Twin Clouds: Secure Cloud Computing with Low Latency

(Full Version)*

Sven Bugiel, Stefan Nürnberger, Ahmad-Reza Sadeghi, Thomas Schneider

Center for Advanced Security Research Darmstadt,
Technische Universität Darmstadt, Germany
{sven.bugiel,stefan.nuernberger,ahmad.sadeghi,thomas.schneider}
@trust.cased.de

Abstract. Cloud computing promises a cost effective enabling technology to outsource storage and massively parallel computations. However, existing approaches for provably secure outsourcing of data and arbitrary computations are either based on tamper-proof hardware or fully homomorphic encryption. The former approaches are not scaleable, while the latter ones are currently not efficient enough to be used in practice.

We propose an architecture and protocols that accumulate slow secure computations over time and provide the possibility to query them in parallel on demand by leveraging the benefits of cloud computing. In our approach, the user communicates with a resource-constrained Trusted Cloud (either a private cloud or built from multiple secure hardware modules) which encrypts algorithms and data to be stored and later on queried in the powerful but untrusted Commodity Cloud. We split our protocols such that the Trusted Cloud performs security-critical precomputations in the setup phase, while the Commodity Cloud computes the time-critical query in parallel under encryption in the query phase.

Keywords: Secure Cloud Computing, Cryptographic Protocols, Verifiable Outsourcing, Secure Computation

1 Introduction

Many enterprises and other organizations need to store and compute on a large amount of data. Cloud computing aims at renting such resources on demand. Today's cloud providers offer both, highly available storage (e.g., Amazon's Elastic Block Store [2]) and massively parallel computing resources (e.g., Amazon's Elastic Compute Cloud (EC2) with High Performance Computing (HPC) Clusters [3]) at low costs, as they can share resources among multiple clients.

* A preliminary version of this paper was published as extended abstract in [7].

On the other hand, sharing resources poses the risk of information leakage. Currently, there is no guarantee that security objectives stated in Service Level Agreements (SLA) are indeed fulfilled. Consequently, when using the cloud, the client is forced to blindly trust the provider's mechanisms and configuration [9]. However, this is accompanied by the risk of data leakage and industrial espionage due to a malicious insider at the provider or due to other customers with whom they share physical resources in the cloud [32]. Example applications that need to protect sensitive data include, but are not limited to, processing of personal health records or payroll databases. Access usually occurs not very frequently, but needs to be processed very fast while privacy of the data should be preserved.

Due to regulatory reasons, contractual obligations, or protection of intellectual property, cloud clients require *confidentiality* of their outsourced data, that computations on their data were processed correctly (*verifiability*), and that no tampering happened (*integrity*). Secure outsourcing of *arbitrary* computations on data is particularly difficult to fulfill if the client does not trust the cloud provider at all. Some cryptographic methods allow specific computations on encrypted data [4,18], or to securely and verifiably outsource storage [24].

Secure computation of *arbitrary* functions, e.g., arbitrary statistics or queries, on confidential data can be achieved based on fully homomorphic encryption as shown in [10,8]. However, these schemes are not yet usable in practice due to their poor efficiency. Furthermore, in a multi-client scenario, cryptography alone is not sufficient and additional assumptions have to be made such as using tamper-proof hardware [42]. Still, secure hardware which provides a shielded execution environment does not scale well as it is expensive and relatively slow.

Our Approach. We propose a model for secure computation of arbitrary functions with *low latency* using two clouds (twins). The resource-constrained *Trusted Cloud* is used for pre-computations whereas the untrusted, but powerful *Commodity Cloud* is used to achieve low latency (cf. Fig. 1). Our approach allows to separate the computations into their security and performance aspects: security-critical operations are performed by the Trusted Cloud in the *Setup Phase*, whereas performance-critical operations are performed on encrypted data in parallel by the Commodity Cloud in the *Query Phase*. Analogous to electricity, this can be seen as a battery that can be charged over night with limited amperage and later provides energy rapidly during discharge.

In the Setup Phase, the Trusted Cloud encrypts the outsourced data and programs using Garbled Circuits (GC) [43] which requires only symmetric cryptographic operations and a constant amount of memory. In the time-critical Query Phase, the Trusted Cloud verifies the results computed by the Commodity Cloud under encryption. Our proposed solution is transparent as the Client uses the Trusted Cloud as a proxy that provides a clearly defined interface to manage the outsourced data, programs, and queries. We minimize the communication over the secure channel (e.g., SSL/TLS) between Client and Trusted Cloud.

