Improved Delegation of Computation using Fully Homomorphic Encryption[⋆]

Kai-Min Chung[⋆⋆1], Yael Kalai[2], and Salil Vadhan[⋆⋆⋆1]

[1] School of Engineering & Applied Sciences, Harvard University, Cambridge MA, USA, `kmchung@fas.harvard.edu`, `salil@seas.harvard.edu`
[2] Microsoft Research New England, Cambridge MA, USA, `yael@microsoft.com`

Abstract. Following Gennaro, Gentry, and Parno (Cryptology ePrint Archive 2009/547), we use fully homomorphic encryption to design improved schemes for delegating computation. In such schemes, a *delegator* outsources the computation of a function F on many, dynamically chosen inputs x_i to a *worker* in such a way that it is infeasible for the worker to make the delegator accept a result other than $F(x_i)$. The "online stage" of the Gennaro et al. scheme is very efficient: the parties exchange two messages, the delegator runs in time $\mathrm{poly}(\log T)$, and the worker runs in time $\mathrm{poly}(T)$, where T is the time complexity of F. However, the "offline stage" (which depends on the function F but not the inputs to be delegated) is inefficient: the delegator runs in time $\mathrm{poly}(T)$ and generates a public key of length $\mathrm{poly}(T)$ that needs to be accessed by the worker during the online stage.

Our first construction eliminates the large public key from the Gennaro et al. scheme. The delegator still invests $\mathrm{poly}(T)$ time in the offline stage, but does not need to communicate or publish anything. Our second construction reduces the work of the delegator in the offline stage to $\mathrm{poly}(\log T)$ at the price of a 4-message (offline) interaction with a $\mathrm{poly}(T)$-time worker (which need not be the same as the workers used in the online stage). Finally, we describe a "pipelined" implementation of the second construction that avoids the need to re-run the offline construction after errors are detected (assuming errors are not too frequent).

Keywords: verifiable computation, outsourcing computation, worst-case/average-case reductions, computationally sound proofs, universal argument systems

1 Introduction

The problem of delegating computation considers a scenario where one party, the *delegator*, wishes to delegate the computation of a function f to another party, the *worker*. The challenge is that the delegator may not trust the worker, and thus it is desirable to have the worker "prove" that the computation was

[⋆] A full version of this paper can be found on [CKV10]
[⋆⋆] Supported by US-Israel BSF grant 2006060 and NSF grant CNS-0831289.
[⋆⋆⋆] Supported by NSF grant CNS-0831289.

done correctly. Obviously, we want verifying this proof to be easier than doing the computation.

This concept of "outsourcing" computation is relevant in several real world scenarios, as illustrated by the following three examples (taken from [GGP09,GKR08]):

1. **Volunteer computing.** The idea of volunteer computing is for a server to split large computations into small units, send these units to volunteers for processing, and reassemble the results (via a much easier computation). The Berkeley Open Infrastructure for Network Computing (BOINC) [And03,And04] is an example of such a platform. Some famous projects using the BOINC platform are SETI@home, and the Great Internet Mersenne Prime Search [Mer07]. We refer the reader to [GKR08] for more details on these projects.
2. **Cloud computing.** In the setting of cloud computing, businesses buy computing time from a service, rather than purchasing their own computing resources.
3. **Weak mobile devices.** Mobile devices, such as cell-phones, security access-cards, music players, and sensors, are typically very weak computationally, and thus need the help of remote computers to run costly computations.

A natural question about such settings is: *what if the workers are dishonest?* For example, in the volunteer computing setting, an adversarial volunteer may introduce errors into the computation. In the cloud computing example, the cloud (i.e., the business providing the computing services) may have a financial incentive to return incorrect answers, if such answers require less work and are unlikely to be detected by the client. Moreover, in some cases, the applications outsourced to the cloud may be so critical that the delegator wishes to rule out accidental errors during the computation. As for weak mobile devices, the communication channel between the device and the remote computer may be corrupted by an adversary.

In practice, many projects cope with such fraud by redundancy; the same work unit is sent to several workers and the results are compared for consistency. However, this requires the use of several workers and provides little defense against colluding workers.

Instead, we would like the worker to *prove* to the delegator that the computation was performed correctly. Of course, it is essential that the time it takes to verify the proof is significantly smaller than the time needed to actually run the computation. At the same time, the running time of the worker carrying out the proof should also be reasonable — comparable to the time it takes to do the computation. For example, when delegating the computation of a function f that takes time T and has inputs and outputs of length n, we would like the delegator to run in time $\mathrm{poly}(n, \log T)$ and the worker to run in time $\mathrm{poly}(T)$.

1.1 Previous Work

The large body of work on probabilistic proof systems, starting with [Bab85,GMR89], is very relevant to secure delegation. Indeed, after computing the delegated function f on input x and sending the result y, the worker can use various types of proof systems to convince the delegator of the statement "$f(x) = y$".

Interactive Proofs. The IP=PSPACE Theorem [LFKN92,Sha92] yields interactive proofs for any function f computable in polynomial space, with a verifier (delegator) running in polynomial time. However, the complexity of the prover (worker) is also only bounded by polynomial space (and hence exponential time). This theorem was refined and scaled down in [FL93] to give verifier complexity $\mathrm{poly}(n, s)$ and prover complexity $2^{\mathrm{poly}(s)}$ for functions f computable in time T and space s, on inputs of length n. Note that the prover complexity is still superpolynomial in T, even for computations that run in the smallest possible space, namely $s = O(\log T)$. However, the prover complexity was recently improved by Goldwasser et al. [GKR08] to $\mathrm{poly}(T, 2^s)$, which is $\mathrm{poly}(T)$ when $s = O(\log T)$. More generally, Goldwasser et al. [GKR08] give interactive proofs for computations of small *depth* d (i.e. parallel time). For these, they achieve prover complexity $\mathrm{poly}(T)$ and verifier complexity $\mathrm{poly}(n, d, \log T)$. (This implies the result for space-bounded computation because an algorithm that runs in time T and space s can be converted into one that runs in time $\mathrm{poly}(T, 2^s)$ and depth $d = O(s^2)$.) However, if we do not restrict to computations of small space or depth, then we cannot use interactive proofs. Indeed, any language that has an interactive proof with verifier running time (and hence communication) T_V can be decided in space $\mathrm{poly}(n, T_V)$.

