

# On the Impossibility of Cryptography Alone for Privacy-Preserving Cloud Computing

Marten van Dijk  
RSA Laboratories  
marten.vandijk@rsa.com

Ari Juels  
RSA Laboratories  
ajuels@rsa.com

## Abstract

*Cloud computing* denotes an architectural shift toward thin clients and conveniently centralized provision of computing resources. Clients' lack of direct resource control in the cloud prompts concern about the potential for data privacy violations, particularly abuse or leakage of sensitive information by service providers. Cryptography is an oft-touted remedy. Among its most powerful primitives is *fully homomorphic encryption* (FHE), dubbed by some the field's "Holy Grail," and recently realized as a fully functional construct with seeming promise for cloud privacy.

We argue that cryptography alone can't enforce the privacy demanded by common cloud computing services, even with such powerful tools as FHE. We formally define a hierarchy of natural classes of private cloud applications, and show that no cryptographic protocol can implement those classes where data is shared among clients. We posit that users of cloud services will also need to rely on other forms of privacy enforcement, such as tamperproof hardware, distributed computing, and complex trust ecosystems.

## 1. INTRODUCTION

*Cloud computing* is a model of information processing, storage, and delivery in which highly centralized physical resources are furnished to remote clients on demand. Rather than purchasing actual physical devices—servers, storage, and networking equipment—clients lease these resources from a cloud provider as a outsourced service that abstracts away physical devices. By sharing infrastructure among tenants, a cloud provider achieves economies of scale and balances workloads, reducing per-unit resource costs and giving clients the ability to ratchet their resource consumption up or down. Cloud computing is flexible and portable in that it can be accessed anytime from anywhere. By using redundant sites and backup storage, cloud providers can also provide greater reliability than local computing systems.

For all the benefits of cloud computing, though, it deprives clients of direct control over the systems that manage their data. Thus arises a central concern of cloud computing: How can clients trust that a cloud provider will protect the privacy of their data, i.e., not leak their data or itself use their data inappropriately?

In this paper, we explore privacy protection in cloud architectures. In particular, we consider the challenge of having a cloud service run applications over client data while: (1) Not being able to learn any information itself and (2) Releasing output values to clients in accordance with an access-control policy. We argue that by itself, cryptography—and by implication, any logical layer information security tool—can't solve this problem in its full generality. Yet this privacy-preserving model is exactly the one ultimately desired for cloud applications involving multiple tenants, such as social networking, document sharing, and so forth. Given recent

excitement over the potential of new, powerful constructs such as *fully homomorphic encryption* (FHE) [11] to support the privacy needs of cloud computing, we believe that our negative message is an important and sobering one.

### 1.1 Cloud Model

We treat a cloud for simplicity as a highly resourced, monolithic entity  $S$ . We denote each entity relying on  $S$ 's resources as a *client* or *tenant*. (In usual parlance, a tenant is a relying entity; a client is a machine. We use the terms interchangeably.) We denote the set of  $n$  tenants of  $S$  by  $\mathcal{C} = \{C_1, C_2, \dots, C_n\}$ .

In our model of cloud computing, clients are thin. They have limited local computation and storage, delegating as much as possible to a cloud provider. And they are not consistently on-line. They may deposit data in the cloud and go offline indefinitely. Consequently, a cloud provider assumes responsibility for processing data in the absence of its owners.

For the purposes of our exploration here, we treat the data of each player  $C_i$  as a static, private value  $x_i$ .  $C_i$  stores  $x_i$  with  $S$ .  $S$  is tasked with executing various applications over  $\{x_i\}$ . The overarching goal of privacy preservation we explore in this paper dictates that in no case should  $S$  learn any portion of any piece  $x_i$  of private data. Applications that operate over the data of multiple clients respect access-control policies, as we explain.

### 1.2 Our Contribution

We explore the challenge of privacy preservation for clients in the cloud by proposing a (nested) hierarchy of three classes of privacy-preserving forms of computing. These classes are meant to characterize natural applications that  $S$  might be called upon to execute in the cloud over clients' sensitive data. We define privacy preservation here to mean that  $S$  itself should learn *no information* from any application execution, while select clients should learn limited output information. (We do not consider *bidirectional privacy* here, i.e., we do not consider the privacy of  $S$ .) The three classes, in order of increasing generality, are:

1. **Private single-client computing:** These applications execute over the data  $x_i$  of a given client  $C_i$ . Their access-control policy stipulates that only  $C_i$  may learn any output. Observe that an access-control policy restricting  $C_i$ 's access to outputs is not meaningful: Since  $x_i$  belongs to  $C_i$ , no information is leaked by revealing any function of  $x_i$  to  $C_i$ .

*Example:* A privacy-preserving tax-preparation program might be implemented via private single-client computing. The data  $x_i$  consists of the financial statements of  $C_i$ —to be hidden from  $S$ . The output of the program is a prepared tax return.

2. **Private multi-client computing:** These applications execute over the data  $\{x_i\}_{i=1}^n$  of multiple clients  $\{C_i\}_{i=1}^n$ . Since clients may not be mutually trusting (and might collude with  $S$ ), a multi-client application’s access-control policy must stipulate release of information selectively to different clients. Such release may be asymmetric, i.e., for a given  $f$ ,  $C_j$  may be granted permission to learn  $f(x_i)$ , while  $C_i$  cannot learn  $f(x_j)$ .

*Example:* A social networking system might be designed as a private multi-client system. Here,  $x_i$  is the personal profile of client  $C_i$ .  $C_i$  additionally specifies which friends are entitled to what portions / functions of her data, i.e., gives an access-control policy.

3. **Stateful private multi-client computing:** These are private multi-client applications in which the access-control policy on a client’s data is stateful, in the sense that it depends on the history of application execution by  $S$ .

*Example:* A healthcare-research system might be implemented via stateful private multi-client computing in which a client is either a patient or a research facility. A patient  $C_i$  furnishes healthcare record  $x_i$ . A research facility  $C_i$  is permitted to learn certain aggregate statistics over the full set of healthcare records in the system. The access-control policy is stateful in the following sense, though: The aggregate information a research facility receives from the system should never be sufficient to reveal individually identifying data. (The system might enforce a standard privacy metric such as  $k$ -anonymity [27].)

Single-client private computing is realizable via FHE, as we explain below.

