Achieving trustworthy Homomorphic Encryption by combining it with a Trusted Execution Environment

Nir Drucker[1,2] and Shay Gueron[1,2]*
[1]*University of Haifa, Haifa, Israel*
[2]*Amazon Web Services, Seattle, WA, USA*
drucker.nir@gmail.com, shay@math.haifa.ac.il

Abstract

Cloud database services become very appealing solutions. They offer performance and storage capabilities that client platforms do not have. However, in order to protect the users' confidentiality and to ensure the integrity of their computations, solutions often use one of three approaches: a) Encrypting the data prior to uploading it with some symmetric encryption; b) Using a Trusted Execution Environments (TEE) such as OS containers, Virtual Machines or Intel's Software Guard Extension (SGX); c) using Homomorphic Encryption (HE) schemes. A newer approach, which we call the "combined model" 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 paper, we explain the combined model and we show how to use it in the context of modern Multi Party Computations (MPC) schemes. In addition, we demonstrate how to construct a voting system that leverages its capabilities.

Keywords: Secure Guard Extension, Homomorphic Encryption, Trusted Execution Environment, Paillier cryptosystem, Cloud database, Multi Party Computations

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 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 to be returned.

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). Examples include [1, 2, 3, 4, 5, 6, 7, 8]. Solutions based on secure hardware are [9, 10, 11], and [12] (TrustedDB), have also been demonstrated. Other systems [13, 4] allow users to verify computation results, but do not protect code and data confidentiality.

A different approach was taken in [14], where privacy is achieved by leveraging the power of HE [15], and multiparty computations. In general, there are two types of HE: a) Fully Homomorphic Encryption (FHE), which allows to perform general-purpose computations over encrypted data. Currently known schemes are too inefficient to be considered practical; b) Partially Homomorphic 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.

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 9:1 (March 2018), pp. 86-99
*Corresponding author : University of Haifa, Haifa, Israel and Amazon Web Services, Seattle, Washington, USA, Tel: +972-4-8324858 (This work was done prior to joining Amazon.)

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. CryptDB [16], is an example for an SQL database that uses PHE (Paillier cryptosystem) for performing queries of the type "SELECT SUM ..." and "UPDATE X=X+1 ...". One of the case studies in [16] is HotCRP, a popular conference review application. MrCrypt [17] is a client side static analysis tool that analyzes user data, before a database exists and identifies what type of PHE scheme should be use and where. Both, CryptDB and MrCrypt protect data confidentiality, but do not guarantee code and data integrity.

Some recent database implementations combine HE and isolation solutions. Monomi [18] 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 [19] 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 [20], M^2R [21] and [22], offer a distributed MapReduce cloud system solution (a framework for processing problems across large datasets, using a large number of computers). It keeps 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^2R implementations that use SGX require the users to trust Intel (currently, it is the only attestation service for SGX).

In [23], we introduced a new combined model that decouples the properties for trust and for privacy. This model uses a TEE (e. g., Intel SGX) to secure the code 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.

To show that this model has practical performance, we constructed a demonstration that uses SGX as the TEE and Paillier cryptosystem as the HE. The results indicate that the performance costs of such combined solution are sufficiently reasonable to make it practical. In addition, this paper extends the previous examples by explaining how this model can be used in the context of modern MPC schemes, furthermore, we demonstrate how to construct a voting system that leverages the capabilities of the combined model.

The paper is organized as follows. Section 2 briefly describes Paillier PHE and Intel's SGX. We describe our combined model in Section 3, and present the implementation and some performance measurements, in Section 4. Section 5 describe some modern usages of our model and Section 7 concludes the paper.

2 Preliminaries and notation

2.1 Paillier cryptosystem

Paillier cryptosystem [24] 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,

$$DEC\big(ENC(m_1) \cdot ENC(m_2) \pmod{n^2}\big) = m_1 + m_2 \pmod{n} \tag{1}$$

$$DEC\big(ENC(m_1)^k \pmod{n^2}\big) = m_1 \cdot k \pmod{n} \tag{2}$$

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 $L(\mu) = (\mu - 1)\ \mathtt{div}\ n$. 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

$$ENC(m) = c = g^m \cdot r^n \pmod{n^2}.$$

Decryption: to decrypt the ciphertext $c \in \mathbb{Z}^*_{n^2}$ computed

$$DEC(c) = m = L(c^{\lambda(n)} \pmod{n^2}) \cdot \mu \pmod{n}.$$

A typical implementation of Paillier encryption, which is considered secure, uses 1024-bit primes, and therefore a 4096-bit modulus (n^2).

