Twin Clouds: An Architecture for Secure Cloud Computing

(Extended Abstract)

Sven Bugiel[1], Stefan Nürnberger[1], Ahmad-Reza Sadeghi[1], Thomas Schneider[2]

[1] Center for Advanced Security Research Darmstadt, Technische Universität Darmstadt, Germany
{sven.bugiel,stefan.nuernberger,ahmad.sadeghi}@trust.cased.de*
[2] System Security Lab, Ruhr-University Bochum, Germany
thomas.schneider@trust.rub.de**

Abstract. Cloud computing promises a more cost effective enabling technology to outsource storage and computations. Existing approaches for secure outsourcing of data and arbitrary computations are either based on a single tamper-proof hardware, or based on recently proposed fully homomorphic encryption. The hardware based solutions are not scaleable, and fully homomorphic encryption is currently only of theoretical interest and very inefficient.

In this paper we propose an architecture for secure outsourcing of data and arbitrary computations to an untrusted commodity cloud. In our approach, the user communicates with a trusted cloud (either a private cloud or built from multiple secure hardware modules) which encrypts and verifies the data stored and operations performed in the untrusted commodity cloud. We split the computations such that the trusted cloud is mostly used for security-critical operations in the less time-critical setup phase, whereas queries to the outsourced data are processed in parallel by the fast commodity cloud on encrypted data.

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

1 Introduction

Many enterprises and other organizations need to store and operate on a huge 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 (EBS) [AmEa]) and massively parallel computing resources (e.g., Amazon's Elastic Compute Cloud (EC2) with High Performance Computing (HPC) Clusters [AmEb]) at relatively low costs.

While cloud computing is promised to be cost-effective and provides more flexibility for the clients, it introduces security risks organizations have to deal with in order to isolate their data from other cloud clients and to fulfil confidentiality and integrity demands of their customers. Moreover, since the IT infrastructure is under control of the cloud provider, the customer has not only to trust the security mechanisms and configuration of the cloud provider, but also the cloud provider itself. When data and computation is outsourced to the cloud, prominent security risks are: malicious code that is running on the cloud infrastructure could manipulate computation and force wrong results or steal data; personnel of the cloud provider could misuse their capabilities and leak data and vulnerabilities in the shared resources could lead to data leakage or manipulated computation [(CS10]. In general, important requirements of cloud clients are *confidentiality* of their data, that their data and computation was processed in the expected way (*verifiability*) and has not

* Supported by the European Commission through the ICT program under contract 257243 TClouds.
** Supported by the European Commission through the ICT program under contract 216676 ECRYPT II.

been tampered with (*integrity*). Example scenarios in which these properties must be guaranteed include processing sensitive data, e.g. outsourcing of medical data (cf. [NGSN10], Microsoft's HealthVault[3]).

Secure outsourcing of *arbitrary* computation and data storage is particularly difficult to fulfill if a cloud client does not trust the cloud provider at all. There are proposals for cryptographic methods which allow to perform specific computations on encrypted data [APRS01,HL05], or to securely and verifiably outsource storage [KL10a].

Secure computation of arbitrary functions on confidential data can be achieved based on fully homomorphic encryption [Gen09b] as shown in [GGP10,CKV10]. However, these schemes are not yet usable in practice due to their low efficiency. Further, in a multi-client scenario, cryptography alone is not sufficient and additional assumptions have to be made and other measures have to be taken such as tamper-proof hardware [DJ10]. Other approaches are based on secure hardware which provides a shielded execution environment. However, these approaches do not scale well as secure hardware is expensive and relatively slow (cf. §3 for a detailed discussion of related works).

Our Approach. The architecture we propose consists of two clouds (twins), a *Trusted Cloud* and a *Commodity Cloud*.

Our approach allows to separate the underlying computations into their security and performance aspects: the security-critical operations are performed by the Trusted Cloud in a Setup Phase, whereas the performance-critical operations are performed on encrypted data by the Commodity Cloud. This allows maximum utilization of the expensive resources of the Trusted Cloud, while high loads of queries can be processed on-demand by the Commodity Cloud. The Trusted Cloud requires only a constant amount of storage and is used constantly in the *Setup Phase* for pre-computing encryptions. The *untrusted* Commodity Cloud provides a large amount of storage and is used in the time-critical *Query Phase* to process encrypted queries in parallel with minimal latency.