Private multi-client computing is an important class to consider because it provides natural cloud functionality that is fairly limited, but, as we prove, still not realizable by *any* cryptographic protocol. We prove that private multi-client computing implies general *program obfuscation*, which is provably unachievable in software alone [4]. (Special cases are realizable; our impossibility result applies to the class as a whole.) Thus private multi-client applications require *trustworthy computation* of some type. (Exactly how general such trustworthy computation needs to be is an open problem.) By trustworthy computation, we mean integrity-protected execution history and integrity-protected application of history to access control—i.e., functionality equivalent to a fully trusted party.

Stateful private multi-client applications are an important class to study, as they characterize the norm in the cloud. They include social networks (e.g., Facebook), shared applications (e.g., Google Apps), customer relationship management (e.g., Salesforce.com), etc. They are growing in prevalence. Stateful private multi-client functions clearly imply trustworthy computation in  $S$ .

**Organization:** In section 2, we discuss FHE and survey related work. We explore our three privacy-preserving application classes in detail in section 3, and prove that the class of private multi-client programs cannot be constructed. We conclude in section 4 with a discussion of practical approaches to cloud privacy.

## 2. RELATED WORK

Privacy is a well recognized sticking point in the cloud. Garfinkel [9] discusses how Google Chrome OS realizes the thin-client / monolithic server model we explore here and the privacy concerns that the resulting data amalgamation and loss of infrastructural control bring to consumers. Enterprises too cite security and privacy as top

challenges in cloud adoption, as shown in surveys, e.g., [1], and generally cautious industry adoption [26].

Researchers tend to advocate a consistent set of approaches to privacy enforcement in the cloud. Chow et al. [7] classify these approaches in three major categories: (1) “Information-centric” security, in which data objects are tagged with access-control policies—essentially the mode of operation envisioned in the multi-client classes of the private computing hierarchy we propose here; (2) Trusted computing; and (3) Privacy-preserving cryptographic protocols, which are, of course, the main focus of our work in this paper.

There are a number of privacy-preserving cryptographic protocols appropriate for specific cloud applications. Among these is Private Information Retrieval (PIR) [6], which allows a client to query a database without  $S$  learning which queries the client has submitted. Another example is searchable encryption; see, e.g., [25] for early work and [24] for more recent results. Searchable encryption allows the owner of a set of documents to authorize another party to conduct searches on a pre-specified set of keywords, without revealing any additional information. These are special cases of private multi-player applications.

Proposed as a research challenge in 1978 [23], and long considered the “Holy Grail” of cryptography [21], Fully Homomorphic Encryption was first realized by Gentry in 2009 [11]. FHE enables computation over encrypted data. In a cloud environment, a client can store encrypted data on a server. The server can compute over this data without decrypting, and can send a ciphertext result to the client for decryption. Thus the server computes “under the covers” in a fully privacy-preserving way, never learning the client’s data. While not yet efficient enough for practice, FHE in theory provides general privacy protection for a client-server relationship.

FHE provides a general solution for secure two-party computation, also called secure function evaluation (SFE) [30]. In this sense FHE is a special case of *secure multiparty computation* (SMC), first proposed in [13], and subsequently explored in an extensive literature.<sup>1</sup> SMC allows a set of (multiple) players to compute an arbitrary (bounded complexity) function over private inputs. It realizes, as an interactive protocol, the ideal functionality provided by a trusted party (or piece of hardware). In its general form, however, SMC requires players to be online, and thus isn’t suitable for thin-client cloud architectures.

## 3. CLOUD-APPLICATION CLASS HIERARCHY

### 3.1 Private Single-Client Computing

In the private single-client scenario, client  $C$  asks the cloud  $S$  to evaluate a function  $f$  over  $C$ ’s private input  $x$ .  $S$  should learn no information from the computation, so it is necessary that  $x$ ,  $f(x)$ , and any intermediate values in the computation of  $f(x)$  remain encrypted under  $C$ ’s public key  $p$ .<sup>2</sup>

More generally,  $C$ ’s private input  $x$  can be a composite  $x = (x_1, \dots, x_n)$  of different values  $\{x_i\}_{i=1}^n$  supplied respectively by other clients  $\{C_i\}_{i=1}^n$ . Each  $x_i$  is encrypted under  $C$ ’s public key  $p$  to yield corresponding ciphertext  $c_i$ . Figure 1 depicts this more general scenario. The cloud  $S$  evaluates the resulting ciphertexts  $c_i$

<sup>1</sup>In [12, pp. 22-24], Gentry explains the full interconnection between FHE, SFE with Yao’s garbled circuit, and SMC.

<sup>2</sup>In single-client applications where only the client encrypts her data, symmetric-key encryption suffices. If other entities contribute data, then public key encryption is necessary; these entities need the client’s public key to encrypt their contributed private data.

via evaluation algorithm  $\text{Eval}_{\mathcal{E}}$ . The final result is a ciphertext  $c$ , an encryption of function value  $f(x_S, x_1, \dots, x_n)$  under  $p$ . Here  $f(x_S, x_1, \dots, x_n) = f(x_S, x)$  is  $f$  evaluated in  $C$ 's private value  $x$  together with a value  $x_S$  supplied by  $S$ .<sup>3</sup> The subscript  $\mathcal{E}$  indicates that the evaluation algorithm  $\text{Eval}_{\mathcal{E}}$  is associated with the encryption scheme  $\mathcal{E}$ , which consists of a key-generating, an encryption, and a decryption algorithm.

Note that the  $\{x_i\}$  are private with respect to  $S$ , an important issue when we examine multi-client scenarios.

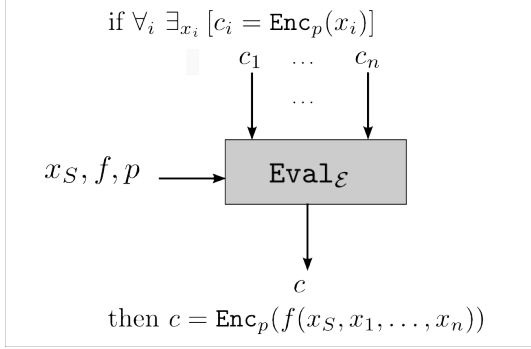


Figure 1: Private Single-Client Computing

It is possible to construct a semantically secure (against chosen plaintext attacks) encryption scheme  $\mathcal{E}$  together with an evaluation algorithm  $\text{Eval}_{\mathcal{E}}$  that satisfies the property depicted in Figure 1. Gentry [11] constructed the first fully homomorphic encryption (FHE) scheme, which solves this problem. Non-homomorphic or partially homomorphic, i.e., ordinary encryption of data does not allow someone without knowledge of the secret decryption key to manipulate the underlying data in a general way. In an FHE scheme, any  $f$  realizable as a (polynomial-size) circuit can be executed without leaking information about inputs, intermediate values, or outputs.

