Token-Based Cloud Computing*

Secure Outsourcing of Data and Arbitrary Computations with Lower Latency

Ahmad-Reza Sadeghi, Thomas Schneider, and Marcel Winandy

Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
{ahmad.sadeghi,thomas.schneider,marcel.winandy}@trust.rub.de

Abstract. Secure outsourcing of computation to an untrusted (cloud) service provider is becoming more and more important. Pure cryptographic solutions based on fully homomorphic and verifiable encryption, recently proposed, are promising but suffer from very high latency. Other proposals perform the whole computation on tamper-proof hardware and usually suffer from the the same problem. Trusted computing (TC) is another promising approach that uses trusted software and hardware components on computing platforms to provide useful mechanisms such as attestation allowing the data owner to verify the integrity of the cloud and its computation. However, on the one hand these solutions require trust in hardware (CPU, trusted computing modules) that are under the physical control of the cloud provider, and on the other hand they still have to face the challenge of run-time attestation.

In this paper we focus on applications where the latency of the computation should be minimized, i.e., the time from submitting the query until receiving the outcome of the computation should be as small as possible. To achieve this we show how to combine a trusted hardware token (e.g., a cryptographic coprocessor or provided by the customer) with Secure Function Evaluation (SFE) to compute arbitrary functions on secret (encrypted) data where the computation leaks no information and is verifiable. The token is used in the setup phase only whereas in the time-critical online phase the cloud computes the encrypted function on encrypted data using symmetric encryption primitives only and without any interaction with other entities.

Keywords: Cloud Computing, Hardware Token, Outsourcing.

1 Introduction

Enterprises and other organizations often have to store and operate on a huge amount of data. Cloud computing offers infrastructure and computational services on demand for various customers on shared resources. Services that are offered range from infrastructure services such as Amazon EC2 (computation) [1]

* Supported by EU FP7 projects CACE and UNIQUE, and ECRYPT II.

A. Acquisti, S.W. Smith, and A.-R. Sadeghi (Eds.): TRUST 2010, LNCS 6101, pp. 417–429, 2010.
© Springer-Verlag Berlin Heidelberg 2010

or S3 (storage) [2], over platform services such as Google App Engine [13] or Microsoft's database service SQL Azure [21], to software services such as outsourced customer relationship management applications by Salesforce.com.

While sharing IT infrastructure in cloud computing is cost-efficient 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 fulfill confidentiality and integrity demands. Moreover, since the IT infrastructure is now 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 [8]. In general, important requirements of cloud clients are that their data is processed in a confidential way (*confidentiality*), and that their data and computation was processed in the expected way and has not been tampered with (*integrity and verifiability*).

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 [3], or to securely and verifiably outsource storage [18]. Arbitrary computation on confidential data can be achieved with fully homomorphic encryption [12], in combination with garbled circuits [30] for verifiability [11]. While this cryptographic scheme can fulfill the aforementioned requirements, it is currently not usable in practice due to its low efficiency as we discuss later in §4.2.

Another line of works tries to solve these problems by establishing trusted execution environments where the cloud client can verify the integrity of the software and the configuration of the cloud provider's hardware platform. This requires, however, secure software such as secure hypervisors for policy enforcement and attestation mechanisms for integrity verification. The use of trusted computing-based remote attestation in the cloud scenario was recently discussed in [7]. Trusted Virtual Domains [5,6] are one approach that combines trusted computing, secure hypervisors, and policy enforcement of information flow within and between domains of virtual machines. However, those approaches require trust in a non-negligible amount of hardware (e.g., CPU, Trusted Platform Module (TPM) [29]) which are under the physical control of the cloud provider. According to the specification of the Trusted Computing Group, the TPM is not designed to protect against hardware attacks, but provides a shielded location to protect keys. However, the TPM cannot perform arbitrary secure computations on data. It can protect cryptographic keys and perform only pre-defined cryptographic operations like encryption, decryption, and signature creation. In particular, if data should be encrypted it must be provided in plaintext to the TPM, and if data should be decrypted it will be given in plaintext as output.

Unfortunately, the TPM cannot be instructed to decrypt data internally, perform computations on the data, and encrypt it again before returning the output. A virtualized TPM [4] that is executed in software could be enhanced with additional functionality (see, e.g., [25]). However, such software running on the CPU has access to unencrypted data at some point to compute on it. Hence, if the cloud provider is malicious and uses specifically manipulated hardware, confidentiality and verifiability cannot be guaranteed by using trusted computing.