More specifically, the client uses the Trusted Cloud as a proxy to securely outsource his data and computations to the untrusted Commodity Cloud. The client communicates to the Trusted Cloud over a secure channel (e.g., SSL/TLS) and a clearly defined interface (e.g., a web service API) which allows the client to manage the outsourced data, programs, and queries. We optimize the amount of data transferred between the client and the Trusted Cloud. The Trusted Cloud is used mostly in the Setup Phase to encrypt the outsourced data and programs using (improved versions of) Yao's garbled circuits [Yao86] which requires only symmetric cryptographic operations and only a constant amount of memory. Afterwards, in the Query Phase, the computations on the encrypted data are performed in parallel by the fast but untrusted Commodity Cloud, and finally verified by the Trusted Cloud.

We envision the Trusted Cloud to be either a private cloud (e.g., client's existing IT infrastructure) or composed from multiple secure hardware tokens (e.g., operated and offered by a service provider).

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 thereon consisting of 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 garbled circuits, the currently most efficient method for secure computation. Our proposed solution has several advantages over previous proposals:

1. *Communication Efficiency.* We minimize the communication between the client and the Trusted Cloud as only a compact description of the functionality in form of a program in a Hardware Description Language (HDL) is transferred and compiled on-the-fly into a circuit.

[3] `http://www.healthvault.com/`

2. *Transparency.* The client communicates to the Trusted Cloud over a secure channel and simple interfaces. All complicated cryptographic operations are hidden from the client.
3. *Scalability.* Our approach is highly scalable as both clouds can be composed from multiple nodes.
4. *Multiple Clients.* In contrast to solutions based on fully homomorphic encryption, our solution can be extended to multiple clients which operate on the same data.

2 Related Work

In the following 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

One approach to achieve trustworthy computations in cloud infrastructures is to adapt existing trusted computing solutions to the cloud computing paradigm or to use these solutions as building blocks in new cloud architecture models. The most prominent approach to Trusted Computing technology has been specified by the Trusted Computing Group[4] (TCG). The TCG proposes to extend common computing platforms with trusted components in software and hardware. In particular the hardware extension, called *Trusted Platform Module* (TPM) [Gro07] acts as a hardware trust anchor and enables the integrity measurement of the platform's software stack at boot-/load-time (*authenticated boot*) [SZJVD04] and the secure reporting of these measurements to a remote party (*remote attestation*) [Gro07]. Thus, it provides the means to achieve verifiability and transparency of a trusted platform's software state.

Trusted Computing, based on the TPM and its remote attestation feature, enables the establishment of trusted execution environments in commodity cloud infrastructures (as shown in [SMV+10,SGR09]). However, the reliable and efficient attestation of the execution environment at run-time is a research problem to which no complete solution exists yet.

Nevertheless, Trusted Computing approaches are in principal orthogonal to our approach presented in this paper and can be combined with our approach in order to augment the Trusted Cloud (cf. 4) with attestation capabilities.

2.2 Secure Hardware / HSMs

Secure computation based on dedicated security hardware is already well-established and applied for a wide range of applications, e.g., banking or governmental applications. For example, cryptographic co-processors such as the IBM 4765 or 4764 [IBM] provide a high-security, tamper-resistant execution environment for sensible 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 generic secure execution environment to execute customer supplied programs. However, cryptographic co-processors are usually very expensive and do not scale very well. Thus, they do not qualify as building blocks for a cost-efficient, performant and scalable cloud computing infrastructure.

More cost effective secure execution environments (on off-the-shelf hardware) are for instance ARM TrustZone [Lim05] for embedded and mobile devices, or IBM's Cell processor [SHL07]. However, this hardware is usually not tamper-resistant and thus does not qualify as remotely installed secure execution environment, e.g., in cloud service providers' infrastructures.

[4] `http://www.trustedcomputinggroup.org/`

2.3 Secure Computation

Secure computation allows mutually distrusting parties to securely perform computations on their private data without involving a trusted third party such as a secure hardware. 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) [Yao86] 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., [NPS99,ACCK01,GKR08,JKSS10b,HS10]), the trusted GC creator can generate GCs in a setup phase and subsequently GCs are evaluated by one or more untrusted parties. The GC creator is able to verify that the computations indeed have been performed correctly (*verifiability*). Our protocols shown in §5 also make use of these features of GCs and hence can be interpreted as a mapping of the aforementioned works to the specific requirements of the cloud computing scenario.