FHE can be used by a single client to outsource private computation to the cloud.<sup>4</sup> But the range of cloud operations enabled by FHE is restricted to an encryption domain defined by the public key  $p$  of a single client. For more general cloud applications, we need to define a more general class.

### 3.2 Private Multi-Client Computing

The objective in a multi-client setting is to compute across data supplied by multiple clients, but also to reveal output values to multiple clients in a privacy-preserving way. To achieve this goal, we need a new primitive that has functionality beyond FHE. In particular, there are two new requirements:

1. **Access-controlled ciphertexts:** Because computation takes place across multiple clients, it's important that a client  $C_i$  be able to stipulate what functions may be computed on its private input  $x_i$ . If arbitrary computation is permitted, then  $x_i$  itself may be revealed to all other clients (and a colluding  $S$ ). We refer to this privacy requirement as *functional privacy*.

<sup>3</sup>Function  $f$  may discard  $x_S$ . Value  $x_S$  can also be encrypted input if the cloud also plays the role of another entity who contributes private data.

<sup>4</sup>E.g., query a database in the cloud privately. Here the privacy of the query is protected into the extend that only its corresponding database access pattern leaks, i.e., the range of items among which the query searches inevitably leaks.

2. **Re-encryption:** Privacy-protected transformation of a ciphertext under a key  $p'$  to a key  $p$  is required to enforce functional privacy. If the encryption keys  $p'$  and  $p$  are identical, then any client that can decrypt outputs can also decrypt and learn inputs, preventing any kind of access control.

In the private multi-client setting, then,  $S$  evaluates function  $f$  on private inputs  $\{x_i\}$  encrypted under (potentially) different clients' public keys  $p_i$ . We let  $c_i$  denote the ciphertext of  $C_i$ . Functional privacy is enforced by allowing  $C_i$  to tag ciphertext  $c_i$  with access-control policy  $A_i$  that indicates whether  $x_i$  can be used as input to a given function  $f$  with output encrypted under public key  $p$ . We write  $c_i = \text{Enc}_{p_i}(x_i, A_i)$ . We model  $A_i$  as a membership circuit that takes as input triples  $(i, f, p)$ . If  $A_i(i, f, p) = \text{true}$ , then client  $C_i$  allows  $x_i$  to be used as the  $i$ th input to  $f$  if its final result is encrypted under  $p$ . (Note that any client  $C_i$  can be a permitted receiver of output in this model, namely when  $p = p_i$ .)

Figure 2 depicts the new situation. If and only if access-control policies on all ciphertexts  $\{c_i\}$  are met, the evaluation algorithm  $\text{Eval}_{\mathcal{E}}$  returns a ciphertext  $c = \text{Enc}_p(f(x_S, x_1, \dots, x_n))$ .

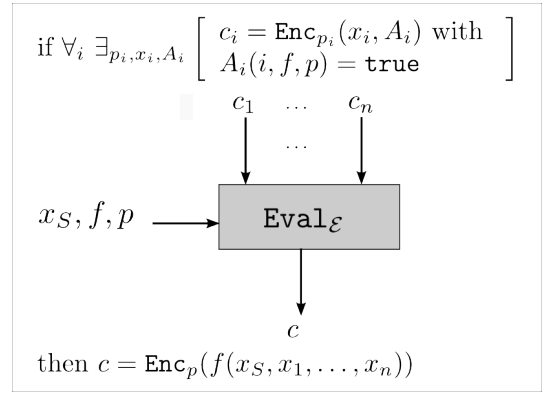


Figure 2: Private Multi-Client Computing

We now prove that private multi-client computing is in general unachievable using cryptography.

**Two-player setting:** For the purposes of our proof, it is simplest to consider a special case of the private multi-client computing class, namely a two-player setting as depicted in Figure 3. There is one sender and one receiver. The function  $f$  takes only two inputs,  $x_S$  and  $x_1$ . The sender uses a simple access-control policy  $C_{(1, f, p)}$ , a membership circuit that outputs  $\text{true}$  only for input  $(1, f, p)$ , i.e., allows only one function  $f$  and one output key  $p$ . The receiver knows the secret key  $s$  corresponding to  $p$  and is able to decrypt the result and retrieve the function output  $f(x_S, x_1)$  for any  $x_S$ . In this sense the receiver has oracle access to the function  $x_S \rightarrow f(x_S, x_1)$ . (Observe that this two-player scheme may be viewed as a multi-player application in which there are two clients and  $S$  learns the decryption key of or colludes with one client.)

**DEFINITION 1.** A two-player private computing scheme has functional privacy over circuits if: For all ppt adversaries  $\mathcal{A}$ , there exists a ppt simulator  $\mathcal{S}$ , and a negligible function  $\alpha$ , such that for all  $(p_1, s_1), (p, s) \leftarrow \text{Gen}(1^\lambda)$ , for all circuits  $f$ , for all ciphertexts  $c_1 \leftarrow \text{Enc}_{p_1}(x_1, C_{(1, f, p)})$  and for all poly-time computable binary predicates  $\pi$ ,

$$\begin{aligned} &Pr[\mathcal{A}(p_1, f, p, c_1, s) = \pi(x_1)] \\ &\leq Pr[\mathcal{S}^{\{x_S \rightarrow f(x_S, x_1)\}}(1^\lambda) = \pi(x_1)] + \alpha(\lambda). \end{aligned}$$

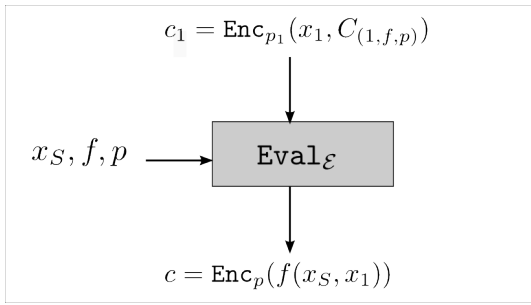


Figure 3: 2-Player Setting

Here,  $\lambda$  is a security parameter; all asymptotics are in  $\lambda$ .  $S$  represents a simulator with oracle access to function  $x_S \rightarrow f(x_S, x_1)$ .