A hardware token which is tamper-proof against physical attacks but can perform arbitrary computations would enable the cloud client to perform confidential and verifiable computation on the cloud provider's site, given that the client trust the manufacturer of the token that it does not leak any information to the provider. For example, secure coprocessors [27,31] are tamper-proof active programmable devices that are attached to an untrusted computer in order to perform security-critical operations or to allow to establish a trusted channel through untrusted networks and hardware devices to a trusted software program running inside the secure coprocessor. This can be used to protect sensitive computation from insider attacks at the cloud provider [17]. If cloud providers offer such tokens produced by trustworthy third-party manufacturers, or offer interfaces to attach hardware tokens provided by clients to their infrastructure (and by assuming hardware is really tamper-proof), cloud clients could perform their sensitive computations inside those tokens. Data can be stored encrypted outside the token in cloud storage while decryption keys are stored in shielded locations of the trusted tokens.

The token based approach is reasonable because both, cryptographic coprocessors and standardized interfaces (e.g., smartcard readers or PCI extension boards) exist that can be used for such tokens. Of course, for trust reasons, the token vendor should not be the same as the cloud provider. However, the whole security-critical computation takes place in the token. Hence, such computation is not really outsourced to the cloud because the function is computed within the token. Some applications, however, require fast replies to queries which cannot be computed online within the tamper-proof token. For example, queries in personal health records or payroll databases may occur not very frequently, but need to be processed very fast while privacy of the data should be preserved.

In this paper, we focus on cloud application scenarios where private queries to the outsourced data have to be processed and answered with low latency.

Our Contributions and Outline. First we introduce our model for secure verifiable outsourcing of data and *arbitrary* computations thereon in §2.1. Cryptographic primitives and preliminaries are given in §3. In §4 we present architectures to instantiate our model: The first architecture computes the function within a tamper-proof hardware token (§4.1) and the second architecture is based on fully homomorphic encryption (§4.2).

The main technical contribution of our paper is a third architecture (§4.3) that combines the advantages of the previous architectures and overcomes their respective disadvantages. Our solution deploys a resource constrained tamper-proof hardware token in the setup pre-processing phase. Then, in the online

phase only symmetric cryptographic operations are performed in parallel within the cloud without further interaction with the token.

In particular, we adopt the embedded secure function evaluation protocol of [16] to the large-scale cloud-computing scenario.

Finally, in §5 we compare the performance of all three proposed architectures and show that our scheme allows secure verifiable outsourcing of data and *arbitrary* computations thereon with low latency.

2 Model for Secure Outsourcing of Data and Arbitrary Computations

We consider the model shown in Fig. 1 that allows a client $\mathcal{C}$ to verifiably and securely outsource a database D and computations thereon to an untrusted (cloud) service provider $\mathcal{S}$.

A client $\mathcal{C}$ (e.g., a company) wants to securely outsource data D and computation of a function f (represented as a boolean circuit) thereon to an untrusted service provider $\mathcal{S}$ who offers access to (cloud) storage services and to (cloud) computation services. Example applications include outsourcing of medical data, log files or payrolls and computing arbitrary statistics or searches on the outsourced data. In addition, the evaluation of f can depend on a session-specific private query x_i of $\mathcal{C}$ resulting in the response $y_i = f(x_i, D)$. However, $\mathcal{S}$ should be prevented from learning or modifying D or x_i (*confidentiality and integrity*) or to compute f incorrectly (*verifiability*).[1] Any cheating attempts of a malicious $\mathcal{S}$ who tries to deviate from the protocol should be detected by $\mathcal{C}$ with overwhelming probability where $\mathcal{C}$ outputs the special failure symbol $\perp$.[2]

While this scenario can be easily solved for a restricted class of functions (e.g., private search of a keyword x_i using searchable encryption [18]), we consider the general case of arbitrary functions f. Due to the large size of D (e.g., a database) and/or the computational complexity of f, it is not possible to securely outsource D to $\mathcal{S}$ only and let $\mathcal{C}$ compute f after retrieving D from $\mathcal{S}$. Instead, the confidentiality and integrity of the outsourced data D has to be protected while at the same time secure computations on D need to be performed at $\mathcal{S}$ *without* interaction with $\mathcal{C}$.