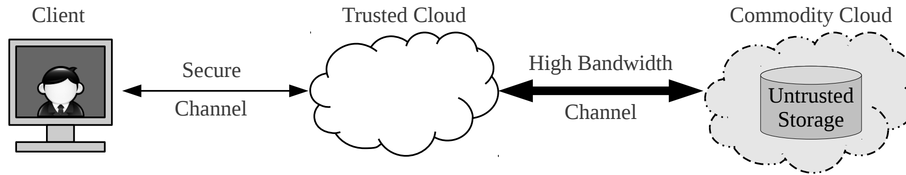

Fig. 1. Twin Clouds model with Client, Trusted Cloud, and Commodity Cloud.

Outline and Contribution. After summarizing related work in §2 and preliminaries in §3 we present the following contributions in the respective sections: In §4 we present our model for secure outsourcing of data and arbitrary computations with low latency using two clouds. The *Trusted Cloud* is mostly involved in the Setup Phase, while queries are evaluated under encryption and in parallel by the untrusted *Commodity Cloud*. In §5 we give an instantiation of our model based on GCs, the currently most efficient method for secure computation.

Our proposed solution has several advantages over previous proposals (cf. §2):

1. *Communication Efficiency.* We minimize the communication between the client and the Trusted Cloud as only a program, i.e., a very compact description of the function, is transferred and compiled on-the-fly into a circuit.
2. *Transparency.* The client communicates with the Trusted Cloud over a secure channel and clear interfaces that abstract from the underlying cryptography.
3. *Scalability and Low Latency.* Our approach is highly scalable as both clouds can be composed from multiple nodes. In the Query Phase, the Trusted Cloud performs only few computations (independent of the function's size).
4. *Multiple Clients.* Our protocols can be extended to multiple clients such that the Commodity Cloud securely and non-interactively computes on the clients' input data.

2 Related Work

Here, we summarize related works for secure outsourcing of storage and arbitrary computations based on Trusted Computing (§2.1), Secure Hardware (§2.2), Secure Computation (§2.3), and Architectures for Secure Cloud Computing (§2.4).

2.1 Trusted Computing

The most prominent approach to Trusted Computing technology was specified by the Trusted Computing Group (TCG) [40]. The TCG proposes to extend common computing platforms with trusted components in software and hardware, which enable the integrity measurement of the platform's software stack

at boot-/load-time (*authenticated boot*, [35]) and the secure reporting of these measurements to a remote party (*remote attestation*, [14]). Thus, it provides the means to achieve verifiability and transparency of a trusted platform's software state. Trusted Computing enables the establishment of trusted execution environments in commodity cloud infrastructures [37,36]. However, the reliable and efficient attestation of execution environments at run-time remains an open research problem. Trusted Computing is orthogonal to our approach and could be used to augment the Trusted Cloud with attestation capabilities.

2.2 Secure Hardware / HSMs

Cryptographic co-processors, such as the IBM 4765 or 4764 [19], provide a high-security, tamper-resistant execution environment for sensitive cryptographic operations. Such co-processors are usually certified, e.g., according to FIPS or Common Criteria. Hardware Security Modules (HSM) or Smartcards additionally provide a secure execution environment to execute custom programs. As secure hardware is usually expensive, relatively slow, and provides only a limited amount of secure memory and storage, it does not qualify as building block for a cost-efficient, performant, and scalable cloud computing infrastructure.

2.3 Secure Computation

Secure computation allows mutually distrusting parties to securely perform computations on their private data without involving a trusted third party. Existing approaches for secure computation are either based on computing with encrypted functions (called Garbled Circuits), or computing on encrypted data (using homomorphic encryption) as summarized in the following.

Garbled Circuits. Yao's Garbled Circuits (GC) [43] allow secure computation with encrypted functions. On a high level, one party (called constructor) "encrypts" the function to be computed using symmetric cryptography and later, the other party (called evaluator) decrypts the function using keys that correspond to the input data (called "garbled values"). We give a detailed description of GCs later in §3.2. Although GCs are very efficient as they use only symmetric cryptographic primitives, their main disadvantage is that each GC can be evaluated only once and its size is linear in the size of the evaluated function. As used in several works (e.g., [28,1,13,23,17]), the trusted GC creator can generate GCs in a setup phase and subsequently GCs are evaluated by one or more untrusted parties. Afterwards, the GC creator can verify efficiently that the computations indeed have been performed correctly (*verifiability*). Our protocols in §5 use GCs in commodity clouds that are composed from off-the-shelf hardware. In particular our protocols do not require that the cloud is equipped with trusted hardware modules (as proposed in [20,21,34,26]), while they could benefit from hardware accelerators such as FPGAs or GPUs (cf. [23]).