The definition formalizes the intuition that privacy means that an adversary learns no more about  $x_1$  than a simulator can learn using oracle access to  $x_S \rightarrow f(x_S, x_1)$ . That is, an adversary learns about  $x_1$  only what the access control policy  $C_{(1,f,p)}$  dictates.

Note that functional privacy does not imply semantic security. Two-player private computing—and by implication, full multi-client private computing—is not semantically secure against chosen-plaintext attacks. The receiver can distinguish between plaintexts by choosing  $x_1, x'_1, x_S$  and  $f$  such that  $f(x_s, x_1) \neq f(x_s, x'_1)$ .

**Reducing two-player scheme to program obfuscation:** Figure 4 shows a reduction from a two-player private computing scheme with functional privacy to an efficient circuit obfuscator  $\mathcal{O}$  which takes any circuit  $g$  as input and outputs an obfuscated circuit  $\mathcal{O}_g$ . In this reduction, all circuit sizes and running times are polynomial in  $\lambda$  where  $\lambda$  is set as  $|g|$ , the circuit size of  $g$ .

The main idea (explained in detail in the appendix) is to create an execution environment that evaluates a given program  $g$  over an input  $x$  “under the covers,” i.e., in the domain of encryption under key  $p_1$ . This is accomplished by feeding a representation  $\langle g \rangle$  of  $g$  into  $\text{Eval}_E$  (expressed as an evaluation circuit<sup>5</sup>) in  $x_1$  and setting input value  $x = x_S$ . The actual function evaluation  $g(x)$  is performed by a “meta-circuit”  $F$  that takes as input  $x$  and  $\langle g \rangle$ , i.e.,  $F$  is a generic circuit that runs any circuit  $g$  on any input value.<sup>6</sup>  $F$  is the homomorphically computed function here. (Ciphertext  $x_1$  is tagged with access-control policy  $C_{(1,F,p)}$ , which permits application of  $F$ .) The computation result  $g(x)$  is output by a decryption circuit.<sup>7</sup>

The set  $BB = (p_1, F, p, c_1 = \text{Enc}_{p_1}(\langle g \rangle, C_{(1,F,p)}), s)$  of values circled in Figure 4 fully defines the execution environment, i.e., is all the data needed to realize it. The only variable value is the input  $x$ . Thus, running the two-player scheme on  $BB$  gives us a “black box” that takes input  $x$  and outputs  $g(x)$ —an obfuscated circuit  $\mathcal{O}_g$  that executes  $g$ .<sup>8</sup>

By definition 1, it is easy to show that for execution of  $BB$ , we have

$$\Pr[A(\mathcal{O}_g) = \pi(g)] \leq \Pr[S^g(1^{|g|}) = \pi(g)] + \alpha(|g|).$$

<sup>5</sup>We assume that evaluation is compact, i.e., there exists a polynomial  $h$  such that for every value of  $\lambda$ ,  $\text{Eval}_E$  can be expressed as a circuit  $E$  of size  $|E| \leq h(\lambda)$ .

<sup>6</sup> $F$  can be constructed such that its size is polynomial in  $\lambda$ , i.e., polynomial in the size of its inputs.

<sup>7</sup>We assume that decryption  $\text{Dec}$  is compact.

<sup>8</sup>This procedure describes a circuit obfuscator  $\mathcal{O}$ , which is efficient in that  $\mathcal{O}$  itself is a polynomial time algorithm, and its output  $\mathcal{O}_g$  has circuit size polynomial in  $|g|$ .

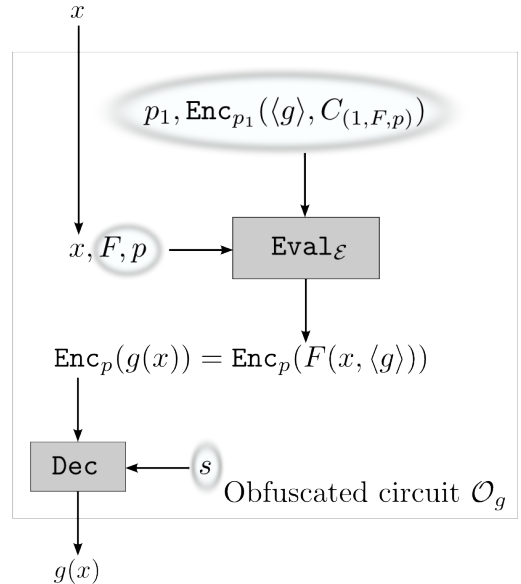


Figure 4: Circuit Obfuscation

That is, execution of  $BB$  obfuscates any (poly-size) program  $g$ , which we know is not achievable [4]. We conclude that general multi-client private computing based solely on cryptographic assumptions is impossible.

**Remarks:** Our proof technique is general: it can be used to prove the impossibility of multi-client computing schemes with functional privacy defined over Turing Machines (TM) or constant depth threshold circuits (TC0) (for which obfuscation impossibility results exist [4]).

While *general* program obfuscation is impossible, the literature does include positive results for *specific* forms of obfuscation, including point functions [20, 29, 14] and certain cryptographic primitives [17, 16].

### 3.3 Private Stateful Multi-Client

Due to lack of space, we omit a formal definition of private stateful multi-client computing. We remark only that in this class, the access control policies for a ciphertext include *the full history of computation of  $S$*  over the data of the client  $C$  that owns the ciphertext. A trustworthy computation environment is clearly necessary to realize this class of applications. As remarked above, this class includes many important applications in the cloud.

A key question, then, regards the relationship between private multi-client computing and stateful private multi-client computing. We proved above that private multi-client computing cannot be realized with cryptography (i.e., software) alone; thus such applications require trusted state / execution of some sort. Are the two application classes equivalent, then, in the sense of having identical trusted execution requirements? This as an important open problem.