2.1 Tamper-Proof Hardware Token $\mathcal{T}$

To improve the efficiency of the secure computation, our model additionally allows that $\mathcal{C}$ uses a *tamper-proof hardware token* $\mathcal{T}$, integrated into the infrastructure of $\mathcal{S}$, that is capable of performing computations on behalf of $\mathcal{C}$ within a shielded environment, i.e., must be guaranteed not to leak any information to $\mathcal{S}$.

[1] $\mathcal{S}$ might attempt to cheat to save storage or computing resources or simply manipulate the result.

[2] As detailed in [11] it is necessary that $\mathcal{S}$ does not learn whether $\mathcal{C}$ detected an error or not to avoid that $\mathcal{S}$ can use this single bit of information to construct a decryption or verification oracle.

As $\mathcal{T}$ needs to be built tamper-proof and cost-effective, it will have a restricted amount of memory only. In many cases the available memory within $\mathcal{T}$ will not be sufficient to store D or intermediate values during evaluation of f. If needed, $\mathcal{T}$ might resort to additional (slow) secure external memory (e.g., [10]).

The token $\mathcal{T}$ could be instantiated with a cryptographic coprocessor built by a third-party manufacturer whom $\mathcal{C}$ trusts in a way that $\mathcal{T}$ does not leak any information to $\mathcal{S}$. A possible candidate would be the IBM Cryptographic Coprocessor 4758 or its successor 4764 which is certified under FIPS PUB 140-2 [27,15]. Such cryptographic coprocessors allow to generate secret keys internally and securely transport them to $\mathcal{C}$ or to another token for migration purposes, and authentication to verify that the intended software is executed within the shielded environment. (For details on migrating a state (key) between two trusted environments (cryptographic coprocessors) we refer to [4,25].) As such tokens based on cryptographic coprocessors can be used for multiple users in parallel, their costs amortize for service provider and users.

For extremely security critical applications where $\mathcal{C}$ does not want to trust the manufacturer of cryptographic coprocessors offered by $\mathcal{S}$, $\mathcal{C}$ can choose his own hardware manufacturer to produce the tamper-proof hardware token $\mathcal{T}$ and ship this to $\mathcal{S}$ for integration into his infrastructure. We note that this approach is similar to "server hosting" which assumes outsourcing during long periods; this somewhat contradicts the highly dynamic cloud computing paradigm where service providers can be changed easily.

3 Preliminaries

In this section we introduce the cryptographic building blocks used in the architectures presented afterwards in §4.

3.1 Encryption and Authentication

Confidentiality and authenticity of messages can be guaranteed either symmetrically (using one key) or asymmetrically (using two keys).

The symmetric case can be instantiated with a combination of symmetric encryption (e.g., AES [22]) and a message authentication code (e.g., AES-CMAC [28] or HMAC [20]). These schemes use a respective symmetric key for encryption/authentication and the same key for decryption/verification.

Alternatively, public-key cryptography (e.g., RSA or elliptic curves) allows usage of separate keys for encryption/authentication and other keys for decryption/verification. This could be used for example to construct an outsourced database to which new entries can be appended by multiple parties without using shared symmetric keys (cf. Fig. 1).

Notation. $\widehat{x}$ denotes authenticated and $\overline{x}$ encrypted and authenticated data x.

3.2 Fully Homomorphic Encryption

Fully homomorphic encryption is semantically secure public-key encryption that additionally allows computing an arbitrary function on encrypted data using