Homomorphic Encryption. Homomorphic Encryption (HE) allows to compute on encrypted data without using additional helper information. Traditional HE schemes are restricted to specific operations (e.g., multiplications for RSA [33], additions for Paillier [29], or additions and up to one multiplication for [6]). They allow to outsource specific computations, e.g., encryption and signatures [18], to untrusted workers, but require interaction to compute arbitrary functions. Recently, Fully HE (FHE) schemes have been proposed for arbitrary computations on encrypted data [11,38,41]. When combined with GCs for verifiability (cf. above), FHE allows to securely outsource data and arbitrary computations [10,8]. However, FHE is not yet sufficiently efficient to be used in practical applications [38,12].

Multiple Data Owners. The setting of secure outsourcing of data and computations can be generalized to multiple parties who provide their encrypted inputs to an untrusted server that non-interactively computes the verifiably correct result under encryption. However, using cryptography alone, this is only possible for specific functions [15], but not arbitrary ones [42]. This impossibility result can be overcome by using a trusted third party, in our case the Trusted Cloud.

2.4 Architectures for Secure Cloud Computing

We combine advantages of the following architectures for secure cloud computing.

An architecture for Signal Processing in the Encrypted Domain (SPED) in commodity computing clouds is described in [39]. SPED is based on cryptographic concepts such as secure multiparty computation or homomorphic encryption, which enable the secure and verifiable outsourcing of the signal processing. The authors propose a middleware architecture on top of a commodity cloud which implements secure signal processing by using SPED technologies. The client communicates via a special API, provided by a client-side plugin, with the middleware in order to submit new inputs and retrieve results. However, the authors do not elaborate on how to instantiate their protocols efficiently and do not answer problems regarding the feasibility of their approach. For instance, if GCs are used, they need to be transferred between the client-side plugin and the middleware which requires a large amount of communication. We parallelize the client plugin within the Trusted Cloud, provide a clear API that abstracts from cryptographic details, and give complete protocols.

Another architecture for secure cloud computing was proposed in [34]. The authors propose to use a tamper-proof hardware token which generates GCs in a setup phase that are afterwards evaluated in parallel by the cloud. The token receives the description of a boolean circuit and generates the corresponding GC using a constant amount of memory (using the protocol of [22]). The hardware token is integrated into the infrastructure of the cloud service provider either in form of a Smartcard provided by the client, or as a cryptographic co-processor.

We overcome several restrictions of this architecture by transferring smaller program descriptions instead of boolean circuits, virtualizing the hardware token in the Trusted Cloud, and providing a clear API for the client.

This idea of secure outsourcing of data and computations based on a tamper-proof hardware token was extended to the multi-cloud scenario in [26]. In this scenario, multiple non-colluding cloud providers are equipped with a tamper-proof hardware token each. On a conceptual level, the protocol of [26] is similar to that of [34]: The token outputs helper information, i.e., multiplication tuples (resp. garbled tables in [34]), to the associated untrusted cloud provider who uses this information within a secure multi-party computation protocol executed among the cloud providers (resp. for non-interactive computation under encryption) based on additive secret-sharing (resp. garbled circuits). The tokens in both protocols need to implement only symmetric cryptographic primitives (e.g., AES or SHA) and require only a constant amount of memory. In contrast, our Twin Clouds protocol is executed between two clouds (one trusted and one untrusted) and does not require trusted hardware.

3 Preliminaries

Our constructions make use of the following building blocks.

3.1 Encryption and Authentication

Confidentiality and authenticity of data can be guaranteed with symmetric cryptography: either with a combination of symmetric encryption (e.g., AES) and a Message Authentication Code (MAC, e.g., HMAC), or by using authenticated encryption, a special mode of operation of a block cipher (e.g., EAX [5]).

Notation. $\widehat{x} = \mathrm{AuthEnc}(x)$ denotes the authentication and encryption of data x; $x = \mathrm{DecVer}(\widehat{x})$ denotes the corresponding verification and decryption process.