Although hardware tokens can be used to speed up the evaluation of GCs (as proposed in Faerieplay [Ili09,IS10] and [JKSS10b]) we evaluate GCs in software only in order to make use of commodity clouds composed of off-the-shelf hardware.

Homomorphic Encryption. Homomorphic encryption allows to compute on encrypted data without using additional helper information such as garbled circuits. Traditional homomorphic encryption schemes were restricted to specific operations (e.g., multiplications for RSA [RSA78], additions for Paillier [Pai99], or additions and up to one multiplication for [BGN05]). Such schemes allow to outsource specific computations, such as encryption and signatures [HL05], to untrusted workers.

Recently, fully homomorphic encryption schemes have been proposed that allow arbitrary computations on encrypted data [Gen09b,Gen09a,SV10,DGHV10]. When combined with garbled circuits for verifiability (cf. above), fully homomorphic encryption allows to securely outsource data and arbitrary computations as described in [GGP10,CKV10]. However, as shown in [SV10,GH10], fully homomorphic encryption is not yet sufficiently efficient to be used in practical applications. Further, the impossibility result of [DJ10] shows that using cryptography alone is not sufficient if data is shared among more than one client – in this case, additional assumptions need to be made.

2.4 Architectures for Secure Cloud Computing

Our model (cf. §4) and approach (§5) combine the advantages of the following two architectures for secure outsourcing of data and arbitrary combinations.

In [TPPG10] the authors describe an architecture for Signal Processing in the Encrypted Domain (SPED) in commodity computing clouds. 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 the details of their implementation and do not answer problems regarding the feasibility of their approach. For instance, if garbled circuits are used, the garbled circuits need to be transferred between the client-side plugin and the middleware which requires a huge amount of communication. In our approach, we

parallelize the client plugin within the trusted cloud and provide a clear API to the client which abstracts from the underlying cryptographic details. On top of a generic architecture we give a concrete instantiation for our protocols.

The authors of [SSW10] adapt the protocol of [JKSS10a] to the cloud computing scenario. They propose to use a tamper-proof hardware token which generates garbled circuits in a setup phase which are afterwards evaluated in parallel by the cloud. The token receives the description of a boolean circuit and generates a corresponding garbled circuit using a constant amount of memory (using the protocol of [JKSS10a]). 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 model by transferring substantially smaller program descriptions instead of boolean circuits, and virtualizing the hardware token in the trusted cloud.

3 Preliminaries

Our constructions make use of the following building blocks.

3.1 Encryption and Authentication

Confidentiality and authenticity of data can be guaranteed by means of symmetric cryptography: either with a combination of symmetric encryption (e.g., AES [NIS01]) and a Message Authentication Code (MAC, e.g., HMAC [KBC97]), or by using authenticated encryption, a special mode of operation of a block cipher (e.g., EAX [BRW04]). These schemes use a respective symmetric key for encryption/authentication and the same key for decryption/verification.

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

3.2 Garbled Circuits (GC)

The most efficient method for secure computation of arbitrary functions known today is based on Yao's garbled circuits (GC) [Yao86]. Compared to fully homomorphic encryption (cf. §2.3), GCs are highly efficient as they are based on symmetric cryptographic primitives only but require helper information to evaluate non-XOR gates as described below.

The main idea of GCsis 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 $\tilde{w}_i^0$, $\tilde{w}_i^1$ that correspond to the respective values 0 and 1. Note that $\tilde{w}_i^j$ 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 table* $\tilde{T}_i$ 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, 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 even against malicious evaluator (cf. [GKR08]) and demonstration of valid output keys implicitly proves that the computation was performed correctly (cf. [GGP10]). A fundamental property of GCs is that they can be evaluated only *once*, i.e., for each evaluation a new GC must be generated.

Efficient GC constructions. The efficient GC construction of [KS08] 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 $4t$ 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 [NIS02]) and evaluation needs 1 invocation. The construction is provably secure in the random-oracle model. As shown in [JKSS10a], *generation* of GCs requires only a constant amount of memory (independent of the size of the evaluated function) and only symmetric cryptographic operations (SHA-256 and AES). The implementation results of [PSSW09] show that *evaluation* of GCs can be performed efficiently on today's hardware: Evaluation of the GC for the reasonably large AES functionality (22,546 XOR and 11,334 non-XOR gates) took 2s on an Intel Core 2 Duo with 3.0 GHz and 4 GB RAM.