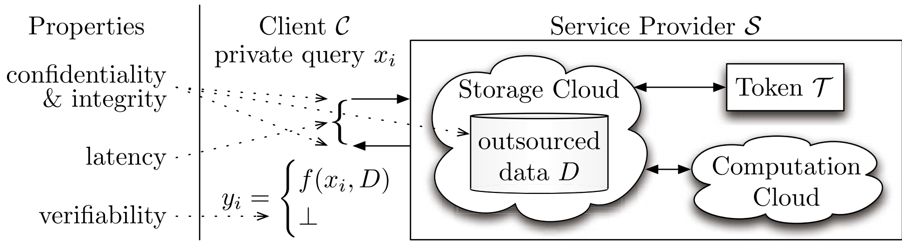

Fig. 1. Model for Secure Outsourcing of Data and Computation

the public-key only, i.e., given a ciphertext $[\![x]\!]$, a function f and the public-key pk, it is possible to compute $[\![y]\!] = \text{EVAL}_{\text{pk}}(f, [\![x]\!]) = [\![f(x)]\!]$. Constructing a homomorphic encryption scheme with polynomial overhead was a longstanding open problem. Recently, there are several proposals starting with [12] and subsequent extensions and improvements of [9,26]. Still, all these schemes employ computationally expensive public-key operations for each gate of the evaluated function and hence are capable of evaluating only very small functions on today's hardware. Recent implementation results of [26] show that even for small parameters where the multiplicative depth of the evaluated circuit is $d = 2.5$, i.e., at most two multiplications, encrypting a single bit takes 3.7s on 2.4GHz Intel Core2 (6600) CPU.

Notation. We write $[\![x]\!]$ for homomorphically encrypted data x.

3.3 Garbled Circuit (GC)

The most efficient method for secure computation of arbitrary functions known today is based on Yao's garbled circuits (GC) [30]. Compared to fully homomorphic encryption, 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 GCs as shown in Fig. 2 is that the *constructor* generates an encrypted version of the function f (represented as boolean circuit), called *garbled circuit* $\widetilde{f}$. For this, he 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 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, evaluator obtains

the corresponding garbled output values $\widetilde{y}$ which allow the constructor to decrypt them into the corresponding plain output $y = f(x)$.

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 $\widetilde{y} = \widetilde{f}(\widetilde{x})$.

Security and Verifiability of GCs. GCs are secure even against malicious evaluator (cf. [14]) and demonstration of valid output keys implicitly proves that the computation was performed correctly (cf. [11]). A fundamental property of GCs is that they can be evaluated only once, i.e., for each evaluation a new GC must be generated.

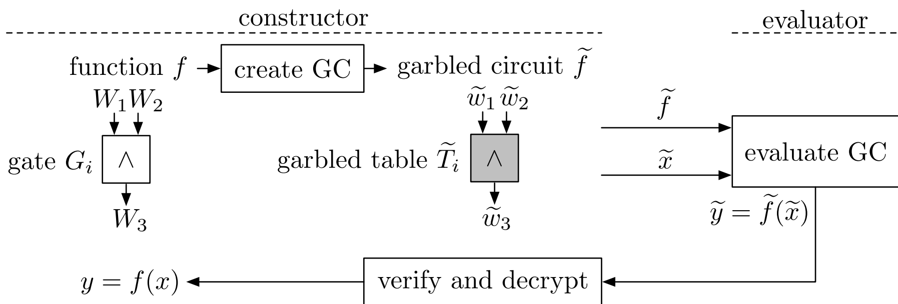

Fig. 2. Overview of Garbled Circuits

Efficient GC construction. The efficient GC construction of [19] 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; its creation requires 4 invocations of a cryptographic hash function (e.g., SHA-256 [23]) and 1 invocation for evaluation. The construction is provably secure in the random-oracle model.

Efficient creation of GCs in hardware. As shown in [16], GCs can be generated within a low-cost tamper-proof hardware token. The token requires only a constant amount of memory (independent of the size of the evaluated function) and performs only symmetric cryptographic operations (SHA-256 and AES). Generation of the GC for the aforementioned AES functionality took 84ms on a 66MHz FPGA neglecting the delay for communicating with the token [16].

Efficient evaluation of GCs in software. The implementation results of [24] show that GCs can be evaluated 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 3.0GHz and 4GB RAM [24].

4 Architectures for Secure Outsourcing of Data and Arbitrary Computation

In this section we present several architectures for our model of §2.

4.1 Token Computes

A first approach, also used in [17], is to let the token $\mathcal{T}$ compute f as shown in Fig. 3. For this, $\mathcal{C}$ and $\mathcal{T}$ share symmetric keys for encryption and verification. The encrypted and authenticated database $\overline{D}$ and the authenticated function $\widehat{f}$ is stored within the storage cloud of service provider $\mathcal{S}$. In the online phase, $\mathcal{C}$ sends the encrypted and authenticated query $\overline{x}_i$ to $\mathcal{T}$ and the storage cloud provides $\overline{D}$ and $\widehat{f}$ one-by-one. $\mathcal{T}$ decrypts and verifies these inputs and evaluates $y_i = f(x_i, D)$ using secure external memory. If $\mathcal{T}$ detects any inconsistencies, it continues evaluation substituting the inconsistent value with a random value, and finally sets y_i to the failure symbol $\perp$. Finally, $\mathcal{T}$ sends the authenticated and encrypted response $\overline{y}_i$ back to $\mathcal{C}$ who decrypts, verifies and obtains the output y_i.