PCPs and MIPs. The MIP=NEXP Theorem [BFL91] and its scaled-down version by Babai et al. [BFLS91] yield multiprover interactive proofs and probabilistically checkable proofs for time T computations with a prover running in time $\mathrm{poly}(T)$ and a verifier running in time $\mathrm{poly}(n, \log T)$, exactly as we want. However, using these for delegation require specialized communication models — either 2 noncommunicating provers, or a mechanism for the prover to give the verifier random access to a long PCP (of length $\mathrm{poly}(T)$) that cannot be changed by the prover during the verification.

Interactive Arguments. Instead of changing the communication model, interactive arguments [BCC88] (aka computationally sound proofs [Mic94]) relax the soundness condition to be computational. That is, instead of requiring that no prover strategy whatsoever can convince the verifier of a false statement, we instead require that no computationally feasible prover strategy can convince the verifier of a false statement. In this model, Kilian [Kil92] and Micali [Mic94] gave constant-round protocols with prover complexity $\mathrm{poly}(T, k)$ and verifier complexity $\mathrm{poly}(n, k, \log T)$ (where k is the security parameter), assuming the existence of collision-resistant functions. Under a subexponential hardness assumption, the security parameter can be taken as small as $\mathrm{polylog}(T)$; this also holds for the schemes described below.

Towards Non-interactive Solutions. In this work, we are interested in getting closer to non-interactive solutions (with computational soundness). Ideally, the worker/prover should be able to send a proof to the delegator/verifier in the same message that it sends the result of the computation.

This possibility of efficient non-interactive arguments was suggested by Micali [Mic94], who showed that non-interactive arguments with prover complexity $\mathrm{poly}(T,k)$ and verifier complexity $\mathrm{poly}(n,k,\log T)$ are possible in the Random Oracle Model (the oracle is used to eliminate interaction a la Fiat–Shamir [FS86]). Heuristically, one might hope that by instantiating the random oracle with an appropriate family of hash functions, we could obtain a non-interactive solution to delegating computation: in an offline stage, the verifier/delegator (or a trusted third party) chooses and publishes a random hash function from the family, and in the online stage, the proofs are completely non-interactive (just one message from the prover to the verifier). However, the Random Oracle Heuristic is known to be unsound in general [CGH04] and even in the context of Fiat–Shamir [Bar01,GK03]. Thus, despite extensive effort, the existence of efficient non-interactive arguments remains a significant open problem in complexity and cryptography.

There has been some recent progress in reducing the amount of interaction needed. Using a transformation of Kalai and Raz [KR09], Goldwasser, Kalai, and Rothblum [GKR08] showed how to convert their interactive proofs for small-depth computations into non-interactive arguments in a "public key" model (assuming the existence of single-server private-information retrieval (PIR) schemes): in an offline stage, the verifier/delegator generates a public/secret key pair, publishes the public key and stores the secret key. Then, in the online stage, the prover/worker retrieves the public key and can construct a proof to send along with the result of the computation. However, like the interactive proofs of [GKR08], this solution applies only to small-depth computations, as the verifier's complexity grows linearly with the depth.

Very recently, Gennaro, Gentry, and Parno [GGP09] showed how to delegate arbitrary computations by increasing the verifier's offline complexity and public-key size, and using a fully homomorphic encryption (FHE) scheme (as recently constructed by Gentry [Gen09]). In their construction, the delegator invests $\mathrm{poly}(T,k)$ work in the offline stage to construct a public key of size $\mathrm{poly}(T,k)$ and a secret key of size $\mathrm{poly}(k)$ (for delegating a function f that is computable in time T). In the online stage, the delegator's running time is reduced to $\mathrm{poly}(n,k,\log T)$ for an input of length n, and the worker's complexity is $\mathrm{poly}(T,k)$. Thus, the delegator's large investment in the offline stage can be amortized over many executions of the online stage to delegate the computation of f on many inputs. Their online stage is not completely non-interactive, but consists of two messages. However, in many applications, two messages will be necessary anyway, as the delegator may need to communicate the input x to the worker.

We remark that in the schemes where the delegator has a secret key (namely [GKR08] and [GGP09], as well as two of our constructions below), soundness is only guaranteed as long as the adversarial worker does not learn that the delegator has rejected a proof. Thus, either the accept/reject decision should be kept secret, or the (possibly expensive) offline stage should be re-run after rejection.

1.2 Our Results

In this work, we provide the following protocols that improve over the work of Gennaro et al. [GGP09]:

- Our first protocol eliminates the large public key of the Gennaro et al. scheme. That is, the delegator still performs $\mathrm{poly}(T, k)$ work in the offline stage, but the result of this computation is just a secret key of length $\mathrm{poly}(n, k, \log T)$; there is no need for any interaction with the worker(s) in advance of the online stage (not even to transmit a public key).
- Our second protocol reduces the work of the delegator in the offline stage to $\mathrm{poly}(n, k, \log T)$, at the price of a constant-round interaction with a worker that runs in time $\mathrm{poly}(T, k)$. With this protocol, re-running the offline stage after a rejected proof becomes more reasonable, and thus there is no reason to keep the accept/reject decisions secret.
- Finally, we describe a "pipelined" implementation of our second protocol that avoids the latency of re-running the offline stage, while maintaining soundness even if the accept/reject decisions are revealed. This solution requires both parties to maintain state, and completeness holds provided that faults do not occur too often. Thus, this solution is most suitable for cases where the delegator is using a single worker many times and there are random faults (in communication or computation) that may cause the delegator to reject occasionally.

Like [GGP09], all of our protocols require the use of a fully homomorphic encryption scheme, and have a 2-message online stage. A full comparison of our model and results with previous work is given in Table 1.

Organization. Brief preliminaries on fully homomorphic encryption schemes are presented in Section 2. Then we present a formal definition of our model in Section 3. In Section 4 – 8, we start with a simple scheme Del_1 that achieves rather weak properties, and strengthen it through a series of steps leading to our main delegation schemes Del_4 and Del_5.

Due to space constraints, we skip all the proofs. Please refer to the full version of this paper [CKV10] for details.

2 Preliminaries on Fully Homomorphic Encryption

Inspired by the recent work of Gennaro, Gentry, and Parno [GGP09] on secure delegation, our constructions rely on the use of a fully homomorphic encryption scheme.