## 4. CONCLUSION: HOW TO GET CLOUD PRIVACY?

We have shown the limitations of cryptography alone in meeting the challenges of cloud privacy. So what practical options are there for trustworthy computation? One frequently advocated tool is *trusted computing*, i.e., privacy (and security) enforcement via tamper-resistant hardware. The limitations of that approach too are

legion. They include vulnerability to low-resource hardware attacks [2] and man-in-the-middle attacks during bootstrapping [22]. Even well-functioning hardware cannot guarantee system integrity. Trusted Platform Modules (TPMs) [28], the most prevalent form of trusted hardware, provide only a root of trust: They help ensure the execution of a given software stack, but don't protect against software vulnerabilities. Newer trusted computing technologies such as Intel TXT protect executables, but of course cannot ensure the trustworthiness of applications themselves [18]. Software introspection via, e.g., a trusted hypervisor, can help [10], but also falls far short of comprehensive security assurance.

Additionally, a meaningful trusted computing architecture for the cloud presumes an external entity that can verify the security and privacy posture of a provider. Cloud infrastructure providers are already developing architectures that presume such distributed trust: Trusted hardware and software logging tools generate attestations for consumption by an auditing or compliance-verification system [8].

Cloud architectures based on distributed trust don't align with FHE so much as with SMC. One approach to cloud privacy protection is for clients to distribute their data across a collection of service providers, and rely on the trustworthiness of a majority of them. By executing applications via SMC, providers could then process client data in a privacy-preserving manner (in a stateful multi-client model). In its general form, though, SMC demands impractically intensive computation and communication. We believe that in the short-to-medium term, limited-capability distributed trust models will prevail. While as always helping demarcate trust boundaries, cryptography will also help verify specific security requirements of cloud deployments, e.g., correctly configured storage [3, 5, 19]. It will be one supporting component in a complex ecosystem of trust that depends on interlocking technical, regulatory, and commercial security and privacy enforcement approaches.

## 5. REFERENCES

- [1] IDC IT cloud services survey, 2009.
- [2] R. J. Anderson. *Security Engineering: A guide to building dependable distributed systems*. Wiley, 2008.
- [3] G. Ateniese et al. Provable data possession at untrusted stores. In *ACM CCS*, pages 598–609, 2007.
- [4] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *In Advances in Cryptology - CRYPTO 2001, volume 2139 of Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2001.
- [5] K. D. Bowers, M. van Dijk, A. Juels, A. Oprea, and R. Rivest. How to tell if your cloud files are vulnerable to drive crashes, 2010. In submission. Referenced 2010 at eprint.iacr.org/2010/214.
- [6] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Privacy information retrieval. *J. ACM*, 46(6):965–981, 1998.
- [7] R. Chow et al. Controlling data in the cloud: Outsourcing computation without outsourcing control. In *CCSW*, 2009.
- [8] S. Curry et al. Infrastructure security: Getting to the bottom of compliance in the cloud, March 2010. RSA Security Brief.
- [9] S. L. Garfinkel. A less personal computer. *Technology Review*, May/June 2010.
- [10] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *SOSP*, pages 193–206, 2003.
- [11] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.
- [12] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. <http://crypto.stanford.edu/craig>.
- [13] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – a completeness theorem for protocols with honest majority. *J. of the ACM*, 38(1):691–729, 1991. Preliminary version in FOCS '86.
- [14] S. Goldwasser and Y. Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 553–562. IEEE Computer Society, 2005.
- [15] S. Goldwasser and G. N. Rothblum. On best-possible obfuscation. In *In TCC 2007, LNCS 4392*, pages 194–213, 2007.
- [16] D. Hofheinz, J. Malone-lee, and M. Stam. Obfuscation for cryptographic purposes. In *In TCC 2007, LNCS 4392*, pages 214–232, 2007.
- [17] S. Hohenberger, G. N. Rothblum, a. shelat, and V. Vaikuntanathan. Securely obfuscating re-encryption. In *In TCC 2007, LNCS 4392*, pages 233–252, 2007.
- [18] Intel Corp. Intel trusted execution technology: Software development guide, December 2009.
- [19] A. Juels and B. Kaliski. PORs—proofs of retrievability for large files. In *ACM CCS 2007*, pages 584–597. ACM, 2007.
- [20] B. Lynn, M. Prabhakaran, and A. Sahai. Positive results and techniques for obfuscation. In *In Advances in Cryptology - EUROCRYPT 2004, volume 3027 of Lecture Notes in Computer Science*, pages 20–39, 2004.
- [21] D. Micciancio. A first glimpse of cryptography's Holy Grail. *C. ACM*, 53(3):96, March 2010.
- [22] B. Parno. Bootstrapping trust in a trusted platform. In *3rd USENIX Workshop on Hot Topics in Security (HotSec)*, 2008.
- [23] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–179, 1978.
- [24] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *TCC*, pages 457–473, 2009.
- [25] D. Song, D. Wagner, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE S&P*, pages 44–55, 2000.
- [26] B. Stone and A. Vance. Companies slowly join cloud-computing. *New York Times*, page B1, 19 April 2010.
- [27] L. Sweeney. k-anonymity: A model for protecting privacy. *Int. J. Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.
- [28] Trusted Computing Group. TPM main specification level 2 version 1.2, revision 103, 2007.
- [29] H. Wee. On obfuscating point functions. In *In Proceedings of the 37th ACM Symposium on Theory of Computing (STOC'05)*, pages 523–532. ACM Press, 2005.
- [30] A. C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

## APPENDIX

### A. PROOF OF IMPOSSIBILITY

#### A.1 Encryption Scheme

Following the notation of Gentry [11, 12], we denote by

$$\mathcal{E} = \{\text{Gen, Enc, Dec}\}$$

an encryption scheme that consists of encryption  $\text{Enc}$ , decryption  $\text{Dec}$ , and a key generating algorithm  $\text{Gen}$ .

$\text{Gen}$  is a randomized algorithm that takes a security parameter  $\lambda$  as input, and outputs a public secret key pair  $(p, s)$ :

$$(p, s) \leftarrow \text{Gen}(1^\lambda).$$

The computational complexity is  $\text{poly}(\lambda)$ , polynomial in the size of the input  $1^\lambda$ . Public key  $p$  defines both the plaintext and ciphertext space; both plaintexts and ciphertexts have size  $\text{poly}(\lambda)$ .