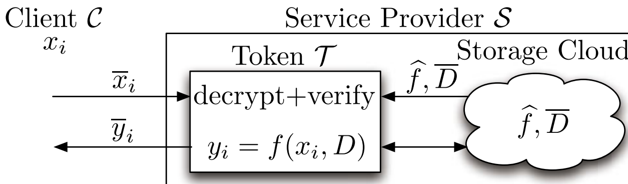

Fig. 3. Architecture: Token Computes [17]

Performance. In this approach, the latency of the online phase, i.e., the time from sending the query x_i to receiving the response y_i, depends on the performance of $\mathcal{T}$ (in particular on the performance of secure external memory) and cannot be improved by using the computation cloud services offered by $\mathcal{S}$.

4.2 Cloud Computes

The approach of [11] shown in Fig. 4 does not require a trusted HW token but combines garbled circuits for verifiability with fully homomorphic encryption for confidentiality of the outsourced data and computations. The main idea is to evaluate the same garbled circuit $\widetilde{f}$ under fully homomorphic encryption and use the resulting homomorphically encrypted garbled output values to verify that the computation was performed correctly:

During setup, $\mathcal{C}$ generates a garbled circuit $\widetilde{f}$ and encrypts its garbled tables with the fully homomorphic encryption scheme resulting in $[\![\widetilde{f}]\!]$ which is sent to $\mathcal{S}$ and stored in the storage cloud. To outsource the database D, the corresponding garbled values $\widetilde{D}$ are encrypted with the fully homomorphic encryption scheme and $[\![\widetilde{D}]\!]$ is stored in $\mathcal{S}$'s storage cloud as well.

In the online phase, $\mathcal{C}$ sends the homomorphically encrypted garbled query $[\![\widetilde{x}_i]\!]$ to $\mathcal{S}$ who evaluates the homomorphically encrypted garbled circuit $[\![\widetilde{f}]\!]$ on $[\![\widetilde{x}_i]\!]$ and $[\![\widetilde{D}]\!]$ using the homomorphic properties of the fully homomorphic encryption scheme. As result, $\mathcal{S}$ obtains $[\![\widetilde{y}_i]\!] = [\![\widetilde{f}(\widetilde{x}_i, \widetilde{D})]\!]$ and sends this back to $\mathcal{C}$. After decryption, $\mathcal{C}$ obtains $\widetilde{y}_i$ and can verify whether the computation was performed correctly. Otherwise, $\mathcal{C}$ outputs the failure symbol $\perp$.

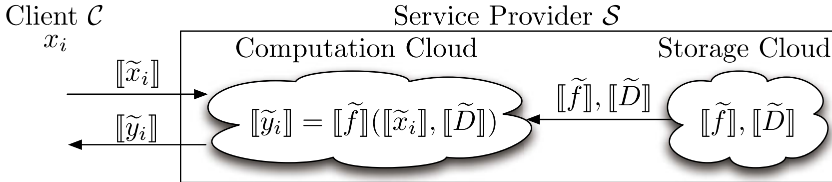

Fig. 4. Architecture: Cloud Computes [11]

Performance. The advantage of this approach is that it does not require any trusted hardware and hence can be computed in parallel in the computation cloud. However, the performance of today's fully homomorphic encryption schemes (in addition to the overhead caused by evaluating a garbled circuit under fully homomorphic encryption) is not sufficient that this approach can be used for practical applications in the near future (see §3.2).

4.3 Token Sets Up and Cloud Computes

Our approach combines a tamper-proof hardware token $\mathcal{T}$ used in the setup phase only with efficient computations performed in parallel in the computation cloud as shown in Fig. 5. The basic idea is that $\mathcal{T}$ generates a garbled circuit during the setup phase and in the time-critical online phase the garbled circuit is evaluated in parallel by the computation cloud.

In detail, our architecture consists of the following three phases:

During *System Initialization*, client $\mathcal{C}$ and the tamper-proof hardware token $\mathcal{T}$ agree on a symmetric (long-term) key k (cf. §2.1). Additionally, $\mathcal{C}$ provides $\mathcal{S}$ with the authenticated function $\widehat{f}$ (represented as boolean circuit) and the authenticated and encrypted data $\overline{D}$ who stores them in the storage cloud.