Fully Homomorphic Encryption. A public-key encryption scheme $\mathrm{E} = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ is said to be *fully homomorphic* if it is associated with an additional polynomial-time algorithm Eval, that takes as input a public key pk, a ciphertext $\hat{x} = \mathrm{Enc}(x)$ and a circuit C, and outputs, a new ciphertext $c = \mathrm{Eval}_{\mathsf{pk}}(\hat{x}, C)$, such that

			offline		keys		online		
Ref	Assumption	Soundness	# msgs	D complexity	$\|PK\|$	$\|SK\|$	# msgs	D complexity	W complexity
[GKR08]	none	stat	0	0	0	0	$\text{poly}(d, \log T)$	$\text{poly}(n, d, \log T)$	$\text{poly}(T)$
[BFL91,BFLS91]	none	MIP/PCP	0	0	0	0	1	$\text{poly}(n, \log T)$	$\text{poly}(T)$
[Kil92,Mic00]	CRH	comp	0	0	0	0	4	$\text{poly}(k, n, \log T)$	$\text{poly}(k, T)$
[Kil92,Mic00]	RO-Heur	comp	1	$\text{poly}(k)$	$\text{poly}(k)$	0	1	$\text{poly}(k, n, \log T)$	$\text{poly}(k, T)$
[GKR08,KR09]	PIR	comp	1	$\text{poly}(k)$	$\text{poly}(k, d, \log T)$	$\text{poly}(k, d, \log T)$	1	$\text{poly}(k, n, d, \log T)$	$\text{poly}(k, T)$
[GGP09]	FHE	comp	1	$\text{poly}(k, T)$	$\text{poly}(k, T)$	$\text{poly}(k, n)$	2	$\text{poly}(k, n, \log T)$	$\text{poly}(k, T)$
Thm. 1	FHE	comp	0	$\text{poly}(k, T)$	0	$\text{poly}(k, n)$	2	$\text{poly}(k, n, \log T)$	$\text{poly}(k, T)$
Thm. 3	FHE	comp	4	$\text{poly}(k, n, \log T)$	0	$\text{poly}(k, n)$	2	$\text{poly}(k, n, \log T)$	$\text{poly}(k, T)$

Table 1. Results on Delegating Computation. D = delegator/verifier, W = worker/prover, $PK = D$'s public key, $SK = D$'s secret key, k = security parameter. Parameters of computation f being delegated: n = input length, T = time, d = depth/parallel time (we assume $n \leq T \leq 2^d$)

$\mathrm{Dec}_{\mathsf{sk}}(c) = C(x)$, where sk is the secret key corresponding to the public key pk. It is required that the size of $c = \mathrm{Eval}_{\mathsf{pk}}(\mathrm{Enc}_{\mathsf{pk}}(x), C)$ depends polynomially on the security parameter and the length of $C(x)$, but is otherwise independent of the size of the circuit C. We also require that Eval is deterministic, and the scheme has perfect correctness (i.e. it always holds that $\mathrm{Dec}_{\mathsf{sk}}(\mathrm{Enc}_{\mathsf{pk}}(x)) = x$ and that $\mathrm{Dec}_{\mathsf{sk}}(\mathrm{Eval}_{\mathsf{pk}}(\mathrm{Enc}_{\mathsf{pk}}(x), C)) = C(x)$). For security, we simply require that E is semantically secure.

In a recent breakthrough, Gentry [Gen09] proposed a fully homomorphic encryption scheme based on ideal lattices. In his basic scheme, the complexity of the algorithms $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ depends linearly on the *depth* of the circuit C, where d is an upper bound on the depth of the circuit C that are allowed as inputs to Eval. However, under the additional assumption that his scheme is circular secure (i.e., it remains secure even given an encryption of the secret key), the complexity of these algorithms are independent of C. Furthermore, Gentry's construction satisfies the perfect correctness and the Eval of his scheme can be made deterministic. We refer the reader to [Gen09] for details.

An interesting aspect of the [GGP09] construction is how they use the *secrecy* property of fully homomorphic encryption schemes in order to achieve a *soundness* property in their delegation scheme; this phenomenon also recurs several times in our work.

3 The Model

In this section, we formally define a model that captures the delegating computation scenario we are interested in.

Definition 1 (Delegation Scheme). *A* delegation scheme *is an interactive protocol* $\mathsf{Del} = \langle \mathsf{D}, \mathsf{W} \rangle$ *between a delegator* D *and a worker* W *with the following structure:*

1. *The scheme* Del *consists of two stages: an offline/preprocessing stage and an online stage. The offline stage is executed once before the online stage, whereas the online stage can be executed many times.*
2. *In the offline stage, both the delegator* D *and the worker* W *receive a security parameter* k *and a function* $F : \{0,1\}^n \to \{0,1\}^m$*, represented by a Turing machine* M *and a time bound* T *for* M*. At the end of the interaction, the delegator* D *decides whether to accept or reject. If* D *accepts, then* D *outputs a* secret key σ_{D} *and a* public key σ_{W}*. We will denote this by* $(\sigma_{\mathsf{D}}, \sigma_{\mathsf{W}}) = \langle \mathsf{D}, \mathsf{W} \rangle(F, 1^k)$*. We will use the notation* M*,* n*,* m*, and* T *as the Turing machine and parameters associated with* F *throughout the paper, and we will often omit the security parameter from the notation.*
3. *In the online stage, both parties receive* F*,* 1^k*, and an input* $x \in \{0,1\}^n$*, and execute a one round communication protocol. Namely,* D *sends* $q = \mathsf{D}(F, x, \sigma_{\mathsf{D}})$ *to* W*, and then* W *sends* $a = \mathsf{W}(F, x, \sigma_{\mathsf{W}}, q)$ *to* D*. Then the delegator* D *either accepts or rejects. If* D *accepts, then* D *also generates a private output* $y = \mathsf{D}(F, x, \sigma_{\mathsf{D}}, q, a) \in \{0,1\}^m$*, which is supposed to be* $F(x)$*.*

For simplicity, we will omit the function F and the security parameter from the input of the online stage.

We also define the following properties of delegation schemes.

- *A delegation scheme* Del *has an* **efficient** *delegator in the online (resp., offline) stage if the computational complexity of* D *in the online (resp., offline) stage is* $\mathrm{poly}(k, n, m, |M|, \log T)$.
- *A delegation scheme* Del *has an* **efficient** *worker if the computational complexity of* W *is* $\mathrm{poly}(k, |M|, T)$.
- *A delegation scheme* Del *has a* **non-interactive** *offline stage if* D *and* W *do not interact at all during the offline stage, and only* D *does some computation. Note that if* Del *has a non-interactive offline stage, then we can assume w.l.o.g. that* D *always accepts in the offline stage.*

For a delegation scheme to be meaningful, it needs to have completeness and soundness properties. Informally, the completeness property says that the delegator D always learns the desired value $F(x)$, assuming both parties follow the prescribed protocol. The soundness property says that the delegator D mistakenly accepts a wrong value $y \neq F(x)$ from a malicious worker with only negligible probability.