$\text{Enc}$  is a randomized algorithm that takes  $p$  and a plaintext  $x$  as input, and outputs a ciphertext  $c$ :

$$c \leftarrow \text{Enc}_p(x).$$

The computational complexity is  $\text{poly}(\lambda)$ , polynomial in the size of plaintext  $x$  and the size of the public key  $p$ .

$\text{Dec}$  is an algorithm that if it takes  $s$  and the ciphertext  $c$  as input, then it outputs the corresponding plaintext  $x$ . This defines *correct decryption*:

$$x = \text{Dec}_s(c).$$

The computational complexity is  $\text{poly}(\lambda)$ , polynomial in the size of ciphertext  $c$  and the size of the secret key  $s$ .

We call  $\mathcal{E}$  compact if it has *compact decryption*: there exists a polynomial  $d$  such that for every value of  $\lambda$ ,  $\mathcal{E}$ 's decryption algorithm can be expressed as a circuit  $D$  of size  $|D| \leq d(\lambda)$ .

## A.2 Evaluation

There exists a *correct evaluation* algorithm  $\text{Eval}_{\mathcal{E}}$  associated to  $\mathcal{E}$ , which satisfies the property of Figure 2 as described in section 3.2 if all plaintexts in Figure 2 are such that encryption is possible (e.g., if the size of input  $|(x_i, A_i)|$  is  $\text{poly}(\lambda)$  and if  $f$  can be represented as a circuit of size  $\text{poly}(\lambda)$ ).

We define  $\mathcal{E}^* = \mathcal{E} \cup \{\text{Eval}_{\mathcal{E}}\}$  and call  $\mathcal{E}^*$  compact if  $\mathcal{E}$  is compact and if it has *compact evaluation*: there exists a polynomial  $e$  such that for every value of  $\lambda$ ,  $\text{Eval}_{\mathcal{E}}$  can be expressed as a circuit  $E$  of size  $|E| \leq e(\lambda)$ .

## A.3 Definition of Security

**Semantic Security:** Suppose that  $\mathcal{E}^*$  has correct decryption and correct evaluation. Consider the following game: Suppose that a challenger runs  $(p, s) \leftarrow \text{Gen}(1^\lambda)$  and gives  $p$  to an adversary  $\mathcal{A}$ . In response,  $\mathcal{A}$  runs  $(p', s') \leftarrow \text{Gen}(1^\lambda)$ , creates the identity circuit  $f$  (i.e.,  $f(x) = x$  for  $x \in \{0, 1\}$ ), and constructs a membership circuit  $A$  that encodes the controlled access policy rule  $(1, f, p')$ . We remind the reader that this means that if  $x$  is encrypted together with  $A$ , then  $x$  can be used as the 1st input of  $f$ , if the resulting evaluation of  $f$  will be encrypted under  $p'$ . The adversary  $\mathcal{A}$  constructs the plaintexts  $m_0 = (0, A)$  and  $m_1 = (1, A)$ , and sends these to the challenger. Now the challenger picks  $b \in \{0, 1\}$ , encrypts  $c = \text{Enc}_p(m_b)$ , and transmits this to  $\mathcal{A}$ .  $\mathcal{A}$  runs the evaluation circuit on  $(f, p')$  and the encrypted input  $c = \text{Enc}_p(m_b) = \text{Enc}_p(b, A)$ . This produces  $\text{Enc}_{p'}(f(b)) = \text{Enc}_{p'}(b)$  since  $f$  is the identity, which  $\mathcal{A}$  decrypts to  $b$  by using the secret key  $s'$ . We conclude that  $\mathcal{A}$  is able to guess  $b$  correctly and that  $\mathcal{E}$  is not semantically secure under chosen plaintext attacks (CPA).

The above is consistent with our understanding: access control policies allow private inputs to leak. In order to model only intended leakage we need to define a weaker form of security that makes more sense in our situation. For example, an adversary should learn no information about the  $x_i$ 's at all if he has no initial knowledge about any secret key  $s'$  that corresponds to one of the

public keys  $p'$  for rules  $(i', f', p')$  in one of the  $A_i$ 's. There should be some form of semantic security against such an adversary.

We will not bother investigating what kind of security would be appropriate: scheme  $\mathcal{E}^*$  is not only an encryption scheme, it also controls the functions that are allowed to compute on private input. This control or functional privacy needs a separate definition. We will show that even if we only require a minimal amount of functional privacy, there cannot exist a compact scheme  $\mathcal{E}^*$  regardless of the kind of semantic security any cryptographer would be interested in.

**Functional Privacy:** Let  $(p_1, A_1, c_1 \leftarrow \text{Enc}_{p_1}(x_1, A_1))$  be a triple with a ciphertext  $c_1$ , a corresponding public key  $p_1$  and access control policy  $A_1$ . Let  $(j, f, p) \in A_1$ . This means that the evaluation algorithm can be used to evaluate  $f$  with  $x_1$  as the  $j$ -th input under encryption with  $p$ . By creating one's own ciphertexts  $\text{Enc}_{p'}(y_i, C_{(i, f, p)})$  (here, and from here onward,  $C_{(i, f, p)}$  is the membership circuit that outputs  $\text{true}$  only for input  $(i, f, p)$ ), it is possible to use the evaluation algorithm to evaluate the function

$$(y_S, y_1, \dots, y_{j-1}, y_{j+1}, \dots, y_n) \\ \rightarrow f(y_S, y_1, \dots, y_{j-1}, x_1, y_{j+1}, \dots, y_n)$$

in arbitrary inputs under encryption with  $p$ .

Let  $X = \{p_i, A_i, c_i \leftarrow \text{Enc}_{p_i}(x_i, A_i)\}_{i=1}^n$  be a larger set of ciphertexts, their corresponding public keys and access control policies. By generalizing the previous analysis, the evaluation algorithm can be used to produce encryptions of outputs of functions that take as input a value  $y_S$  and values  $y_j, j \in J$  and evaluate  $f$  in  $y_S, \{y_j\}_{j \in J}$ , and  $\{x_{i_j}\}_{j \in \{1, \dots, n\} - J}$  if there exists a public key  $p$  with  $(j, f, p) \in A_{i_j}$  for  $j \in J - \{1, \dots, n\}$ . Denote by  $\mathcal{X}$  this set of functions that can be evaluated under encryption.