In the *Setup Phase*, $\mathcal{T}$ generates for protocol invocation i an internal session key k_i derived from the key k and i. Using k_i, $\mathcal{T}$ generates a garbled circuit $\widetilde{f}_i$ from the function $\widehat{f}$ and a corresponding garbled re-encryption $\widetilde{D}_i$ of the database $\overline{D}$ which are stored in the storage cloud: According to the construction of [16], the GC can be generated gate-by-gate using a constant amount of memory only. For each gate of $\widehat{f}$, $\mathcal{S}$ provides $\mathcal{T}$ with the description of the gate. $\mathcal{T}$ uses the session key k_i to derive the gate's garbled input values and the garbled output value and returns the corresponding garbled table to $\mathcal{S}$. In parallel, $\mathcal{T}$ accumulates a hash of the gates requested for so far (e.g., by successively updating $h_i =$

$H(h_{i-1}||G_i)$ where H is a cryptographic hash function and G_i is the description of the i-th gate) which is finally used to verify authenticity of $\widehat{f}$ (see [16] for details). Similarly, $\mathcal{T}$ can convert the authenticated and encrypted database $\overline{D}$ into its garbled equivalent $\widetilde{D}_i$ using constant memory only: For each element $\overline{d}$ in $\overline{D}$, $\mathcal{T}$ decrypts and verifies $\overline{d}$ and uses the session key k_i to derive the corresponding garbled value $\widetilde{d}_i$ of $\widetilde{D}_i$. Finally, $\mathcal{T}$ provides $\mathcal{S}$ with an encrypted and authenticated OK message $\overline{\mathrm{OK}}_i$ that contains the session id and whether the verification of $\widehat{f}$ and all elements in $\overline{D}$ were successful ($\mathrm{OK}_i = \langle i, \top \rangle$) or not ($\mathrm{OK}_i = \langle i, \bot \rangle$).

In the *Online Phase*, $\mathcal{C}$ derives the session key k_i and uses this to create the garbled query $\widetilde{x}_i$ which is sent to $\mathcal{S}$. Now, the computation cloud evaluates the pre-computed garbled circuit $\widetilde{f}_i$ in parallel using the garbled query and the pre-computed garbled data $\widetilde{D}_i$ as inputs. The resulting garbled output $\widetilde{y}_i$ is sent back to $\mathcal{C}$ together with the OK message $\overline{\mathrm{OK}}_i$. Finally, $\mathcal{C}$ verifies that both phases have been performed correctly, i.e., $\overline{\mathrm{OK}}_i$ for the setup phase ($\mathrm{OK}_i = \langle i, \top \rangle$) and valid garbled output keys $\widetilde{y}_i$ for the online phase.

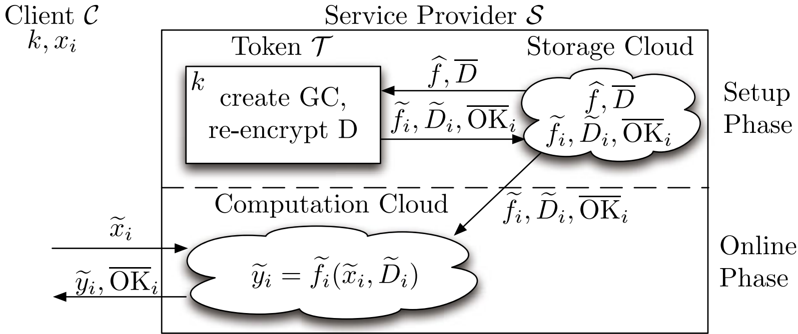

Fig. 5. Our Architecture: Token Sets Up and Cloud Computes

Performance. Our entire architecture is based solely on symmetric cryptographic primitives and hence is very efficient. When $\mathcal{T}$ has access to a hardware accelerator for GC creation (i.e., hardware accelerators for AES and SHA-256), the performance of the setup phase depends mostly on the speed of the interface between the token $\mathcal{T}$ and the storage cloud [16]. The size of $\widetilde{f}_i$ and $\widetilde{D}_i$ is approximately t times larger than the size of $\widehat{f}$ and $\overline{D}$, where t is a symmetric security parameter (e.g., $t = 128$). To evaluate the GC in the online phase, one invocation of SHA-256 is needed for each non-XOR gate while XOR gates are "free" as described in §3.3. GC evaluation can be easily parallelized for many practical functions that usually perform the same operations independently on every entry of the database, e.g., computing statistics or complex search queries.