Definition 2 (Completeness). *A delegation scheme* $\mathsf{Del} = \langle \mathsf{D}, \mathsf{W} \rangle$ *has* perfect completeness *if for all parameters* n, m, T, k*, for every function* F *and every* $x \in \{0,1\}^n$*, the following holds with probability* 1*: When* D *and* W *run the offline stage protocol with input* F*, and then run the online stage protocol with input* x*, the delegator* D *accepts in both the offline and the online stage, and outputs* $y = F(x)$ *in the online stage.*

In order to define the soundness, we introduce the following security game.

Definition 3 (Security Game for Delegation Schemes). *Let* $\mathsf{Del} = \langle \mathsf{D}, \mathsf{W} \rangle$ *be a delegation scheme and* $k \in \mathbb{N}$ *be the security parameter. The* security game $\mathcal{G}(k)$ *for* Del *is the following game played by a worker strategy* W^*.

- *The game starts with the offline stage of* Del*, and is followed by many rounds of the online stage.*
- $\mathsf{W}^*(1^k)$ *first chooses the delegation function* F *and then* D *and* W^* *interact in the offline stage of* Del *with input* F.
- *At the beginning of each round of the online stage (indexed by* ℓ*),* W^* *can either terminate the game or choose an input* $x_\ell \in \{0,1\}^n$*. If the game is not terminated,* D *and* W^* *interact in the online stage of* Del *on input* x_ℓ.
- *Whenever the delegator* D *rejects, the game terminates.*

W^* succeeds *in the game* $\mathcal{G}(k)$ *if there exists a round* ℓ *of the online stage such that* D *accepts and outputs a wrong value* $y_\ell \neq F(x_\ell)$*, where* x_ℓ *is the delegated input chosen by* W^*.

Definition 4 (Soundness). *Let $\varepsilon : \mathbb{N} \to [0,1]$ and $t : \mathbb{N} \to \mathbb{N}$ be efficiently computable functions. A delegation scheme $\mathsf{Del} = \langle \mathsf{D}, \mathsf{W} \rangle$ has* **soundness error** *ε for delegation functions with runtime at most t if for every worker strategy W^*, which runs in time $t(k)$ and chooses a delegation function with runtime at most $t(k)$,*

$$\Pr[\mathsf{W}^* \textit{ succeeds in } \mathcal{G}(k)] \le \varepsilon(k),$$

for sufficiently large k, where $\mathcal{G}(k)$ is the corresponding security game for Del. We say that Del is **sound** *if Del has soundness error $1/k^c$ for delegation functions with runtime at most k^c for every constant c.*

Note that the above definition does not guarantee soundness for delegating functions of complexity superpolynomial in k. However, we have soundness for functions of complexity that is an arbitrarily large polynomial in k, whereas an efficient delegator would run in time that is a fixed polynomial in k; so the delegation is still quite useful. This quantitative relationship stems from the standard asymptotic formulation of security as being with respect to polynomial-time adversaries. If we use a fully homomorphic encryption scheme that is secure against adversaries running time subexponential in k, then we would obtain soundness for delegating functions of subexponential complexity (while the delegator still runs in fixed polynomial time).

In terms of concrete security, the security parameter k should be chosen by the delegator so that breaking the encryption scheme requires an infeasible amount R of resources for the worker, and thus the delegator should only be delegating functions that require significantly less resources than R.

Note that in the security game $\mathcal{G}$, the delegator D rejects and terminates the game, whenever he catches the worker cheating. Thus, the soundness is only guaranteed until the worker cheats. In other words, once the worker cheats, the delegator D can catch this mistake with overwhelming probability, but the delegation scheme no longer guarantees soundness for the next delegated inputs. Therefore, D should restart the delegation scheme from the offline stage to ensure the soundness of future delegated inputs.

The model of [GGP09] takes a different approach. Rather than halting the game after a rejection, they instead consider a game where the delegator's accept/reject decisions are kept secret from the worker. Our protocols also satisfy their definition; indeed, the two definitions are equivalent for schemes where the delegator has no state (other than the secret key).

4 $\mathsf{Del}_1 = \langle \mathsf{D}_1, \mathsf{W}_1 \rangle$: One-time, Random-Input Delegation Scheme

In this section, we present our first warmup delegation scheme $\mathsf{Del}_1 = \langle \mathsf{D}_1, \mathsf{W}_1 \rangle$ for the following one-time and random-input scenario.

Scenario: Suppose the delegator D knows that at some point in the future, he will receive a *random* input $x \in \{0,1\}^n$ drawn from a certain (samplable) distribution $\mathcal{D}$ and he will want to learn the value $F(x)$ quickly. Thus, D decides to delegate the computation of $F(x)$ to an untrusted worker W (who does *not* know the random x), and D wants to be able to verify the answer from W very efficiently.

The idea is simple and similar to the idea underlying reCAPTCHAs [vAMM+08]: In the offline stage, the delegator D_1 samples a random input $r \leftarrow \mathcal{D}$ and pre-computes $F(r)$. In the online stage, D_1 sends both x and r to W_1 in a random order, and asks W_1 to compute both $F(x)$ and $F(r)$. Upon receiving the answers from W_1, the delegator D_1 checks the correctness of the returned value $F(r)$; if it is correct then he accepts the returned $F(x)$, and otherwise he reject. Thus, a malicious worker W^* can convince D_1 with a wrong answer iff W^* can guess which input is the delegator's real input. Since x and r are independent and identically distributed, no malicious prover can guess the real input x and cheat successfully with probability greater than 1/2. A formal description of our random-input delegation scheme $\mathsf{Del}_1 = \langle \mathsf{D}_1, \mathsf{W}_1 \rangle$ can be found in Figure 1. A formal analysis of Del_1 can be found in the full version of this paper [CKV10].

- **Inputs.** Security parameter 1^k and function $F : \{0,1\}^n \rightarrow \{0,1\}^m$, specified by a Turing machine M, and a time bound T.
- **Offline Stage.** Both D_1 and W_1 receive input $(F, \mathcal{D})$
  1. D_1 samples a random input $r \leftarrow \mathcal{D}$, computes $w = F(r)$, and stores the pair (r, w) as his secret state.
- **Online Stage.** D_1 receives $x \in \{0,1\}^n$ (where x is expected to distribute according to $\mathcal{D}$), and W_1 does not receive any input.
  1. D_1 sets $r_0 = r$ and $r_1 = x$. It then samples a random bit $b \in_R \{0,1\}$, and sends $(z_0, z_1) = (r_b, r_{1-b})$ to W_1.
  2. W_1 computes and sends $(y_0, y_1) = (F(z_0), F(z_1))$ to D_1.
  3. D_1 accepts and outputs the answer y_{1-b} iff $w = y_b$.