2.2 SGX

Intel's SGX [25, 26, 27, 28, 29] 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), at any privilege level.

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_{sign}^{priv} in a special way that enables some external service (currently provided via an Intel server) to sign the matching public key K_{sign}^{pub}, and return a signed certificate. This procedure is called "platform provisioning". Using K_{sign}^{priv}, together with the certificate (on K_{sign}^{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_{sign}^{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 [30].

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. Note that Intel has recently announced [31] that SGX will be available for server platforms and not only for the client platforms as it is currently. 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.

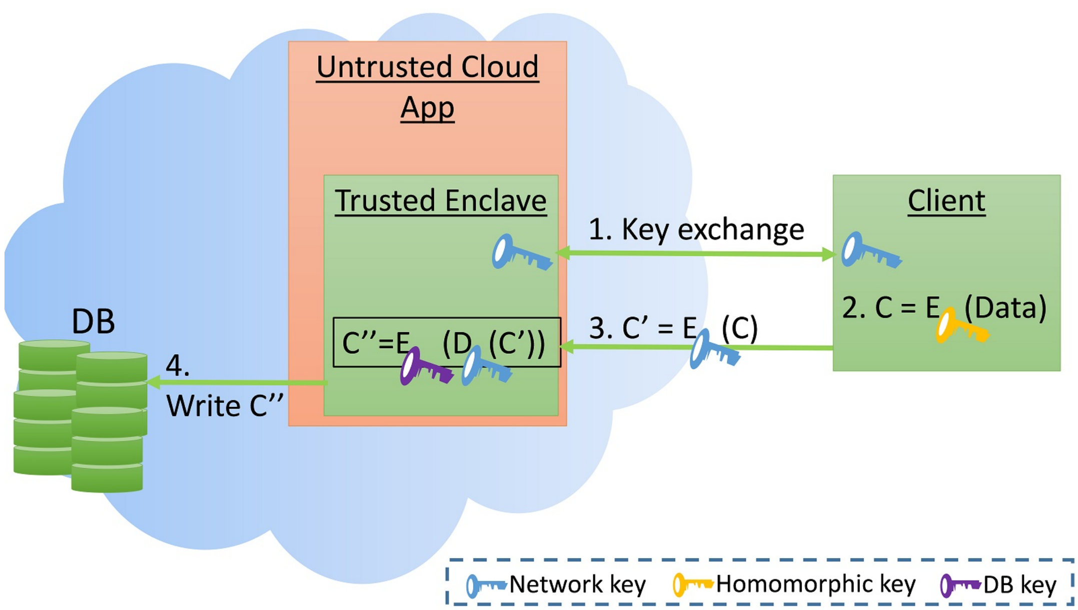

Figure 1: Adding data to a cloud DB. The user establishes a secure channel with the enclave, where both parties agree on the common network key (blue). Subsequently, the user encrypts his data with some PHE (yellow key), and re-encrypts it with the network key. The enclave decrypts the received messages,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

Figures 1 and 2 illustrate the flows of uploading data and querying it from a cloud database. The flows 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 [32]). 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 to `uApp`) can decrypt c' into c. At this point, the enclave needs to store the data (which 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 procedure to request a summation query to the DB is explained in Figure 2.