Extensions. Our architecture can be naturally extended in several ways: To further speed up the setup phase, multiple tokens can be used, that in parallel

Table 1. Complexity Comparison

Architecture	$\mathcal{T}$ Computes (§4.1)	Cloud Computes (§4.2)	$\mathcal{T}$ Sets Up and Cloud Computes (§4.3)
Computation by $\mathcal{C}$	$\mathcal{O}(\|x_i\| + \|y_i\|)$	$\mathcal{O}(\|x_i\| + \|y_i\|)$	$\mathcal{O}(\|x_i\| + \|y_i\|)$
Communication $\mathcal{C} \leftrightarrow \mathcal{S}$	$\mathcal{O}(\|x_i\| + \|y_i\|)$	$\mathcal{O}(\|x_i\| + \|y_i\|)$	$\mathcal{O}(\|x_i\| + \|y_i\|)$
Storage in Cloud	$\mathcal{O}(\|f\| + \|D\|)$	$\mathcal{O}(\|f\| + \|D\|)$	$\mathcal{O}(\|f\| + \|D\|)$
Computation by $\mathcal{T}$	$\mathcal{O}(\|f\|)$ (Online)	none	$\mathcal{O}(\|f\|)$ (Setup)
Computation by Cloud	none	$\mathcal{O}(\|f\|)$ (Online)	$\mathcal{O}(\|f\|)$ (Online)
Online Latency	$\mathcal{T}$ evaluates f	Cloud evaluates $[\![\widetilde{f}]\!]$	Cloud evaluates $\widetilde{f}$

create garbled circuits and re-encrypt the database for multiple or even the same session. The function and the database can be updated dynamically when an additional monotonic revision number is used. Such updates can even be performed by multiple clients $\mathcal{C}_i$ by using public key encryption and signatures as described in §3.1.

5 Conclusion

Summary. We discussed a model and several possible architectures for outsourcing data and arbitrary computations that provide confidentiality, integrity, and verifiability. The first architecture is based on a tamper-proof hardware token, the second on evaluation of a garbled circuit under fully homomorphic encryption, and the third is a combination of both approaches.

Comparison. We conclude the paper with a qualitative performance comparison of the proposed architectures and leave a prototype implementation for their quantitative performance comparison as future work.

As summarized in Table 1, the asymptotic complexity of the presented architectures is the same: the client $\mathcal{C}$ performs work linear in the size of the inputs x_i and the outputs y_i, the storage cloud stores data linear in the size of the evaluated function f and the outsourced data D and the computation performed by the token $\mathcal{T}$ respectively the computation cloud is linear in the size of f. Hence, all three schemes are equally efficient from a complexity-theoretical point of view.

However, the online latency, i.e., the time between $\mathcal{C}$ submitting the encrypted query x_i to the service provider $\mathcal{S}$ until obtaining the result y_i differs substantially in practice.

For the token-based architecture of §4.1, the online latency depends on the performance of the token $\mathcal{T}$ that evaluates f and hence is hard to parallelize and might become a bottleneck in particular when f is large and $\mathcal{T}$ must resort to secure external memory in the storage cloud.

The homomorphic encryption-based architecture of §4.2 does not use a token and hence can exploit the parallelism offered by the computation cloud. However, this architecture is not ready for deployment in practical applications yet, as fully homomorphic encryption schemes are not yet sufficiently fast enough for

evaluating a large functionality such as a garbled circuit under fully homomorphic encryption.

Our proposed architecture of §4.3 achieves low online latency by combining both approaches: $\mathcal{T}$ is used in the setup phase only to generate a garbled circuit and re-encrypt the database. In the online phase, the garbled circuit $\widetilde{f}$ is evaluated in parallel by the computation cloud.

Acknowledgements. We thank the anonymous reviewers of the Workshop on Trust in the Cloud held as part of TRUST 2010 for their helpful comments.

