ®Software Guard Extensions: EPID provisioning and attestation services. White Paper **1** (April 2016) 1–10
16. McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C.V., Shafi, H., Shanbhogue, V., Savagaonkar, U.R.: Innovative instructions and software model for isolated execution. In: Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy. HASP '13, New York, NY, USA, ACM (2013) 10:1–10:1
17. Drucker, N., Gueron, S., Pinkas, B.: Faster secure cloud computations with a trusted proxy. IEEE Security & Privacy **-** (in press) 1–14
18. Aublin, P.L., Kelbert, F., O'Keeffe, D., Muthukumaran, D., Priebe, C., Lind, J., Krahn, R., Fetzer, C., Eyers, D., Pietzuch, P.: Talos: Secure and transparent tls termination inside sgx enclaves (2017)
19. Drucker, N., Gueron, S.: Paillier-encrypted databases with fast aggregated queries. In: 2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC), Las Vegas, NV, USA, IEEE, IEEE (Jan 2017) 848–853