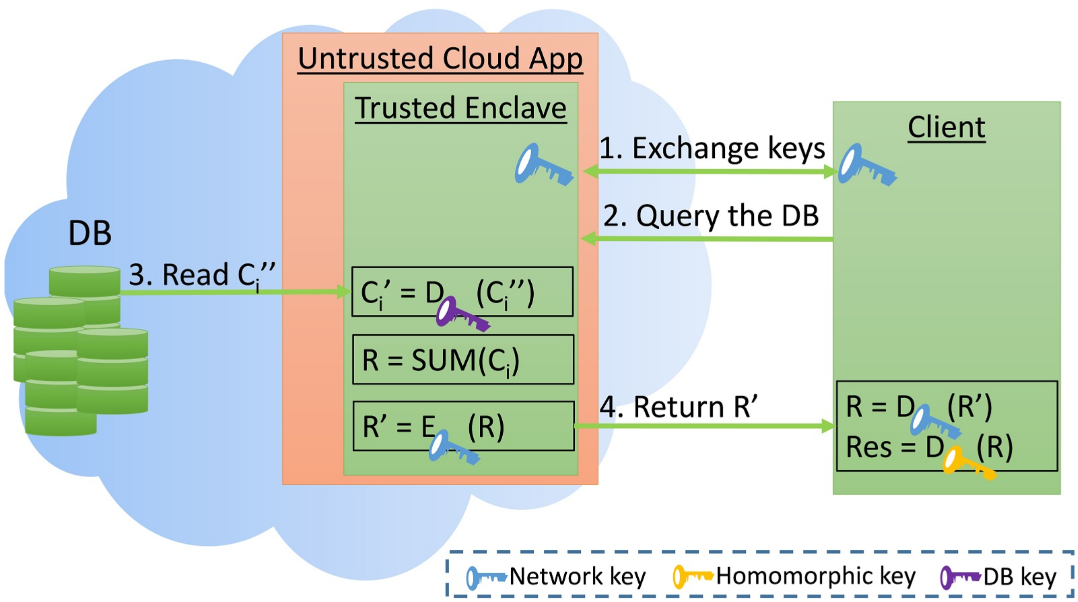

Figure 2: A summation query to the DB. The user and the enclave establish a secure channel (blue key). The user submits a query request to the enclave. The enclave reads from the DB, and authenticates this data. It then performs the required calculations by means of homomorphic operations (on the PHE-encrypted entries). The result is re-encrypted the results with the network key, and sent back. The user can decrypt with 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 [32]), 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. Data authentication with the database key uses the `sgx_rijndael128_cmac_msg` API of the SGX SDK [33].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 BETWEEN *low* AND *high*

Paillier encryption allows such queries to be carried out on the ciphertext (this property can be easily extended to support queries that request an average or a standard deviation). We also leveraged the method of [34], where the ciphertext that is stored on the server is already converted to a Montgomery friendly format. To query the database, the user provides a range $[low, high]$ of IDs, and the server accumulates the entries in this range (the code can skip IDs in the specified range, which do not exist in the database).

We note that Paillier ciphertext has twice the size of the corresponding plaintext. This is different from symmetric encryption where the ciphertext and the plaintext have the same size. To properly scale the comparison, we set the same plaintext size in all three models. Specifically, each row of the tale was represented by a vector of 64-bit integers, stored in a 2048-bit container.

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 (Micro-architecture 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 operating system was Ubuntu 64 bits. Each measurement was run 100 times (warm-up), followed by 50 iterations that were clocked (using the RDTSC instruction) and averaged. To minimize the effect of background tasks running on the system, we repeated each measurement and record the average result.

Figure 4 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 Multi Party Homomorphic Encryption (MPHE)

This section presents additional use cases that can benefit from our combined approach. These are cryptosystems that use FHE in a secure MPC setting. An MPC scheme involves n parties p_i, $i = 1, \ldots, n$,