3.2 Garbled Circuits (GC)

Arbitrary functions can be computed securely based on Yao's Garbled Circuits (GC) [43]. Compared to FHE (cf. §2.3), GCs are highly efficient as they use only symmetric cryptographic primitives but require helper information (cf. Fig. 2).

The main idea of GCs is that the *constructor* generates an encrypted version of the function f (represented as boolean circuit), called *garbled circuit* $\widetilde{f}$. For this, it assigns to each wire W_i of f two randomly chosen garbled values $\widetilde{w}_i^0$, $\widetilde{w}_i^1$ that correspond to the respective values 0 and 1. Note that $\widetilde{w}_i^j$ does not reveal any information about its plain value j as both keys look random. Then, for each gate of f, the constructor creates helper information in form of a *garbled*

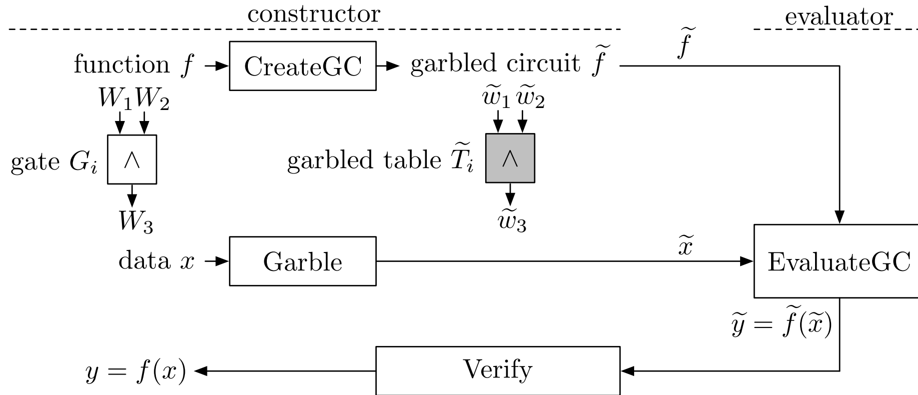

Fig. 2. Overview of Garbled Circuits

table $\widetilde{T}_i$ that allows to decrypt only the output key from the gate's input keys (details below). The garbled circuit $\widetilde{f}$ consists of the garbled tables of all gates. Later, the *evaluator* obtains the garbled values $\widetilde{x}$ corresponding to the inputs x of the function and evaluates the garbled circuit $\widetilde{f}$ by evaluating the garbled gates one-by-one using their garbled tables. Finally, the evaluator obtains the corresponding garbled output values $\widetilde{y}$ which allow the constructor to decrypt them into the corresponding plain output $y = f(x)$.

Security and Verifiability. GCs are secure against malicious evaluator (cf. [13]) and demonstration of valid output keys implicitly proves that the computation was performed correctly (cf. [10]). To guarantee security and verifiability, a GC can be evaluated only *once*, i.e., a new GC must be created for each evaluation.

Efficient GC constructions. The efficient GC construction of [25], provably secure in the random oracle model, provides "free XOR" gates, i.e., XOR gates have no garbled table and negligible cost for evaluation. For each 2-input non-XOR gate the garbled table has size $\approx 4t$ bits, where t is the symmetric security parameter (e.g., $t = 128$); creation of the garbled table requires 4 invocations of a cryptographic hash function (e.g., SHA-256) and evaluation needs 1 invocation. As shown in [22], *generation* of GCs requires only a constant amount of memory (independent of the size of the evaluated function) and only symmetric cryptographic operations (e.g., SHA-256). The implementation results of [31] show that *evaluation* of GCs can be performed efficiently on today's hardware: GC evaluation of the reasonably large AES functionality (22,546 XOR; 11,334 non-XOR gates) took 2s on a single core of an Intel Core 2 Duo with 3.0 GHz.

Notation. $\widetilde{x}$ is the garbled value corresponding to x. $\widetilde{C}$ is the GC for boolean circuit C (with $|C|$ non-XOR gates). $\widetilde{y} = \widetilde{C}(\widetilde{x})$ denotes evaluation of $\widetilde{C}$ on $\widetilde{x}$.