Notation. We write $\widetilde{x}$ for the garbled value corresponding to x and $\widetilde{f}$ for the garbled circuit of function f. Evaluation of $\widetilde{f}$ on garbled input $\widetilde{x}$ is written as $\tilde{y} = \tilde{f}(\tilde{x})$.

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 [MNPS04] Secure Function Description Language (SFDL) which resembles a simplified version of a hardware description language, such as Verilog or VHDL[5], 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 [PSS09,HKS+10,MK10]. As shown in [HKS+10], the compilation into a circuit can be implemented with a low memory footprint.

Notation. We write $C = \text{Compile}(P)$ to denote that the boolean circuit C was compiled from program P.

4 Our Model

In this section we present our model for secure outsourcing of data and arbitrary computations to an untrusted commodity cloud. Our model is depicted in Fig. 1.

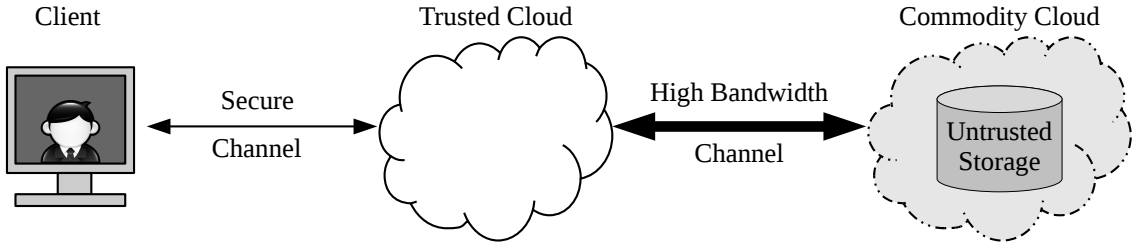

Fig. 1. *Our Model:* The client communicates with the *Trusted Cloud* over a low bandwidth, secure channel. The two clouds are connected with an insecure, high bandwidth channel. The *Commodity Cloud* further provides untrusted storage.

[5] Very high speed integrated circuit Hardware Description Language

In our model, a 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 outsourced data must be confidentiality and integrity protected (from a potentially malicious provider) and the correctness of the outsourced computations must be verifiable by the Client.

While this problem can be easily solved for a restricted class of computations (e.g., private search of a keyword using searchable encryption [KL10b]), we consider in our model the general case of arbitrary computations. Due to the assumed large size of the client's data (e.g., a database) and/or the computational complexity of the computations thereon, it is not possible to securely outsource the data into the cloud provider's infrastructure only and let the client execute its computations locally after retrieving its data from the provider's cloud. Instead, the confidentiality and integrity of the outsourced data has to be protected while at the same time secure computations on it need to be performed in the commodity cloud without interaction with the client.

In order to achieve these goals and satisfy the above mentioned security requirements of the client, our 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 cannot be performed directly within the Trusted Cloud.

Instead, the Trusted Cloud provides an interface for secure storage and computations to the client while abstracting from the service provider's cloud infrastructure. The interface offered by the Trusted Cloud to the client (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., via SSL/TLS).

The Trusted Cloud is used mostly during a pre-computation phase (*Setup Phase*), but performs only few computations during the time-critical *Query Phase*. The Trusted Cloud 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 using encryption and authentication (cf. §3.1). 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 based on a private cloud operated by 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 Cryptographic Coprocessor 4764 [SW99,IBM] or other Hardware Security Modules such as TPMs), which are offered as a service by a third party and which provide the necessary hardware-based security features to implement a verifiable remote execution environment trusted by the client.

5 Our Protocols

We give efficient protocols to instantiate the model presented in §4 next.

In order to achieve secure outsourcing of data and arbitrary computations with low latency query response times, we split our protocols into two phases. First, in the *Setup Phase* encryptions are pre-computed by the *Trusted Cloud* which are subsequently evaluated in the *Query Phase* by the *Commodity Cloud*.

The Trusted Cloud works as a transparent interface to add the needed security properties to the Commodity Cloud. Our architecture is then responsible for securely computing the program, while exploiting the advantages of a commodity fast cloud. Our approach aims at being transparent to the *Client* as it adds additional security features (*integrity, confidentiality, verifiability*) while benefiting from the performance of existing cloud infrastructures without their modification.