Fig. 1. Delegation Scheme $\mathsf{Del}_1 = \langle \mathsf{D}_1, \mathsf{W}_1 \rangle$

5 $\mathsf{Del}_2 = \langle \mathsf{D}_2, \mathsf{W}_2 \rangle$: One-time, Arbitrary-input Delegation Scheme

Recall that in the random-input delegation scheme $\mathsf{Del}_1 = \langle \mathsf{D}_1, \mathsf{W}_1 \rangle$, it was essential that the input x is hidden from the worker in the online stage to guarantee the soundness. If the worker knew x, he could discriminate between r and x, and cheat by answering correctly on r and incorrectly on x.

We eliminate this strong limitation by using a fully-homomorphic encryption scheme to "*computationally randomize*" the input: Instead of sending x in the clear, the delegator will encrypt the input x to obtain $\hat{x} \overset{\text{def}}{=} \mathrm{Enc}_{\mathsf{pk}}(x)$. Then the delegator will ask the worker to compute the *deterministic* homomorphic evaluation $\hat{F}(\hat{x}) \overset{\text{def}}{=} \mathrm{Eval}_{\mathsf{pk}}(\hat{x}, F)$ of F on the encrypted value $\hat{x}$, from which he can decrypt to obtain the desired answer $F(x)$.[3] Notice that even if x is fixed,

[3] We note that in order to compute $\mathrm{Eval}_{\mathsf{pk}}(\hat{x}, F)$, the Turing machine F needs to be turned into a circuit. This can be done via a standard simulation of Turing machines by circuits.

the distribution of $\hat{x} = \mathrm{Enc}_{\mathsf{pk}}(x)$ is computationally indistinguishable from the distribution of $\mathrm{Enc}_{\mathsf{pk}}(\bar{0})$, which is efficiently samplable and independent of x. Thus, the delegator can precompute an encryption $\hat{r} = \mathrm{Enc}_{\mathsf{pk}}(\bar{0})$ together with $\hat{F}(\hat{r}) = \mathrm{Eval}_{\mathsf{pk}}(\hat{r}, F)$, and use the pair $(\hat{r}, \hat{F}(\hat{r}))$ to verify the worker's answer as before.

We emphasize that the delegator checks the correctness of the *ciphertext* $\hat{F}(\hat{r}) = \mathrm{Eval}_{\mathsf{pk}}(\hat{r}, F)$ obtained from homomorphic evaluation of F on $\hat{r} = \mathrm{Enc}_{\mathsf{pk}}(\bar{0})$, as opposed to the *value* $f(\bar{0})$ underlying the ciphertext. Indeed, it is insufficient for the delegator to only check the correctness of the value $f(\bar{0})$, since an adversarial worker W^*, who knows the input x, could easily cheat by applying $\hat{G}(\hat{r}) = \mathrm{Eval}_{\mathsf{pk}}(\hat{r}, G)$, where $G(y)$ equals $F(y)$ iff $y \neq x$.

The above computational randomization technique extends the random-input delegation scheme Del_1 to a (standard) delegation scheme Del_2 with one-time soundness error $1/2$. We formally describe the delegation scheme $\mathsf{Del}_2 = \langle \mathsf{D}_2, \mathsf{W}_2 \rangle$ in Figure 2 below.

- **Inputs.** Security parameter 1^k and function $F : \{0,1\}^n \to \{0,1\}^m$, specified by a Turing machine M, and a time bound T.
- **Offline Stage.** Both D_2 and W_2 receive as input a function F.
  1. D_2 generates a pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{KeyGen}(1^k)$, computes an encryption $\hat{r} = \mathrm{Enc}_{\mathsf{pk}}(\bar{0})$ and the (deterministic) homomorphic evaluation $\hat{w} = \hat{F}(\hat{r}) = \mathrm{Eval}_{\mathsf{pk}}(\hat{r}, F)$, and stores the tuple $(\mathsf{pk}, \mathsf{sk}, \hat{r}, \hat{w})$ as his secret key.
- **Online Stage.** Both D_2 and W_2 receive an input $x \in \{0,1\}^n$.
  1. D_2 computes an encryption $\hat{x} = \mathrm{Enc}_{\mathsf{pk}}(x)$, sets $\hat{r}_0 = \hat{r}$ and $\hat{r}_1 = \hat{x}$, samples a random bit $b \in_R \{0,1\}$, and sends the public key pk and $(\hat{z}_0, \hat{z}_1) = (\hat{r}_b, \hat{r}_{1-b})$ to W_2.
  2. W_2 computes $\hat{y}_i = \hat{F}(\hat{z}_i) = \mathrm{Eval}_{\mathsf{pk}}(\hat{z}_i, F)$ for $i \in \{0,1\}$, and sends $(\hat{y}_0, \hat{y}_1) = (\hat{F}(\hat{z}_0), \hat{F}(\hat{z}_1))$ to D_2.
  3. D_2 accepts and outputs the answer $\mathrm{Dec}_{\mathsf{sk}}(\hat{y}_{1-b})$ iff $\hat{w} = \hat{y}_b$.

Fig. 2. Delegation Scheme $\mathsf{Del}_2 = \langle \mathsf{D}_2, \mathsf{W}_2 \rangle$

It is straightforward to check that if the fully homomorphic encryption scheme has perfect correctness, then Del_2 has the perfect completeness. To argue the soundness of the scheme, we first give the definition of one-time soundness.

Definition 5 (One-time Soundness for Delegation Schemes). *Let* $\mathsf{Del} = \langle \mathsf{D}, \mathsf{W} \rangle$ *be a delegation scheme and* $k \in \mathbb{N}$ *be a security parameter. The* **one-time security game** $\mathcal{G}(k)$ *for* Del *is the same as security game for* Del *defined in Definition 3 excepts that it only allows one round in the online stage. We say that* Del *has* **one-time soundness error** ε *if for every* PPT *worker strategy* W^* *who chooses a polynomial time delegation function, and all sufficiently large* k, $\Pr[\mathsf{W}^*$ *succeeds in* $\mathcal{G}(k)] \leq \varepsilon(k)$.

Lemma 1. *Assume that the fully homomorphic encryption is semantically secure. Then the delegation scheme* Del_2 *has one-time soundness error* $1/2 + \mathsf{ngl}(k)$.