3.3 Circuit Compiler

The functions to be computed securely can be expressed in a compact way in a hardware description language and compiled automatically into a boolean circuit. A prominent example is Fairplay's [27] Secure Function Description Language (SFDL) which resembles a simplified version of a hardware description language, e.g., Verilog or VHDL (Very high speed integrated circuit Hardware Description Language), and supports types, variables, functions, boolean operators ($\wedge, \vee, \oplus, \dots$), arithmetic operators ($+, -$), comparison ($<, \geq, =, \dots$), and control structures like if-then-else or for-loops with constant range. Other candidates for compact description and compilation into boolean circuits are the languages and tools provided by [30,16]. As shown in [16], the compilation into a circuit can be implemented with a low memory footprint. In principle, it would be possible to compile algorithms formulated in any standard programming language such as C or Java into a boolean circuit, as every computable function can be expressed as boolean circuit of polynomial size.

Notation. $C = \text{Compile}(P)$ denotes compilation of program P into circuit C.

4 Twin Clouds Model

Our Twin Clouds model, depicted in Fig. 1 on page 3, allows secure outsourcing of data and arbitrary computations with low latency to an untrusted commodity cloud. In our model, the Client makes use of the services offered by a cloud service provider to outsource its data and computations thereon into the provider's Commodity Cloud in a secure way. The confidentiality and the integrity of the outsourced data must be protected against a potentially malicious provider, and the correctness of the outsourced computations must be verifiable by the Client.

Due to the assumed large size of the Client's data and/or the computational complexity of the computations thereon, it is not possible to securely outsource the data to the Commodity Cloud and let the Client execute its computations locally after retrieving back the entire data. Instead, the computations must be performed by the Commodity Cloud without interaction with the Client.

To achieve these goals and satisfy the above mentioned security requirements, the Twin Cloud model uses a *Trusted Cloud* as proxy between the Client and the Commodity Cloud. The Trusted Cloud provides a resource-restricted execution environment and infrastructure that is fully trusted by the Client. As the resources of the Trusted Cloud are restricted, relatively expensive, and potentially slow, the computations can also not be performed within the Trusted Cloud.

Instead, the Trusted Cloud is a transparent proxy that adds the needed security properties (*integrity, confidentiality, verifiability*) on top of the services provided by the fast but insecure Commodity Cloud. It provides an interface for

secure storage and computations to the Client while abstracting from the service provider's cloud infrastructure. This interface (e.g., a web-frontend or API) allows to securely submit data, programs, and queries to be securely stored and computed. The low-bandwidth connection between Client and Trusted Cloud is secured by a secure channel (e.g., SSL/TLS).

The Trusted Cloud is used mostly during a *Setup Phase*, but performs only few computations during the time-critical *Query Phase*. It is assumed to have a small amount of storage only; if larger amounts of data need to be stored, they can be securely outsourced to the Commodity Cloud's untrusted storage. To allow this secure outsourcing of storage, the Trusted Cloud is connected to the Commodity Cloud over an unprotected high-bandwidth channel.

A possible instantiation of the Trusted Cloud can be a private cloud of the Client (e.g., his existing IT infrastructure). Alternatively, the Trusted Cloud could be a cluster of virtualized cryptographic co-processors (e.g., the IBM 4765 [19] or other Hardware Security Modules) which are offered as a service by a third party and which provide the necessary hardware-based security features to implement a secure remote execution environment trusted by the Client.

5 Twin Clouds Protocols

To efficiently instantiate the Twin Clouds model of §4 we use a "battery" for secure computations: In the Setup Phase, the battery is charged by precomputing encrypted (garbled) data and functions within the resource-limited Trusted Cloud. Later, in the Query Phase, the battery is rapidly discharged by evaluating these encryptions in parallel within the Commodity Cloud.

Simplification. To ease presentation, we assume a single client who outsources a single program P. However, our protocols naturally extend to multiple programs and clients. We also assume that the Trusted Cloud takes appropriate measures to protect against replay attacks, e.g., an internal database of randomly chosen keys for each authenticated encryption and GC with associated garbled data.

Interface. The Client accesses the Trusted Cloud over a secure channel and the following interface which abstracts from all underlying cryptographic details: During the *Setup Phase*, the Client provides the data D to be outsourced and the program P (formulated in a Hardware Description Language, cf. §3.3) to be computed. Later, in the *Query Phase*, the Client issues a query q which should be processed as fast as possible resulting in the response $r = P(q, D)$ output to the Client. Additionally, the Client can update the stored data D or program P.