Let  $K$  be the set of secret keys  $s$  corresponding to the public keys  $p$  for which there exist a triple  $(j, f, p)$  in one of the  $A_i$ 's. A receiver, who knows the secret keys in  $K$ , is able to decrypt the encryptions produced by the evaluation algorithm that takes ciphertexts from  $X$  as input. In this sense the receiver has oracle access to  $\mathcal{X}$ . We want to model the amount of extra information the receiver is able to learn from  $X$  besides the information that can be learned from oracle access to  $\mathcal{X}$ . This will lead to a definition of functional privacy in which we compare an adversary with access to  $X$  and  $K$  to a simulator with oracle access to  $\mathcal{X}$ .

For a simulator algorithm  $\mathcal{S}$ , we denote by  $\mathcal{S}^{\mathcal{X}}(1^\lambda)$  the output of  $\mathcal{S}$  when executed on input  $1^\lambda$  and oracle access to outputs of  $\mathcal{X}$ . Oracle access to outputs of  $\mathcal{X}$  lets the simulator see unencrypted function evaluations; this is the ideal world. For an adversarial algorithm  $\mathcal{A}$  we denote by  $\mathcal{A}(X, K)$  the output of  $\mathcal{A}$  when executed on inputs  $X$  and  $K$ ; this is the view of the receiver in the real world setting.  $\mathcal{A}$  has access to the (public) function descriptions in  $\mathcal{E}^*$ , this allows for the possibility that  $\mathcal{A}$  tampers with ciphertexts, e.g., tries to change a given  $A_i$ .

We say  $\mathcal{E}^*$  is *functional private* if: For all ppt adversaries  $\mathcal{A}$ , there exists a ppt simulator  $\mathcal{S}$ , and a negligible function  $\alpha$ , such that for all sets  $X$  and  $K$ ,

$$\left| \Pr[\mathcal{A}(X, K) = 1] - \Pr[\mathcal{S}^{\mathcal{X}}(1^\lambda) = 1] \right| \leq \alpha(\lambda) \quad (1)$$

( $\lambda$  is the security parameter used for generating public and secret keys in  $X$  and  $K$ ).

We notice that (1) is equivalent to

$$\text{Prob}[\mathcal{A}(X, K) = \pi(\{x_i\}_{i=1}^n)] \\ \leq \Pr[\mathcal{S}^{\mathcal{X}}(1^\lambda) = \pi(\{x_i\}_{i=1}^n)] + \alpha(\lambda)$$

for all predicates  $\pi$ . In words, an adversary cannot learn more infor-

mation about the encrypted  $x_i$ 's than what a simulator can learn using oracle access to  $\mathcal{X}$ , the plaintext results of the evaluation function on inputs  $c_i$ . Here functional privacy means that the adversary cannot learn more about the  $x_i$ 's than a (malicious) collaboration of clients who know the secret decryption keys of the outputs of  $\text{Eval}_{\mathcal{E}}$ . That is, an adversary cannot learn more than what is allowed to be revealed about the  $x_i$ 's.

Our discussion started in the two player setting (one sender, one receiver) with a single triple  $(p_1, A_1, c_1 \leftarrow \text{Enc}_{p_1}(x_1, A_1))$ . Suppose that  $A_1$  is the simplest possible access control policy, a membership circuit  $C_{(1,f,p)}$  with  $f$  a function on two inputs. Then our definition reduces to *2-player functional privacy*: For all ppt adversaries  $\mathcal{A}$ , there exists a ppt simulator  $\mathcal{S}$ , and a negligible function  $\alpha$ , such that for all  $(p_1, f, p, c_1 \leftarrow \text{Enc}_{p_1}(x_1, A_1))$  and  $s$  such that  $(p, s) \leftarrow \text{Gen}(1^\lambda)$ ,

$$\left| \Pr[\mathcal{A}(p_1, f, p, c_1, s) = 1] - \Pr[\mathcal{S}^{\{y_S \rightarrow f(y_S, x_1)\}}(1^\lambda) = 1] \right| \leq \alpha(\lambda). \quad (2)$$

In the next subsection we prove that a compact scheme  $\mathcal{E}^*$  being 2-player functional private is already sufficient to prove the existence of an efficient circuit obfuscator.

#### A.4 Reduction

We will show that the existence of a compact encryption scheme  $\mathcal{E}^*$  that is 2-player functional private reduces to the existence of an efficient circuit obfuscator. Such an obfuscator does not exist [4]. So, even the required minimal privacy cannot be achieved by cryptographic means only.

We will construct an efficient circuit obfuscator  $\mathcal{O}$  which can be asked to produce an obfuscated circuit  $\mathcal{O}_g$  for any circuit  $g$  such that 1)  $\mathcal{O}_g$  describes the same functionality as  $g$ , that is,  $\mathcal{O}_g(x) = g(x)$  for any input  $x$ , 2) there is a polynomial  $h$  such that, for every circuit  $g$ , the size  $|\mathcal{O}_g| \leq h(|g|)$ , and 3)  $\mathcal{O}_g$  is like a "virtual black box" that cannot be used to learn information about (the actual representation or implementation of) circuit  $g$  itself other than its functionality. This *virtual black box property* [4] is very similar to our definition of functional privacy: For all ppt adversaries  $\mathcal{A}$ , there exists a ppt simulator  $\mathcal{S}$ , and a negligible function  $\alpha$ , such that for all circuits  $g$ ,

$$\left| \Pr[\mathcal{A}(\mathcal{O}_g) = 1] - \Pr[\mathcal{S}^g(1^{|g|}) = 1] \right| \leq \alpha(|g|).$$

In words, besides learning the size of circuit  $g$ , an adversary cannot distinguish  $\mathcal{O}_g$  from oracle access to the functionality of  $g$ .