6 $\mathsf{Del}_3 = \langle \mathsf{D}_3, \mathsf{W}_3 \rangle$: One-time, Arbitrary-input Delegation Scheme with Negligible Soundness

In this section, we exploit the above computational randomization technique to improve the soundness. The idea is the following: The delegator D asks the worker to compute $\hat{F}$ on multiple independent rerandomized inputs $\hat{x}_i = \text{Enc}_{\mathsf{pk}}(x)$ together with multiple $\hat{r}_i$'s (sent in a random order), as opposed to a single $\hat{x}$ and a single $\hat{r}$. Upon receiving the worker's answers, the delegator D checks whether (i) the returned value for $\hat{r}_i$ is equal to $\hat{F}(\hat{r}_i)$ for every $\hat{r}_i$, and (ii) the decryption of the returned values for $\hat{x}_i$ are consistent, and accepts the consistent value if the worker's answers pass these two tests. Observe that for a malicious worker to cheat, he needs to simultaneously cheat on all the $\hat{x}_i$'s while providing correct answers on all the $\hat{r}_i$'s. The formal description of the delegation scheme $\mathsf{Del}_3 = \langle \mathsf{D}_3, \mathsf{W}_3 \rangle$ appears in Figure 3.

- **Inputs.** Security parameter 1^k and function $F : \{0,1\}^n \to \{0,1\}^m$, specified by a Turing machine M and a time bound T, and an additional parameter t.
- **Offline Stage.** Both D_3 and W_3 receive as input a function F
  1. D_3 generates a pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{KeyGen}(1^k)$, computes t independent encryptions $\hat{r}_i = \text{Enc}_{\mathsf{pk}}(\bar{0})$ and the homomorphic evaluations $\hat{w}_i = \hat{F}(\hat{r}_i) = \text{Eval}_{\mathsf{pk}}(\hat{r}_i, F)$ for $i \in [t]$, and stores pk, sk, and the pairs $(\hat{r}_1, \hat{w}_1), \ldots, (\hat{r}_t, \hat{w}_t)$ as his secret key.
- **Online Stage.** Both D_3 and W_3 receive an input $x \in \{0,1\}^n$.
  1. D_3 computes t independent encryptions $\hat{r}_{i+t} = \text{Enc}_{\mathsf{pk}}(x)$ for $i \in [t]$, samples a random permutation $\pi \in_R S_{2t}$, and sends the public key pk and $(\hat{z}_{\pi(1)}, \ldots, \hat{z}_{\pi(2t)}) = (\hat{r}_1, \ldots, \hat{r}_{2t})$ to W_3.
  2. W_3 computes $\hat{y}_i = \hat{F}(\hat{z}_i) = \text{Eval}_{\mathsf{pk}}(\hat{z}_i, F)$ for $i \in [2t]$, and sends to D_3 the tuple
  $$(\hat{y}_1, \ldots, \hat{y}_{2t}) = (\hat{F}(\hat{z}_1), \ldots, \hat{F}(\hat{z}_{2t})).$$
  3. D_3 checks two things. First, D_3 checks if $\hat{w}_i = \hat{y}_{\pi(i)}$ for all $i \in [t]$. Second, D_3 decrypts $\hat{y}_{\pi(i+t)}$ for $i \in [t]$, and checks if the decrypted values are the same. D_3 accepts and outputs the consistent decrypted value if the returned values pass the two tests.

Fig. 3. Delegation Scheme $\mathsf{Del}_3 = \langle \mathsf{D}_3, \mathsf{W}_3 \rangle$

In the following lemma, we argue that since the $\hat{x}_i$'s and the $\hat{r}_i$'s are computationally indistinguishable, the probability of cheating is exponentially small in t (which is the number of $\hat{x}_i$'s). Thus, by setting $t = \omega(\log k)$, the protocol $\langle \mathsf{D}_3, \mathsf{W}_3 \rangle$ achieves negligible soundness error.

Lemma 2. *Assume that the fully homomorphic encryption is semantically secure. Then the delegation scheme* Del_3 *has one-time soundness error* $\binom{2t}{t}^{-1} + \mathsf{ngl}(k)$.

7 The First Main Delegation Schemes Del_4

All the delegation schemes presented in Section 4 – 6 had only *one-time* soundness. Namely, the delegator could delegate the computation of only one input x.

In this section, we present *reusable* delegation schemes, which satisfy the (standard) soundness property of Definition 4. To this end, we abstract the idea of Gennaro, Gentry, and Parno [GGP09] and present a generic transformation that converts any delegation scheme with one-time soundness to a reusable delegation scheme (i.e., one which satisfies the soundness property of Definition 4). Applying the transformation to the previous delegation scheme Del_3, we obtain our first main delegation scheme Del_4.

For intuition, let us take a closer look at why the previous delegation scheme $\mathsf{Del}_3 = \langle \mathsf{D}_3, \mathsf{W}_3 \rangle$ is not reusable. Recall that in that scheme it is essential for the worker to not know the $\hat{r}_i$'s: Once a malicious worker W^* learns the values of the $\hat{r}_i$'s, he can easily cheat by answering correctly only on those $\hat{r}_i$'s. Therefore, each precomputed pair $(\hat{r}_i, \hat{C}(\hat{r}_i))$ can be used only once. Phrased more abstractly, the security of the protocol $\langle \mathsf{D}_3, \mathsf{W}_3 \rangle$ relies on assumption that the secret key of the delegator D_3 (i.e., the pairs $(\hat{r}_i, \hat{C}(\hat{r}_i))$), remains secret. However, in that protocol, this secret key is revealed after delegating one input.

To make the protocol reusable, we use the idea of [GGP09], of running the protocol under a fully-homomorphic encryption scheme. Namely, our transformation takes any delegation scheme $\mathsf{Del} = \langle \mathsf{D}, \mathsf{W} \rangle$ which has only one-time soundness, and converts it into a new delegation scheme $\tilde{\mathsf{Del}} = \langle \tilde{\mathsf{D}}, \tilde{\mathsf{W}} \rangle$ with (standard) soundness, as follows: The delegator $\tilde{\mathsf{D}}$, instead of sending the message of D in the clear (which may reveal information about his secret key), will send a public key pk corresponding to a fully homomorphic encryption scheme, and will send the message of D encrypted under the public key pk. The worker $\tilde{\mathsf{W}}$ will then use the homomorphic property of the encryption scheme, to compute an encrypted reply of W. This enables the delegator $\tilde{\mathsf{D}}$ to hide its message (which contains the information about the delegator's secret key) from the worker $\tilde{\mathsf{W}}$, while still allowing the worker to do the computation for the delegator. A formal description of the transformation can be found in Figure 4.