Protocol Overview. On a high-level, our protocols work as follows: The Trusted Cloud stores Client's data D and program P securely in the Commodity Cloud. Then, the Trusted Cloud retrieves back D and re-encrypts it into its garbled equivalent $\widetilde{D}$, and generates GCs $\widetilde{C}$ from P; both are stored in the Commodity Cloud. Later, the Client's query q is encrypted and sent to the Commodity Cloud which computes the garbled result $\widetilde{r} = \widetilde{C}(\widetilde{q}, \widetilde{D})$ under encryption (using a pre-computed $\widetilde{C}$ which is deleted afterwards). Finally, the Trusted Cloud verifies the garbled result and sends $r = P(q, D)$ to the Client.

We describe the details of the two phases next. Actions invoked by the Client are denoted by Latin letters and automatically triggered actions by Greek ones.

5.1 Setup Phase

The *Setup Phase*, depicted in Fig. 3, consists of the following use-cases.

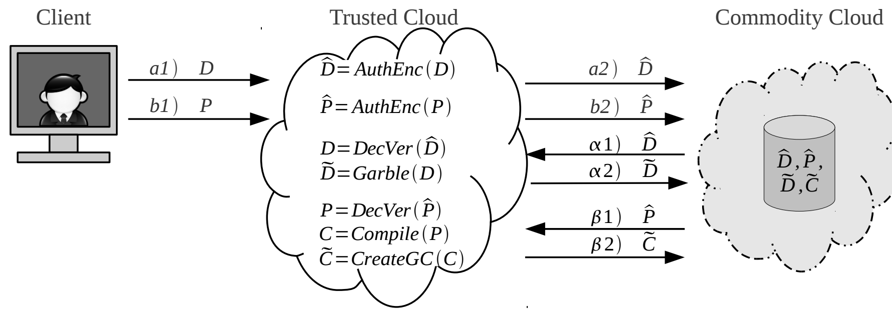

Fig. 3. *Setup Phase*: a,b) Client registers data D and program P to be stored securely in the *Commodity Cloud.* α) Updates of D require re-generation of garbled data $\widetilde{D}$. β) Updates of P require re-generation of garbled circuits $\widetilde{C}$.

a) Modify Data. When the Client provides new or modified data D to be outsourced (a1), D is stored securely as $\widehat{D} = \text{AuthEnc}(D)$ (cf. §3.1) in the Commodity Cloud (a2). Whenever D is modified, the garbled data $\widetilde{D}$ is re-generated (cf. α below) and all pre-computed GCs $\widetilde{C}$ can be deleted.

b) Modify Program. Whenever the Client provides a new or modified program P (b1), P is stored securely as $\widehat{P} = \text{AuthEnc}(P)$ (cf. §3.1) in the Commodity Cloud (a2). Whenever P is modified, all pre-computed GCs $\widetilde{C}$ can be deleted.

α) *Garble Data.* Whenever D is changed, the garbled data $\widetilde{D}$ must be regenerated. For this, the Trusted Cloud requests the securely stored data $\widehat{D}$ from the Commodity Cloud ($\alpha1$), recovers the data $D = \text{DecVer}(\widehat{D})$ (cf. §3.1), generates the corresponding garbled data $\widetilde{D} = \text{Garble}(D)$ (cf. §3.2), and stores this back into the Commodity Cloud ($\alpha2$).

β) *Garble Program.* Whenever D or P is changed or the Trusted Cloud has capacities for pre-computations, new GCs are generated. For this, the Trusted Cloud requests the securely stored program $\widehat{P}$ from the Commodity Cloud ($\beta1$), recovers the program $P = \text{DecVer}(\widehat{P})$ (cf. §3.1), compiles it into a boolean circuit $C = \text{Compile}(P)$ (cf. §3.3), generates a new GC $\widetilde{C} = \text{Garble}(C)$ (cf. §3.2), and stores this back into the Commodity Cloud ($\beta2$).