Simplification. To ease presentation we assume in the following that only a single program P can be computed on the outsourced data. Further, we concentrate on a scenario where only one client accesses the outsourced data. However, our protocols actually extend to multiple programs and clients.

Interface. The Client accesses the following interface offered by the Trusted Cloud over a secure channel: During the *Setup Phase*, the Client provides the data D to be outsourced as well as the program P (formulated in a Hardware Description Language) 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 the program P.

Protocol Overview. On a high-level, our protocols work as follows: As soon as the Client provides the data D or the program P, they are stored securely in the Commodity Cloud. Then, the Trusted Cloud starts to re-encrypt D into its garbled equivalent $\widetilde{D}$ and generates garbled circuits $\widetilde{C}$ for P that are stored in the Commodity Cloud. Later, when the Client issues a query q, it 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 garbled circuit which is deleted afterwards). Finally, the Trusted Cloud verifies the correctness of the garbled result and returns the result r to the Client.

In the following we describe the details of the two phases. Actions invoked by the Client are denoted in Latin letters; indices denote sub-steps. Greek letters denote actions that are triggered automatically.

5.1 Setup Phase

The setup phase depicted in Fig. 2 consists of the following four use-cases.

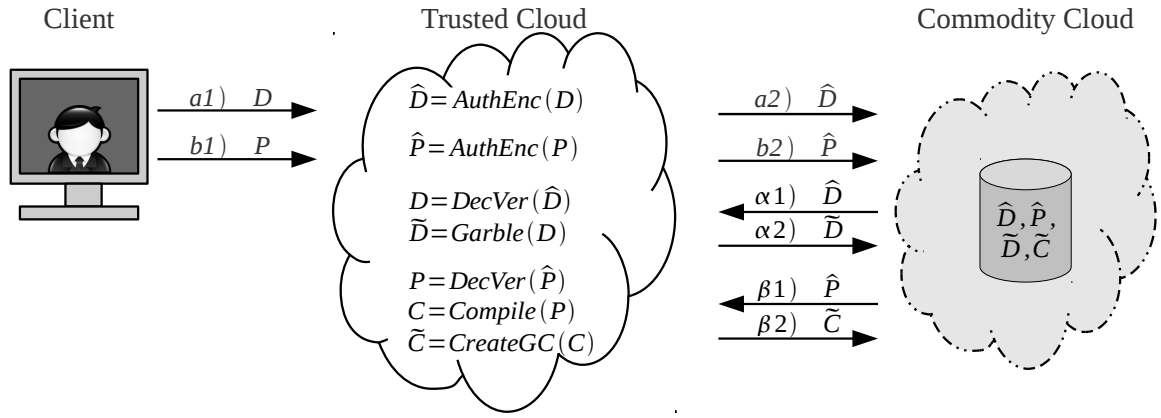

Fig. 2. *Setup Phase*: Client registers the data D and program P that are stored securely in the *Commodity Cloud* (a and b). Updates of P require re-generation of the garbled circuits $\widetilde{C}$ (β), updates of D additionally require re-generation of the garbled data $\widetilde{D}$ (α).

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

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

α) *Garble Data.* Whenever data is changed, the garbled data $\widetilde{D}$ needs to be re-generated. For this, the Trusted Cloud requests the securely stored data $\widehat{D}$ from the Commodity Cloud (α1), 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 (α2).

β) *Garble Program.* Whenever data or the program is changed or all pre-computed garbled circuits have been consumed by the Query Phase, a new garbled circuit $\widetilde{C}$ needs to be generated. For this, the Trusted Cloud requests the securely stored program $\widehat{P}$ from the Commodity Cloud (β1), recovers the data $P = \text{DecVer}(\widehat{P})$ (cf. §3.1), compiles the program into a boolean circuit $C = \text{Compile}(P)$ (cf. §3.3), generates a new garbled circuit $\widetilde{C} = \text{Garble}(C)$ (cf. §3.2) and stores this back into the Commodity Cloud (β2).

5.2 Query Phase

The query phase depicted in Fig. 3 consists of the following use-case.