The transformation. Let $\mathsf{Del} = \langle \mathsf{D}, \mathsf{W} \rangle$ be a *one-time* delegation scheme. We define a transformed delegation scheme $\tilde{\mathsf{Del}} = \langle \tilde{\mathsf{D}}, \tilde{\mathsf{W}} \rangle$ from Del as follows.

- **Inputs.** Security parameter 1^k and function $F : \{0,1\}^n \to \{0,1\}^m$, specified by a Turing machine M and a time bound T.
- **Offline Stage.** $\tilde{\mathsf{Del}}$ has exactly the same offline stage as Del.
  (Recall that in this stage both players receive a function F.)
- **Online Stage.** both $\tilde{\mathsf{D}}$ and $\tilde{\mathsf{W}}$ receive input $x \in \{0,1\}^n$.
  1. $\tilde{\mathsf{D}}$ generates a fresh pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{KeyGen}(1^k)$ of a fully-homomorphic encryption scheme, computes D's message $q = \mathsf{D}(F, x, \sigma_{\mathsf{D}})$ and its encryption $\hat{q} = \mathrm{Enc}_{\mathsf{pk}}(q)$, and sends pk and $\hat{q}$ to $\tilde{\mathsf{W}}$.
  2. $\tilde{\mathsf{W}}$ homomorphically computes an encrypted version of W's message $\hat{a} = \mathrm{Eval}(\hat{q}, \mathsf{W}(F, x, \sigma_{\mathsf{W}}, \cdot))$, and sends $\hat{a}$ to $\tilde{\mathsf{D}}$.
  3. $\tilde{\mathsf{D}}$ decrypts $\hat{a}$ to obtain $a = \mathsf{W}(F, x, \sigma_{\mathsf{W}}, q)$, and computes his decision and his output according to D.

Fig. 4. Transforming *one-time* delegation scheme Del into a *reusable* delegation scheme $\tilde{\mathsf{Del}}$

We next analyze the properties of the resulting (reusable) delegation scheme $\widetilde{\mathsf{Del}}$.

- If the one-time delegation scheme Del has a non-interactive off-line stage, then so does $\widetilde{\mathsf{Del}}$, since the offline stage remains unchanged.
- If the one-time delegation scheme Del has an efficient worker W, then the resulting (reusable) delegation scheme $\widetilde{\mathsf{Del}}$ also has an efficient worker $\widetilde{\mathsf{W}}$, since $\widetilde{\mathsf{W}}$ does the same computation as W, but in an encrypted manner.
- The fact that the complexity of the algorithms $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ are independent of the runtime of F, implies that if the one-time delegation scheme Del has an efficient delegator D then the delegator $\widetilde{\mathsf{D}}$ in the resulting (reusable) delegation scheme $\widetilde{\mathsf{Del}}$ is also efficient.
- The completeness of the fully homomorphic encryption scheme implies that if the one-time delegation scheme Del is complete then the resulting (reusable) delegation scheme $\widetilde{\mathsf{Del}}$ is also complete.

Thus, it remains to analyze the soundness of the resulting delegation scheme $\widetilde{\mathsf{Del}}$. Intuitively, by using a fully homomorphic encryption scheme, the information about the delegator's secret is not leaked, and so the delegator can reuse the secret key to delegate the computation on multiple inputs. However, note that not only the delegator's message, but also the delegator's decision bit can leak information about the delegator's secret key , since the delegator's decision depends on his secret key . Hence, in the security game (see Definition 3), the delegator terminates the scheme once he rejects to ensure the delegator's secret key is not leaked. (As discussed in Section 3, an alternative option is to assume that the worker does not learn the decision of the delegator, and our scheme is also sound in this model.)

Lemma 3. *Assume that the fully homomorphic encryption is semantically secure. Let* $\mathsf{Del} = \langle \mathsf{D}, \mathsf{W} \rangle$ *be a delegation scheme with negligible one-time soundness error, and let* $\widetilde{\mathsf{Del}} = \langle \widetilde{\mathsf{D}}, \widetilde{\mathsf{W}} \rangle$ *be the delegation scheme obtained by applying to* Del *the transformation described in Figure 4. Then* $\widetilde{\mathsf{Del}}$ *also has negligible soundness error.*

Applying the above transformation to the previous delegation scheme Del_3, we obtain our main delegation scheme Del_4. We summarize the properties of Del_4 in the following theorem.

Theorem 1. *Assume that the fully homomorphic encryption scheme is semantically secure. Then the delegation scheme* $\mathsf{Del}_4 = \langle \mathsf{D}_4, \mathsf{W}_4 \rangle$ *has the following properties, for delegating the computation of a function* $F : \{0,1\}^n \to \{0,1\}^m$ *computable by a Turing machine* M *that runs in time* $T \geq \max\{n, m\}$*, on security parameter* k*:*

- *Perfect completeness and negligible soundness error.*
- *Non-interactive offline stage, with* D_4 *running in time* $\mathrm{poly}(T, |M|, k)$ *and generating a secret key of length* $\mathrm{poly}(n, m, k)$*, but not creating any public key.*

- *2-message online stage, with* D_4 *running in time* $\mathrm{poly}(n, m, k)$ *and* W_4 *running in time* $\mathrm{poly}(T, |M|, k)$*. That is, both* D_4 *and* W_4 *are efficient in the online stage.*

8 The Second Main Delegation Scheme Del_5

We note that in all the delegation schemes presented in Section 4 − 7, the delegator needs to run heavy computations in the offline stage. For example, in the offline stage of Del_4, the delegator needs to compute pairs of the form $(\hat{r}_i, \hat{F}(\hat{r}_i))$, where each $\hat{r}_i \leftarrow \mathrm{Enc}_{\mathsf{pk}}(\bar{0})$, and therefore runs in time comparable to the runtime of F.

In this section, we show how to make the offline stage efficient by delegating its computation as well. However, since we do not know how to do non-interactive delegation (this is the problem we started with!), this will come at the price of making the offline stage interactive. In particular, we will use *universal arguments*, a notion developed by [Mic94,Kil92,BG02], and which yields a 4-message delegation scheme. However, we cannot apply universal arguments directly, as they allow the worker to learn the result of the computation, which in our case is supposed to be the secret state of the delegator. To solve this problem, we use yet another layer of fully homomorphic encryption, and use the universal argument to delegate an encrypted form of the computation.