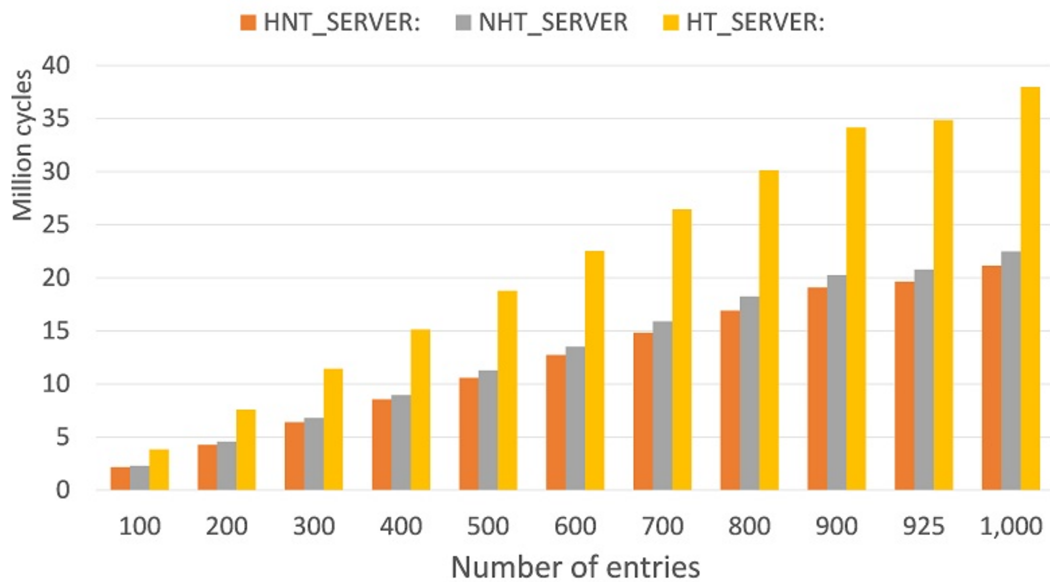

Figure 3: Comparison of a the performance of summation queries in the different cloud database approaches: a) Using only PHE; b) Using only SGX; c) Using both SGX and PHE. The horizontal axis shows the number of summed entries. The vertical axis shows the number of millions of cycles, required to perform the query (lower is better). See explanation in the text.

each one holds a private input x_i, and they wish to compute a given function $y = f(x_1,\ldots,x_n)$. An MPC protocol is considered secure if it ends with exposing y, nothing more than y, and only to the honest parties. In addition, no subset of the parties can collude to obtain the value of y, without agreement of the majority of the participants. The problem was initially studied by Yao [35, 36] for the case of two honest-but-curious parties, who follow the protocol but try to get some extra information from the execution flow. Later, this protocol was extended [37] to an arbitrary number of malicious parties, who do not necessarily follow the protocol. This family of problems has received a significant amount of study.

The efficiency of an MPC protocol can be improved by combining it with some FHE scheme [15]. For example, consider the case where the protocol involves only two honest-but-curious parties p_1, p_2. Here, p_1 can encrypt x_1 with his secret key, and send the ciphertext to p_2. Subsequently, p_2 can use FHE to evaluate the function using his own data and the ciphertext received from p_1. The final results are sent back to p_1. This protocol reduces the communication complexity, and also benefits from the asymmetric computation (the evaluation of y is done only by p_2). A more significant improvement is achieved when this technique is applied to a protocol with an arbitrary number of parties, where a cloud service is used for performing the (heavy) homomorphic calculations. In both cases, the computational complexity for each party depends only on the encryption/decryption complexity, and is independent of the complexity of the function's evaluation (which is carried out on the server on the cloud provider's side).

The examples below are based on the following two MPC variants called Threshold Fully Homomorphic Encryption (TFHE) [38] and Multi-Key Fully Homomorphic Encryption (MFHE) [39]. They deal with an arbitrary number of malicious participants. The TFHE protocol is as follow: a) The parties agree on a common public key and a secret key. Each party gets only a (secret) share of the secret key. Here, all the shares are required in order to recover the secret key; b) Each party p_i uploads the encryption of x_i, using the public key. Subsequently, each party can evaluate y (using FHE); c) The parties collaborate to decrypt the result. The MFHE protocol improves the TFHE protocol by reducing the number of steps. This is done by replacing steps a, b with the following step: party p_i chooses (individually) public and secret keys, encrypts x_i with its public key, and uploads the results. An example for MFHE, based on Learning With Errors (LWE), is shown in [40].

Although, FHE schemes can help in reducing the bandwidth required by an MPC protocol, they are still not very practical due to the associated computational overheads. Reduced HE schemes such as PHE support only one operation, and may therefore be less relevant in some usages. Fortunately, recent research suggests a third family of HE schemes, called Somewhat Homomorphic Encryption (SHE). These cryptosystems are the same as FHE schemes, but are limited by the number of steps they can perform (it is possible to overcome this limitation by adopting some bootstrapping techniques). An MPC protocol that uses SHE is reported in [41].

Examples 1, and 2 propose two scenarios where MPC protocol (equipped with HE) can be used. We argue that both examples can benefit, security-wise, from the combined model presented above.