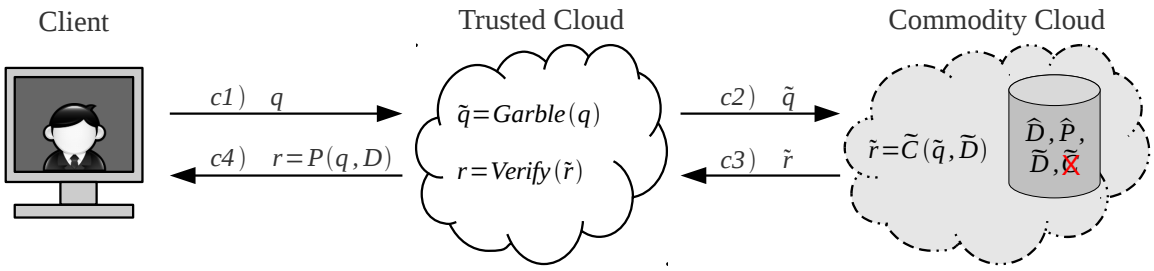

Fig. 3. *Query Phase*: The *Client* sends a query q to the Trusted Cloud to be computed by the Commodity Cloud under encryption (c). The used garbled circuit $\widetilde{C}$ is deleted afterwards.

Process Query. Whenever the Client sends a query q for secure evaluation (c1), the Trusted Cloud converts the query 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 garbled circuit $\widetilde{C}$ (cf. §3.2) in parallel and deleting it afterwards. $\widetilde{r}$ 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.

References

ACCK01. J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *Security and Privacy (S&P'01)*, pages 2–11. IEEE, 2001.

AmEa. Amazon elastic block store (EBS). `http://aws.amazon.com/ebs`.

AmEb. Amazon elastic compute cloud (EC2). `http://aws.amazon.com/ec2`.

APRS01. M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, and E. H. Spafford. Secure outsourcing of scientific computations. *Advances in Computers*, 54:216–272, 2001.

BGN05. D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography Conference (TCC'05)*, volume 3378 of *LNCS*, pages 325–341. Springer, 2005.

BRW04. M. Bellare, P. Rogaway, and D. Wagner. The EAX mode of operation: A two-pass authenticated-encryption scheme optimized for simplicity and efficiency. In *Fast Software Encryption (FSE'04)*, volume 3017 of *LNCS*, pages 389–407. Springer, 2004.

CKV10. K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Advances in Cryptology – CRYPTO'10*, volume 6223 of *LNCS*, pages 483–501. Springer, 2010.

(CS10. Cloud Security Alliance (CSA). Top threats to cloud computing, version 1.0. `http://www.cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf`, March 2010.

DGHV10. M. v. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology – EUROCRYPT'10*, volume 6110 of *LNCS*, pages 24–43. Springer, 2010.

DJ10. M. v. Dijk and A. Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *Hot topics in Security (HotSec'10)*, pages 1–8. USENIX Association, 2010.

Gen09a. C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `http://crypto.stanford.edu/craig`.

Gen09b. C. Gentry. Fully homomorphic encryption using ideal lattices. In *Symposium on Theory of Computing (STOC'09)*, pages 169–178. ACM, 2009.

GGP10. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *Advances in cryptology – CRYPTO'10*, volume 6223 of *LNCS*, pages 465–482. Springer, 2010.

GH10. C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. Cryptology ePrint Archive, Report 2010/520, 2010. `http://eprint.iacr.org/2010/520`.

GKR08. S. Goldwasser, Y. Kalai, and G. Rothblum. One-time programs. In *Advances in Cryptology – CRYPTO'08*, volume 5157 of *LNCS*, pages 39–56. Springer, 2008.

Gro07. Trusted Computing Group. Trusted platform module (TPM) main specification, July 2007. Version 1.2 Revision 103.

HKS+10. W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. In *Computer and Communications Security (CCS'10)*, pages 451–462. ACM, 2010. `http://tastyproject.net`.

HL05. S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In *Theory of Cryptography (TCC'05)*, volume 3378 of *LNCS*, pages 264–282. Springer, 2005.

HS10. A. Herzberg and H. Shulman. Secure guaranteed computation. Cryptology ePrint Archive, Report 2010/449, 2010. `http://eprint.iacr.org/2010/449`.

IBM. IBM. IBM cryptocards. `http://www-03.ibm.com/security/cryptocards/`.

Ili09. A. Iliev. *Hardware-Assisted Secure Computation*. PhD thesis, Dartmouth College, Hanover, NH, USA, 2009. `http://www.cs.dartmouth.edu/~trust/Faerieplay`.