In order to proceed with the reduction we will construct a representation of circuits leading to a meta circuit  $F$  that outputs  $g(x)$  if it receives as input a value  $x$  together with a circuit representation  $\langle g \rangle$ :

$$F(x, \langle g \rangle) = g(x). \quad (3)$$

Let  $g$  be a circuit. Circuits can be designed such that they only use NAND gates. Let  $m$  be the number of NAND gates used in  $g$ ;  $m = |g|$ , the size of  $g$ . Suppose that  $x = (x_1, \dots, x_k)$  is an input to  $g$ . Let  $x_{k+1}, \dots, x_{k+m}$  be the intermediate and final result of  $g$ 's computation after each NAND gate. Without loss of generality, the intermediate results are ordered such that  $x_{k+l+1} = x_i$  NAND  $x_{i'}$  for some  $i$  and  $i'$  both  $\leq k+l$ ; we denote  $lft(k+l) = i$  and  $rght(k+l) = i'$ . For  $1 \leq j \leq k+l$ , define  $z_j^i = 0$  for  $j \neq i$  and  $z_j^i = 1$  for  $j = i$ . Let  $\mathbf{z}^{lft(k+l+1)} = (z_1^i, \dots, z_{k+l}^i)$  and let  $\mathbf{z}^{rght(k+l+1)} = (z_1^{i'}, \dots, z_{k+l}^{i'})$ . Notice that,

$$\text{OR}_j(x_j \text{ AND } z_j^i) = x_i. \quad (4)$$

This shows that there exists a meta circuit  $F$  (independent of  $g$ ) that computes  $g(x)$  as in (3) for inputs  $x$  and

$$\langle g \rangle = (\mathbf{z}^{lft(k+1)}, \mathbf{z}^{rght(k+1)}, \dots, \mathbf{z}^{lft(k+m)}, \mathbf{z}^{rght(k+m)}).$$

Notice that, since  $k \leq m+1$ ,  $|\langle g \rangle| = O(|g|^2)$ . Since circuit  $F$  is composed of  $m$  NAND gates and  $2m$  selection circuits as in (4),  $|F| = O(|g|^2)$ . Notice that  $F$  is independent of  $g$ , it only depends on its size  $|g|$ . The functionality of  $g$  is completely encoded in its representation  $\langle g \rangle$ .

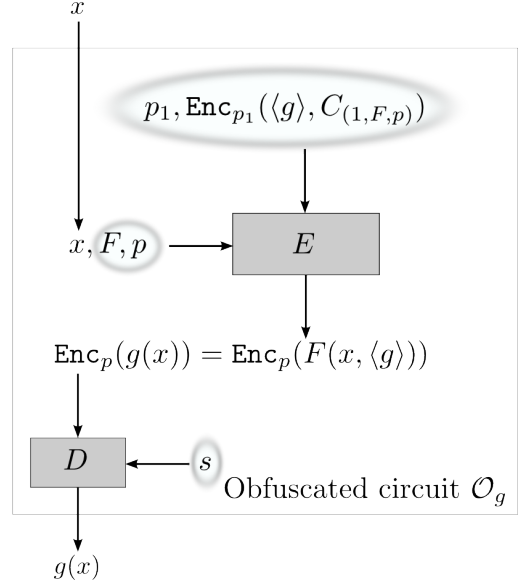


Figure 5: Circuit Obfuscation

Let  $\lambda = O(|g|)$ . We construct an obfuscated circuit  $\mathcal{O}_g$  as follows. We select  $(p_1, s_1) \leftarrow \text{Gen}(1^\lambda)$  and  $(p, s) \leftarrow \text{Gen}(1^\lambda)$ . We construct a membership circuit  $C_{(1,F,p)}$  which has size  $\text{poly}(|g|)$ . We compute a ciphertext  $c_1 \leftarrow \text{Enc}_{p_1}(\langle g \rangle, C_{(1,F,p)})$  (which is possible since the plaintext has size  $\text{poly}(|g|)$ ). Now we proceed as in Figure 5. Since decryption and evaluation are compact, there exist a decryption circuit  $D$  of size  $d(\lambda) = \text{poly}(|g|)$  and an evaluation circuit  $E$  of size  $e(\lambda) = \text{poly}(|g|)$ . The oval shapes represent constant values in circuit  $\mathcal{O}_g$ . As Figure 5 explains, for input  $x$ ,  $\mathcal{O}_g$  outputs  $g(x)$ . Notice that  $\mathcal{O}_g$  has size  $\text{poly}(|g|)$  and that it is uniquely represented by the tuple  $(p_1, F, p, c_1, s)$ , that is,  $\mathcal{O}_g$  can be computed from  $(p_1, F, p, c_1, s)$  and vice versa. By (3), mapping  $y_S \rightarrow F(y_S, \langle g \rangle)$  is identical to function  $g$ . This shows that 2-player functional privacy directly translates into the virtual black box property. We conclude that this describes a circuit obfuscator. Since it is a probabilistic  $\text{poly}(|g|)$  time algorithm, our circuit obfuscator is efficient.

In [4] Barak et al. prove that efficient circuit obfuscators do not exist. This shows that a compact scheme  $\mathcal{E}^*$  which is also 2-player functional private cannot exist. This concludes our impossibility result.

**Impossibility and Possibility:** For completeness: There exist many natural classes of functions that cannot be obfuscated w.r.t to auxiliary input, both when the auxiliary input is dependent on the function being obfuscated and even when the auxiliary input is independent of the function being obfuscated [14]. The more relaxed

requirement that an obfuscated program leaks as little information as any other program with the same functionality (and of similar size) was introduced and studied in [15]. Obfuscation of cryptographic primitives is well studied and has both positive as well as negative results [4, 17, 16]. The possibility of obfuscation of point functions and its impact has been studied in [20, 29, 14]. We notice that each of these results do not affect our impossibility result: if we are interested in a family  $\mathcal{F}$  of circuits for which our technique is still applicable, that is, there exists a "meta circuit" in  $\mathcal{F}$  that can be used to evaluate any circuit, then we are able to construct an obfuscator that can obfuscate any contrived class of circuits, even the one which has been shown in [4] to be impossible to obfuscate.

Our proof provides a general straightforward reduction. We gave only one (and not the most efficient) construction for a meta circuit with circuit representation. There are many other possibilities. To exclude each of these reductions by only requiring  $\mathcal{E}^*$  to evaluate a restricted family of circuits  $\mathcal{F}$  that does not contain a "meta circuit" (such that our reduction is not applicable), seems rather ad hoc. But we do not exclude the possibility that families of circuits  $\mathcal{F}$  with a functionally private compact scheme  $\mathcal{E}^*$  exist. We conjecture that such families are not useful in interesting applications.

For compactness defined by using Turing machines of  $poly(\lambda)$  complexity, our reduction leads to TM obfuscators, which Barak et al. [4] also prove cannot exist. Compactness defined by using constant depth threshold circuits of size  $poly(\lambda)$  (complexity class TC0) also leads to an impossibility result if we assume factoring Blum integers is hard [4].