Example 1 (Modern voting systems)**.** *Electronic voting systems must protect the anonymity of the voters and their votes, assure that each voter votes at most once, and protect the integrity of the final results. A cloud based voting system can be implemented efficiently by using an MFHE protocol. Here, each voter chooses a public and a secret key, encrypts his vote and uploads it to the cloud. All votes are stored in a trusted (and auditable) database. This trustworthiness protects against elimination and duplication of votes, as described in Section 3. The HE scheme, which is used for evaluating the accumulated results, can also be used for enforcing legitimate inputs.*

Note that, the protocol that is outlined in Example 1, can be similarly used for other purposes such as public auctions.

Example 2 (An online purchasing scenario)**.** *The CEO of `Comp` rewards his employees with a coupon for some online purchases. Each employee is limited to a total of T_i dollars, and the total budget allocated for this activity is T ($\sum T_i \leq T$). The CEO wishes to monitor this activity without compromising the privacy of his employees. The MFHE protocol described above can be used as an instantiation.*

6 A voting system protocol based on combined model

Example 1 outlines a voting system protocol, which is based on MFHE. An inherent drawbakc of this protocol is that the results must be decrypted by all the participants (or at least the majority of them). Here, we suggest a different protocol that is based on our combined model, and involves only PHE (e. g., Paillier). Our model has four entities: a) a voting committee ($\mathscr{C}$); b) voting centers (v_i, $0 \leq i \leq |v|$ where $|v|$ denotes the number of voting centers); c) voters (v_{ij}, $0 \leq j \leq n_i$, where n_i is the number of voters in the voting center v_i); d) a cloud server ($\mathscr{S}$). It includes three protocols, **Init**, **CollectVotes** and **Validate**.

Init The committee chooses randomly a public (pk_i) private (sk_i) key pair for each voting center v_i. It initializes a special device d_i with pk_i, and hands it to v_i. In addition, it communicates with an SGX enclave e^{init}, found on $\mathscr{S}$. This enclave receives pk_i from $\mathscr{C}$, and seals the value $acc_i = E_{pk_i}(0)$ on $\mathscr{S}$'s hard drive.

CollectVotes Each voter v_{ij} uses the device d_i to vote. Device d_i triggers a different SGX enclave (e^{vote}), sends it the Paillier encrypted vote ($c_i = E_{pk_i}(v_{ij})$) and waits for a final confirmation to increment its internal counter n_i. e^{vote} unseals acc_i, validates its authenticity, and multiplies it with c_i. The result, acc_i, is then resealed. Finally, the server send a confirmation to d_i.

Validate After all the votes are collected, $\mathscr{S}$ sends $\mathscr{C}$ the accumulated results acc_i, for each v_i from $\mathscr{S}$. The devices d_i provide $\mathscr{C}$ with the number of voters n_i of d_i. Subsequently, $\mathscr{C}$ decrypts the results, and compares the number of votes to n_i. If the numbers match, it approves the votes of v_i.

Remark 1. *To enable votes accumulation, d_i creates a plaintext m that is formatted as a row, where each cell represents one of the choices. Thus, when voting for x candidates, we have $|m| \geq x \times \max(n_i)$ ($|m|$ is the length of the message).*

Remark 2. *The **Validate** functionality can be further improved by handing sk_i and n_i to a third enclave (e^{res}) that runs on $\mathscr{S}$. This enclave can validate the result of each voting center, accumulate them, and post them online. We note that providing the secret key to $\mathscr{S}$ does not violate the privacy of the voters, because at this stage, the voting is already finished, and $\mathscr{S}$ that uses a vetted enclave, is trusted to not store any intermediate results.*

7 Conclusion

This paper presents a combined model for handling information on remote servers. The model leverages the capabilities of a TEE for code and data integrity, and the capabilities of PHE for data privacy. Other existing solutions choose one of these methods, or use both of them as orthogonal components: e. g., using PHE only on non-sensitive data. Our approach decouples the privacy and the integrity considerations. It allows for solving the problem of malleability of PHE by using the features of TEE. Data confidentiality is protected by (homomorphic) encryption, while the system enjoys the advantages of the homomorphic properties.

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 an attack vector. Our numbers show that the combined model is only 1.7x slower than #a and #b, and can be therefore considered a practical solution.