References

1. Amazon Elastic Compute Cloud (EC2), http://aws.amazon.com/ec2
2. Amazon Simple Storage Service (S3), http://aws.amazon.com/s3
3. Atallah, M.J., Pantazopoulos, K.N., Rice, J.R., Spafford, E.H.: Secure outsourcing of scientific computations. Advances in Computers 54, 216–272 (2001)
4. Berger, S., Caceres, R., Goldman, K.A., Perez, R., Sailer, R., Doorn, L.v.: vTPM: Virtualizing the Trusted Platform Module. In: USENIX Security Symposium (USENIX 2006), pp. 305–320. USENIX Association (2006)
5. Bussani, A., Griffin, J.L., Jasen, B., Julisch, K., Karjoth, G., Maruyama, H., Nakamura, M., Perez, R., Schunter, M., Tanner, A., Van Doorn, L., Herreweghen, E.V., Waidner, M., Yoshihama, S.: Trusted Virtual Domains: Secure Foundations for Business and IT Services. Technical Report Research Report RC23792, IBM Research (November 2005)
6. Cabuk, S., Dalton, C.I., Eriksson, K., Kuhlmann, D., Ramasamy, H.G.V., Ramunno, G., Sadeghi, A.-R., Schunter, M., Stüble, C.: Towards automated security policy enforcement in multi-tenant virtual data centers. Journal of Computer Security 18, 89–121 (2010)
7. Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R., Molina, J.: Controlling data in the cloud: outsourcing computation without outsourcing control. In: ACM Workshop on Cloud Computing Security (CCSW 2009), pp. 85–90. ACM, New York (2009)
8. Cloud Security Alliance (CSA). Top threats to cloud computing, version 1.0 (March 2010), http://www.cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf
9. Dijk, M.v., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2009/616 (2009), http://eprint.iacr.org; To appear at EUROCRYPT 2010
10. Garay, J.A., Kolesnikov, V., McLellan, R.: MAC precomputation with applications to secure memory. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 427–442. Springer, Heidelberg (2009)
11. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. Cryptology ePrint Archive, Report 2009/547 (2009), http://eprint.iacr.org
12. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: ACM Symposium on Theory of Computing (STOC 2009), pp. 169–178. ACM, New York (2009)
13. Google App Engine, https://appengine.google.com
14. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-time programs. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 39–56. Springer, Heidelberg (2008)

15. IBM. IBM Cryptocards, http://www-03.ibm.com/security/cryptocards/
16. Järvinen, K., Kolesnikov, V., Sadeghi, A.-R., Schneider, T.: Embedded SFE: Offloading server and network using hardware tokens. In: Financial Cryptography and Data Security (FC 2010), January 25-28. LNCS, Springer, Heidelberg (2010)
17. Jiang, S., Smith, S., Minami, K.: Securing web servers against insider attack. In: Proceedings of the 17th Annual Computer Security Applications Conference, ACSAC (2001)
18. Kamara, S., Lauter, K.: Cryptographic cloud storage. In: Workshop on Real-Life Cryptographic Protocols and Standardization (RLCPS 2010) - co-located with Financial Cryptography, January 2010, LNCS. Springer, Heidelberg (to appear 2010)
19. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
20. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-hashing for message authentication. RFC 2104 (Informational) (February 1997), http://tools.ietf.org/html/rfc2104
21. Microsoft SQL Azure, http://www.microsoft.com/windowsazure
22. 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
23. 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
24. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
25. Sadeghi, A.-R., Stüble, C., Winandy, M.: Property-based TPM virtualization. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 1–16. Springer, Heidelberg (2008)
26. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: PKC 2010. LNCS. Springer, Heidelberg (2010); Cryptology ePrint Archive, Report 2009/571, http://eprint.iacr.org
27. Smith, S.W., Weingart, S.: Building a high-performance, programmable secure coprocessor. Computer Networks 31(8), 831–860 (1999); Special Issue on Computer Network Security
28. Song, J.H., Poovendran, R., Lee, J., Iwata, T.: The AES-CMAC Algorithm. RFC 4493 (Informational) (June 2006), http://tools.ietf.org/html/rfc4493
29. Trusted Computing Group (TCG). TPM main specification. Main specification, Trusted Computing Group (May 2009), http://www.trustedcomputinggroup.org
30. Yao, A.C.: How to generate and exchange secrets. In: IEEE Symposium on Foundations of Computer Science (FOCS 1986), pp. 162–167. IEEE, Los Alamitos (1986)
31. Yee, B.S.: Using Secure Coprocessors. PhD thesis, School of Computer Science, Carnegie Mellon University, CMU-CS-94-149 (May 1994)