5.2 Query Phase

The query phase depicted in Fig. 4 consists of the following use-case:

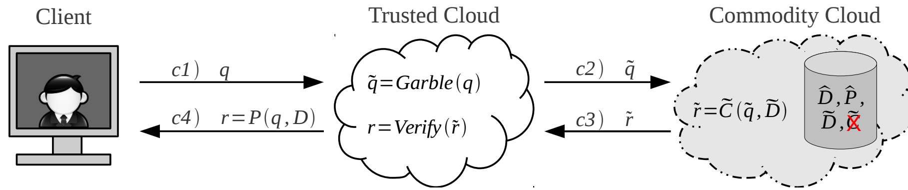

Fig. 4. *Query Phase*: *Client* sends query q to the Trusted Cloud to be computed by the Commodity Cloud under encryption (c). The used GC $\widetilde{C}$ is deleted afterwards.

c) Process Query. When the Client sends a query q for secure evaluation (c1), the Trusted Cloud converts q into its garbled equivalent $\widetilde{q} = \text{Garble}(q)$ (cf. §3.2) which is forwarded to the Commodity Cloud (c2). The Commodity Cloud computes the garbled response $\widetilde{r} = \widetilde{C}(\widetilde{q}, \widetilde{D})$ by evaluating a pre-computed GC $\widetilde{C}$ (cf. §3.2) in parallel and deleting it afterwards. The garbled result $\widetilde{r}$ is returned to the Trusted Cloud (c3) which verifies the correctness of the result $r = \text{Verify}(\widetilde{r})$ (cf. §3.2) and returns $r = P(q, D)$ to the Client.

5.3 Analysis

In the following we analyze the security and efficiency properties of our protocols.

Security Analysis. The security of our protocols stems from the fact that the Trusted Cloud is a secure execution environment, whereas the adversary can have full control over the Commodity Cloud and all communication channels. More specifically, our protocols are secure against a malicious Commodity Cloud provider as well as external adversaries: The Commodity Cloud is neither able to successfully modify nor to learn the outsourced data $\widehat{D}$ or program $\widehat{P}$ as these are authenticated and encrypted (cf. §3.1). The security and verifiability properties of GCs (cf. §3.2) ensure that the Commodity Cloud also cannot successfully modify or learn $\widetilde{q}, \widetilde{D}, \widetilde{C}, \widetilde{r}$, or intermediate results of the computation. Clearly, the Commodity Cloud learns an upper bound on the size of all data which can be circumvented by appropriate padding. The same holds true for external attackers that also cannot interfere with the communication between Client and Trusted Cloud due to the usage of a secure channel (e.g., SSL/TLS).

Efficiency Analysis. The *communication* between Client and Trusted Cloud is minimized as only data and a compact program are transferred over the secure channel, while the communication between Trusted Cloud and Commodity Cloud is dominated by the transfer of $\widetilde{C}$ of size $\approx 4t \cdot |C|$ bits, where t is the symmetric security parameter, e.g., $t = 128$ (cf. §3.2). The Commodity Cloud's *storage* is dominated by $t \cdot (|D| + 4|C|)$ bits for $\widetilde{D}$ and $\widetilde{C}$, while the Trusted Cloud needs only low memory/storage. The dominating factor of the *computation* complexity are $4|C|$ hash function evaluations by the Trusted Cloud in the Setup Phase and $|C|$ parallel hash function evaluations by the Commodity Cloud in the Query Phase. Note that many functionalities such as queries on or statistics over large databases naturally allow parallelization.

Finally, we'd like to emphasize that our protocols can be used to securely outsource data and *arbitrary* computations thereon, use only symmetric-key cryptographic primitives, and do not rely on tamper-proof hardware. A prototype implementation to verify their practical efficiency is left as future work.

Acknowledgements

We thank Radu Sion for pointing out the analogy of our Twin Clouds model with a rechargeable battery that accumulates energy (computations) over some time and can then be uncharged rapidly. This work was in part funded in part by the European Commission through the ICT program under contract 257243 TClouds and 216676 ECRYPT II.