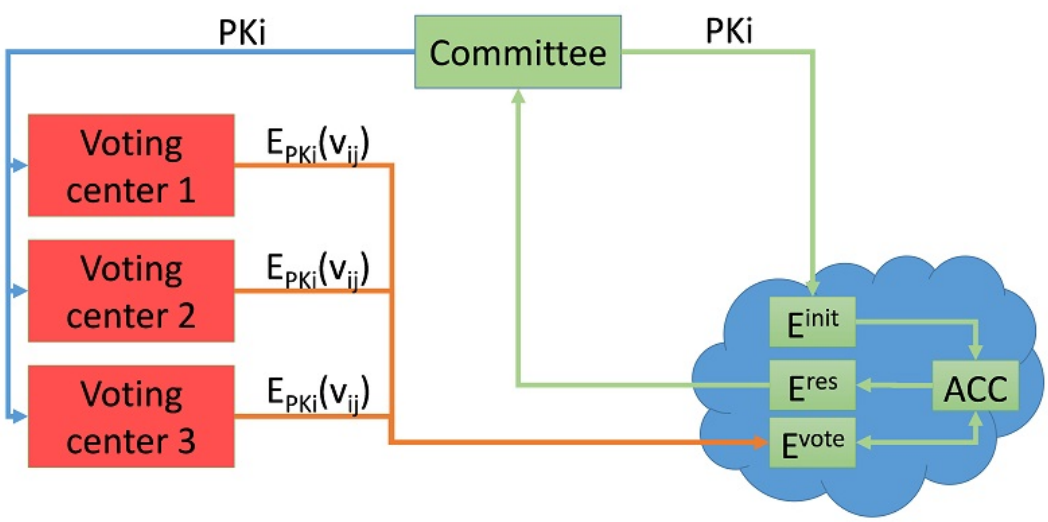

Figure 4: Voting system based on our combined model.

Finally, we reported on several other examples such as MPC and voting systems, where HE is used, and therefore our combined model can enhance their overall security. Our future research directions include performance comparison of different implementations of the combined model, by using different types of TEE or PHE solutions. We plan to explore various software/hardware optimizations that can accelerate cryptographic protocols, such as, for example, auctions.

Acknowledgments

This research was supported by the Israel Science Foundation (grant No. 1018/16), the BIU Center for Research in Applied Cryptography and Cyber Security, in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and the Center for Cyber Law and Policy at the University of Haifa.

References

[1] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports, "Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems," *SIGARCH Computer Architecture News*, vol. 36, no. 1, pp. 2–13, March 2008.

[2] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "Trustvisor: Efficient tcb reduction and attestation," in *Proc. of the 2010 IEEE Symposium on Security and Privacy (SP'10), Berkeley/Oakland, California, USA*. IEEE, May 2010, pp. 143–158.

[3] F. Zhang, J. Chen, H. Chen, and B. Zang, "Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization," in *Proc. of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11), Cascais, Portugal*. ACM, October 2011, pp. 203–216.

[4] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu, "Policy-sealed data: A new abstraction for building trusted cloud services," in *Proc of the 21st USENIX Security Symposium (USENIX Security'12), Belleve, Washington, USA*. USENIX, August 2012, pp. 175–188.

[5] R. Strackx and F. Piessens, "Fides: Selectively hardening software application components against kernel-level or process-level malware," in *Proc. of the 2012 ACM Conference on Computer and Communications Security (CCS'12), Raleigh, North Carolina, USA*. ACM, October 2012, pp. 2–13.

[6] O. S. Hofmann, S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel, "Inktag: Secure applications on an untrusted operating system," in *Proc. of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'13), Houston, Texas, USA*. ACM, March 2013, pp. 265–278.

[7] J. Criswell, N. Dautenhahn, and V. Adve, "Virtual ghost: Protecting applications from hostile operating systems," *SIGPLAN Notices*, vol. 49, no. 4, pp. 81–96, Februray 2014.

[8] Y. Li, J. McCune, J. Newsome, A. Perrig, B. Baker, and W. Drewry, "Minibox: A two-way sandbox for x86 native code," in *Proc. of the 2014 USENIX Annual Technical Conference (USENIX ATC'14), Philadelphia, Pennsylvania, USA*. USENIX Association, June 2014, pp. 409–420.

[9] E. Owusu, J. Guajardo, J. McCune, J. Newsome, A. Perrig, and A. Vasudevan, "Oasis: On achieving a sanctuary for integrity and secrecy on untrusted platforms," in *Proc. of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS'13), Berlin, Germany*. ACM, November 2013, pp. 13–24.