IS10. A. Iliev and S. W. Smith. Small, stupid, and scalable: secure computing with faerieplay. In *Workshop on Scalable Trusted Computing (STC'10)*, pages 41–52. ACM, 2010.

JKSS10a. K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Embedded SFE: Offloading server and network using hardware tokens. In *Financial Cryptography and Data Security (FC'10)*, volume 6052 of *LNCS*, pages 207–221. Springer, January 25-28, 2010.

JKSS10b. K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs. In *Cryptographic Hardware and Embedded Systems (CHES'10)*, volume 6225 of *LNCS*, pages 383–397. Springer, 2010.

KBC97. H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. RFC 2104 (Informational), February 1997.

KL10a. S. Kamara and K. Lauter. Cryptographic cloud storage. In *Financial Cryptography Workshops: Real-Life Cryptographic Protocols and Standardization (RLCPS'10)*, volume 6054 of *LNCS*, pages 136–149. Springer, 2010.

KL10b. S. Kamara and K. Lauter. Cryptographic cloud storage. In *Financial Cryptography and Data Security (FC'10)*, volume 6054 of *LNCS*, pages 136–149. Springer, 2010.

KS08. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

Lim05. ARM Limited. ARM security technology: Building a secure system using trustzone technology, 2005. Whitepaper PRD29-GENC-009492C.

MK10. L. Malka and J. Katz. VMCrypt - modular software architecture for scalable secure computation. Cryptology ePrint Archive, Report 2010/584, 2010. http://eprint.iacr.org/2010/584.

MNPS04. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *USENIX Security Symposium (Security'04)*, pages 287–302. USENIX Association, 2004. http://fairplayproject.net.

NGSN10. S. Narayan, M. Gagné, and R. Safavi-Naini. Privacy preserving EHR system using attribute-based infrastructure. In *Cloud Computing Security Workshop (CCSW'10)*, pages 47–52. ACM, 2010.

NIS01. NIST. U.s. national institute of standards and technology. federal information processing standards (FIPS 197). advanced encryption standard (AES), November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

NIS02. NIST. U.s. national institute of standards and technology. federal information processing standards (FIPS 180-2). announcing the secure hash standard, August 2002. http://csrc.nist.gov/publications/fips/fips180-2/fips-180-2.pdf.

NPS99. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Electronic Commerce (EC'99)*, pages 129–139. ACM, 1999.

Pai99. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

PSS09. A. Paus, A.-R. Sadeghi, and T. Schneider. Practical secure evaluation of semi-private functions. In *Applied Cryptography and Network Security (ACNS'09)*, volume 5536 of *LNCS*, pages 89–106. Springer, 2009. http://www.trust.rub.de/FairplaySPF.

PSSW09. B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *Advances in Cryptology – ASIACRYPT'09*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.

RSA78. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21:120–126, February 1978.

SGR09. N. Santos, K. P. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In *Hot Topics in Cloud Computing (HotCloud'09)*. USENIX Association, 2009.

SHL07. K. Shimizu, H. P. Hofstee, and J. S. Liberty. Cell broadband engine processor vault security architecture. *IBM Journal of Research and Development*, 51:521–528, September 2007.

SMV+10. J. Schiffman, T. Moyer, H. Vijayakumar, T. Jaeger, and P. McDaniel. Seeding clouds with trust anchors. In *Cloud Computing Security Workshop (CCSW'10)*, pages 43–46. ACM, 2010.

SSW10. A.-R. Sadeghi, T. Schneider, and M. Winandy. Token-based cloud computing – secure outsourcing of data and arbitrary computations with lower latency. In *Trust and Trustworthy Computing (TRUST'10) - Workshop on Trust in the Cloud*, volume 6101 of *LNCS*, pages 417–429. Springer, 2010.

SV10. N. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography (PKC'10)*, volume 6056 of *LNCS*, pages 420–443. Springer, 2010.

SW99. S. W. Smith and S. Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31(8):831–860, April 1999.

SZJVD04. R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *USENIX Security Symposium (Security'04)*. USENIX Association, 2004.

TPPG10. J. R. Troncoso-Pastoriza and F. Pérez-González. CryptoDSPs for cloud privacy. In *Workshop on Cloud Information System Engineering (CISE'10)*, 2010.

Yao86. A. C.-C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.