References

1. J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *Security and Privacy*, pages 2–11. IEEE, 2001.
2. Amazon. Elastic Block Store (EBS). `http://aws.amazon.com/ebs`, 2011.
3. Amazon. Elastic Compute Cloud (EC2). `http://aws.amazon.com/ec2`, 2011.
4. M. Atallah, K. Pantazopoulos, J. Rice, and E. Spafford. Secure outsourcing of scientific computations. *Advances in Computers*, 54:216–272, 2001.
5. M. Bellare, P. Rogaway, and D. Wagner. The EAX mode of operation: A two-pass authenticated-encryption scheme optimized for simplicity and efficiency. In *FSE'04*, volume 3017 of *LNCS*, pages 389–407. Springer, 2004.
6. D. Boneh, E. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC'05*, volume 3378 of *LNCS*, pages 325–341. Springer, 2005.
7. Sven Bugiel, Stefan Nürnberger, Ahmad-Reza Sadeghi, and Thomas Schneider. Twin Clouds: An architecture for secure cloud computing (Extended Abstract). Workshop on Cryptography and Security in Clouds (WCSC'11), March 15-16, 2011.
8. K. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO'10*, volume 6223 of *LNCS*, pages 483–501. Springer, 2010.
9. Cloud Security Alliance. Top threats to cloud computing, v. 1.0, 2010.
10. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO'10*, volume 6223 of *LNCS*, pages 465–482. Springer, 2010.
11. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC'09*, pages 169–178. ACM, 2009.
12. Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *EUROCRYPT'11*, 2011. To appear.
13. S. Goldwasser, Y. Kalai, and G. Rothblum. One-time programs. In *CRYPTO'08*, volume 5157 of *LNCS*, pages 39–56. Springer, 2008.
14. Trusted Computing Group. Trusted platform module (TPM) main specification, 2007.
15. Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. Cryptology ePrint Archive, 2011/157, 2011.
16. W. Henecka, S. Kögl, A. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. In *CCS*, pages 451–462. ACM, 2010.
17. A. Herzberg and H. Shulman. Secure guaranteed computation. Cryptology ePrint Archive, Report 2010/449, 2010.
18. S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In *TCC'05*, volume 3378 of *LNCS*, pages 264–282. Springer, 2005.
19. IBM. Cryptocards. `http://www-03.ibm.com/security/cryptocards/`, 2011.
20. A. Iliev. *Hardware-Assisted Secure Computation*. PhD thesis, Dartmouth College, Hanover, NH, USA, 2009.
21. A. Iliev and S. Smith. Small, stupid, and scalable: secure computing with Faerieplay. In *Workshop on Scalable Trusted Computing (STC'10)*, pages 41–52. ACM, 2010.
22. K. Järvinen, V. Kolesnikov, A. Sadeghi, and T. Schneider. Embedded SFE: Offloading server and network using hardware tokens. In *FC'10*, volume 6052 of *LNCS*, pages 207–221. Springer, 2010.

23. K. Järvinen, V. Kolesnikov, A. Sadeghi, and T. Schneider. Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs. In *CHES'10*, volume 6225 of *LNCS*, pages 383–397. Springer, 2010.
24. S. Kamara and K. Lauter. Cryptographic cloud storage. In *Real-Life Cryptographic Protocols and Standardization*, volume 6054 of *LNCS*, pages 136–149. Springer, 2010.
25. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP'08*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.
26. J. Loftus and N. Smart. Secure outsourced computation. In *AFRICACRYPT'11*, 2011. To appear.
27. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *Security*, pages 287–302. USENIX, 2004.
28. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Electronic Commerce (EC'99)*, pages 129–139. ACM, 1999.
29. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.
30. A. Paus, A. Sadeghi, and T. Schneider. Practical secure evaluation of semi-private functions. In *ACNS'09*, volume 5536 of *LNCS*, pages 89–106. Springer, 2009.
31. B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *ASIACRYPT'09*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.
32. T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *CCS'09*, pages 199–212. ACM, 2009.
33. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21:120–126, 1978.
34. A. Sadeghi, T. Schneider, and M. Winandy. Token-based cloud computing: Secure outsourcing of data and arbitrary computations with lower latency. In *TRUST'10 Workshop on Trust in the Cloud*, volume 6101 of *LNCS*, pages 417–429. Springer, 2010.
35. R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Security*. USENIX, 2004.
36. N. Santos, K. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In *Hot Topics in Cloud Computing (HotCloud'09)*. USENIX, 2009.
37. J. Schiffman, T. Moyer, H. Vijayakumar, T. Jaeger, and P. McDaniel. Seeding clouds with trust anchors. In *CCSW'10*, pages 43–46. ACM, 2010.
38. N. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *PKC'10*, volume 6056 of *LNCS*, pages 420–443. Springer, 2010.
39. J. R. Troncoso-Pastoriza and F. Pérez-González. CryptoDSPs for cloud privacy. In *Workshop on Cloud Information System Engineering (CISE'10)*, 2010.
40. Trusted Computing Group. `http://www.trustedcomputinggroup.org`, 2011.
41. M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT'10*, volume 6110 of *LNCS*, pages 24–43. Springer, 2010.
42. M. Van Dijk and A. Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *HotSec'10*, pages 1–8. USENIX, 2010.
43. A. C.-C. Yao. How to generate and exchange secrets. In *FOCS'86*, pages 162–167. IEEE, 1986.