[10] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," *SIGPLAN Notices*, vol. 35, no. 11, pp. 168–177, November 2000.

[11] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "Aegis: Architecture for tamper-evident and tamper-resistant processing," in *ACM International Conference on Supercomputing 25th Anniversary Volume, Munich, Germany*. ACM, June 2014, pp. 357–368.

[12] S. Bajaj and R. Sion, "Trusteddb: A trusted hardware-based database with privacy and data confidentiality," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 752–765, March 2014.

[13] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proc of the 2013 IEEE Symposium on Security and Privacy (S&P), Berkeley, California, USA*. IEEE, May 2013, pp. 238–252.

[14] C. Fournet, M. Kohlweiss, G. Danezis, and Z. Luo, "ZQL: A compiler for privacy-preserving data processing," in *Proc. of the 22nd USENIX Security Symposium (USENIX Security'13), Washington, D.C., USA*. USENIX, August 2013, pp. 163–178.

[15] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. of the 41st Annual ACM Symposium on Theory of Computing (STOC'09), Bethesda, Maryland, USA*. ACM, May 2009, pp. 169–178.

[16] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proc. of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11), Cascais, Portugal*. ACM, October 2011, pp. 85–100.

[17] S. D. Tetali, M. Lesani, R. Majumdar, and T. Millstein, "Mrcrypt: Static analysis for secure cloud computations," in *Proc. of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA'13), Indianapolis, Indiana, USA*. ACM, October 2013, pp. 271–286.

[18] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," in *Proc. of the 39th international conference on Very Large Data Bases (PVLDB'13), Trento, Italy*. VLDB Endowment, March 2013, pp. 289–300.

[19] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan, "Orthogonal security with cipherbase." in *Proc. of the 6th Conference on Innovative Data Systems Research (CIDR'13),Asilomar, California, USA*, January 2013, pp. 1–5.

[20] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "Vc3: Trustworthy data analytics in the cloud using sgx," in *Proc. of the 2015 IEEE Symposium on Security and Privacy (SP'15), San Jose, California, USA*. IEEE, May 2015, pp. 38–54.

[21] T. T. A. Dinh, P. Saxena, E.-C. Chang, B. C. Ooi, and C. Zhang, "M^{2}r: Enabling stronger privacy in mapreduce computation," in *Proc. of the 24th USENIX Security Symposium (USENIX Security'15), Washington, D.C., USA*. USENIX Association, August 2015, pp. 447–462.

[22] R. Pires, D. Gavril, P. Felber, E. Onica, and M. Pasin, "A lightweight mapreduce framework for secure processing with SGX," in *Proc. of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'17), Madrid, Spain*. IEEE Press, May 2017, pp. 1100–1107.

[23] N. Drucker and S. Gueron, "Combining Homomorphic Encryption with Trusted Execution Environment: A Demonstration with Paillier Encryption and SGX," in *Proc. of the 9th International Workshop on Managing Insider Security Threats (MIST'17), Dallas, Texas, USA*. ACM, October 2017, pp. 85–88.

[24] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. of the 1999 International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT'99), Prague, Czech Republic*, ser. Lecture Notes in Computer Science, no. 1592. Springer Berlin Heidelberg, May 1999, pp. 223–238.

[25] Intel, "Intel ® Software Guard Extensions Programming Reference," https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf, [Online; Accessed on March 1, 2018], October 2014.

[26] I. Anati, S. Gueron, S. P. Johnson, and V. R. Scarlata, "Innovative technology for cpu based attestation and sealing," Intel Corporation, August 2013, https://software.intel.com/sites/default/files/article/413939/hasp-2013-innovative-technology-for-attestation-and-sealing.pdf, [Online; Accessed on March 1, 2018].

[27] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions," in *Proc. of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP'13), Tel-Aviv, Israel*. ACM, June 2013, pp. 1–11.

[28] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel ®Software Guard Extensions: EPID provisioning and attestation services," https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services, [Online; Accessed on March 1, 2018], March 2016.

[29] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proc. of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP'13), Tel-Aviv, Israel*. ACM,

June 2013, pp. 1–10.

[30] N. Drucker, S. Gueron, and B. Pinkas, “Faster secure cloud computations with a trusted proxy,” *IEEE Security & Privacy*, vol. 15, no. 6, pp. 61–67, November 2017.

[31] Intel, “Intel unveils data center security strategy at 2017 rsa conference,” https://newsroom.intel.com/news/intel-unveils-data-center-security-strategy-2017-rsa-conference, [Online; Accessed on March 1, 2018], February 2017.

[32] P.-L. Aublin, F. Kelbert, D. O’Keeffe, D. Muthukumaran, C. Priebe, J. Lind, R. Krahn, C. Fetzer, D. Eyers, and P. Pietzuch, “Talos: Secure and transparent tls termination inside sgx enclaves,” Imperial college London, Tech. Rep. 2017/5, May 2017.

[33] Intel, “Intel®sgx software stack,” https://github.com/01org/linux-sgx [Online; Accessed on March 1, 2018], July 2017.

[34] N. Drucker and S. Gueron, “Paillier-encrypted databases with fast aggregated queries,” in *Proc. of the 14th IEEE Annual Consumer Communications Networking Conference (CCNC’17), Las Vegas, Nevada, USA*. IEEE, January 2017, pp. 848–853.

[35] A. C. Yao, “Protocols for secure computations,” in *Proc of the 23rd Annual Symposium on Foundations of Computer Science (SFCS’82), Chicago, Illinois, USA*. IEEE, November 1982, pp. 160–164.

[36] A. C. C. Yao, “How to generate and exchange secrets,” in *Proc. of the 27th Annual Symposium on Foundations of Computer Science (SFCS’86), Toronto, Canada*. IEEE, October 1986, pp. 162–167.

[37] S. Goldwasser, S. Micali, and A. Wigderson, “How to play any mental game, or a completeness theorem for protocols with an honest majority,” in *Proc. of the 19th Annual ACM Symposium on Theory of Computing (STOC’87), New Tork, New York, USA*, vol. 87, May 1987, pp. 218–229.

[38] G. Asharov, A. Jain, and D. Wichs, “Multiparty computation with low communication, computation and interaction via threshold FHE,” http://eprint.iacr.org/2011/613 [Online; Accessed on March 1, 2018], November 2011, cryptology ePrint Archive, Report 2011/613.

[39] A. López-Alt, E. Tromer, and V. Vaikuntanathan, “On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption,” in *Proc. of the 44th Annual ACM Symposium on Theory of Computing (STOC’12), New York, New York, USA*. ACM, May 2012, pp. 1219–1234.

[40] P. Mukherjee and D. Wichs, “Two Round Multiparty Computation via Multi-key FHE,” in *Proc. of the 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT’16), Vienna, Austria*. Springer Berlin Heidelberg, May 2016, pp. 735–763.

[41] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, “Multiparty computation from somewhat homomorphic encryption,” in *Advances in Cryptology - Proc. of the 2012 International Cryptology Conference (CRYPTO’12), Santa Barbara, California, USA*, ser. Lecture Notes in Computer Science, no. 7417. Springer Berlin Heidelberg, August 2012, pp. 643–662.

Author Biography

Drucker Nir is an applied scientist at Amazon Web Services and a PhD student at the University of Haifa. He also worked (2009-2017) at Intel Corporation as a senior software engineer as a member of the SGX team. His research focuses on software and hardware acceleration of cryptographic algorithms and protocols.

Gueron Shay is an associate professor at the University of Haifa. He also worked (2005-2017) at Intel Corporation as a Senior Principal Engineer, and served as the Chief Core Cryptography Architect of the CPU Architecture Group. He is now a Senior Principal Engineer at Amazon Web Services. His interests include cryptography, security, and algorithms. Shay is responsible for some of the recent CPU instructions that speed up cryptographic algorithms, such as the AES-NI and the carry-less multiplier instruction (PCLMULQDQ), the coming VPMADD52 instructions, and for various micro-architectural enhancements through the generations of the Core. He has contributed software to open source libraries, such as OpenSSL and NSS, offering significant performance gains to encryption, authenticated encryption, public key algorithms, and hashing. Shay was one of the architects of Intel Software Guard Extensions (SGX), in charge of its cryptographic definition and implementation, and the inventor of the Memory Encryption Engine. He is a co-author of the nonce misuse resistant mode AES-GCM-SIV, which is currently a CFRG draft.