8.1 Universal Arguments

Consider the language

$$L_{\mathrm{uni}} \triangleq \{(M, x, y, t) : M \text{ is a Turing machine that on input } x \text{ outputs } y \text{ after at most } t \text{ steps}\}$$

Definition 6 (Universal Arguments [BG02]). *A* **universal argument system** *is a pair of interactive Turing machines, denoted by* (P, V)*, that satisfy the following properties.*

- **Efficient verification.** *There exists a polynomial* p *such that for any* $z = (M, x, y, t)$ *the total runtime of* V*, on common input* z*, is at most* $p(|z|)$*. In particular, all the messages exchanged in the protocol have length smaller than* $p(|z|)$*.*
- **Completeness via a relatively-efficient prover.** *For every* $(M, x, y, t) \in L_{\mathrm{uni}}$*,* $\Pr[(P, V)(M, x, y, t) = 1] = 1$*.*
  Furthermore, there exists a polynomial p *such that for every* $(M, x, y, t) \in L_{\mathrm{uni}}$*, the total runtime of* P *on input* $z = (M, x, y, t)$ *is at most* $p(|M|, t)$*.*
- **Computational soundness.** *For every polynomial-size circuit family* $P^* = \{P^*_n\}_{n\in\mathbb{N}}$ *there exists a negligible function* μ *such that for every* $(M, x, y, t) \in \{0,1\}^n \setminus L_{\mathrm{uni}}$*,* $\Pr[(P^*_n, V)(M, x, y, t) = 1] \le \mu(n)$*.*

Remark. We note that Barak and Goldreich [BG02] consider a more general language L, where they allow the Turing machine M to be non-deterministic. Moreover, they require an additional proof-of-knowledge type property. In this work, we are only interested in deterministic Turing machines, and only focus on the properties that we need.

Theorem 2 ([Kil92,Mic94,BG02]). *Assuming the existence of collision-resistant hash functions, there exists a 4-message (2-round) universal argument system.*

We remark that the existence of fully homomorphic encryption schemes implies the existence of collision-resistant hash functions [IKO05].

8.2 Our New Delegation Scheme Del_5

We now show how to use a universal argument (P, V), together with a fully homomorphic encryption scheme $\mathrm{E} = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$, to convert any delegation scheme $\mathsf{Del} = (\mathsf{D}, \mathsf{W})$ with a non-interactive offline stage into a delegation scheme $\tilde{\mathsf{Del}} = (\tilde{\mathsf{D}}, \tilde{\mathsf{W}})$, such that the online stage remains unchanged, but the offline stage of $\tilde{\mathsf{Del}}$ is now interactive (consists of 4 messages) and the delegator $\tilde{\mathsf{D}}$ is *efficient* in the offline stage.

Instead of having the delegator carry out its computations on its own in the offline stage, it will use a worker to do it for him. However, as previously noted, there is a subtle issue here: the result of the computation done by the delegator in the offline stage should remain secret for soundness to hold. Therefore, we cannot simply delegate this computation. Instead, will delegate this computation in a secret manner; namely, we will do a universal argument over encrypted data, as follows.

Suppose without loss of generality, that in the offline stage the delegator D chooses some randomness $r \in \{0,1\}^\ell$ for $\ell = \mathrm{poly}(k)$[4] computes a function $g(r)$, where g may depend on both the delegated function F and the security parameter k. The delegator D can delegate this computation, in a secret manner, by giving the worker an encryption of r (rather than r in the clear); i.e., giving the worker a pair $(\mathsf{pk}, \mathrm{Enc}_{\mathsf{pk}}(r))$, and delegating the computation of the function $\mathrm{Eval}_{\mathsf{pk}}(\mathrm{Enc}_{\mathsf{pk}}(r), g)$ to the worker, by running a universal argument protocol. Then all the delegator needs to do is to decrypt the message he gets from the worker. A formal description of this transformation can be found in Figure 5.

Analysis of our transformation can be found in the full version of this paper [CKV10]. By applying the transformation above to the delegation scheme Del_4, and relying on Theorem 1, we get the following theorem.

Theorem 3. *Assume that there exists a fully homomorphic encryption scheme. Then there is a secure delegation scheme* $\mathsf{Del} = \langle \mathsf{D}, \mathsf{W} \rangle$ *with the following properties, for delegating the computation of a function* $F : \{0,1\}^n \to \{0,1\}^m$ *computable by a Turing machine* M *that runs in time* $T \geq \max\{n, m\}$*, on security parameter* k*:*

[4] The randomness of D can always be reduced to $\mathrm{poly}(k)$ by use of a pseudorandom generator if needed.

The transformation. Let $\mathsf{Del} = \langle \mathsf{D}, \mathsf{W} \rangle$ be a delegation scheme with a non-interactive offline stage. We define a transformed delegation scheme $\tilde{\mathsf{Del}} = \langle \tilde{\mathsf{D}}, \tilde{\mathsf{W}} \rangle$ from Del as follows.

- **Inputs.** Security parameter 1^k and function $F : \{0,1\}^n \rightarrow \{0,1\}^m$, specified by a Turing machine M and a time bound T.
- **Offline Stage.** Both $\tilde{\mathsf{D}}$ and $\tilde{\mathsf{W}}$ receive as input the functin F.
  Suppose that in the delegation scheme Del, the delegator D chooses a random $r \leftarrow \{0,1\}^\ell$ (where $\ell = \text{poly}(k)$) and computes $\sigma_{\mathsf{D}} = \mathsf{D}(1^k, F; r)$. We denote by $g(\cdot) \overset{\text{def}}{=} \mathsf{D}(1^k, F; \cdot)$. The offline stage of $\tilde{\mathsf{Del}}$ proceeds as follows.
  1. The delegator $\tilde{\mathsf{D}}$ chooses a random $r \leftarrow \{0,1\}^\ell$; chooses a random key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \text{KeyGen}(1^k)$; computes $\hat{r} = \text{Enc}_{\mathsf{pk}}(r)$; and sends the pair $(\mathsf{pk}, \hat{r})$ to the worker $\tilde{\mathsf{W}}$.
  2. The worker $\tilde{\mathsf{W}}$ computes $c = \text{Eval}_{\mathsf{pk}}(\hat{r}, g)$ and sends c to the delegator.
  3. Then the worker $\tilde{\mathsf{W}}$ and the delegator $\tilde{\mathsf{D}}$ engage in a universal argument, where the worker proves to the delegator that indeed $c = \text{Eval}_{\mathsf{pk}}(\hat{r}, g)$.
  4. If the delegator $\tilde{\mathsf{D}}$ accepts the universal argument, then he decrypts the ciphertext c and outputs $\sigma_{\mathsf{D}} \leftarrow \text{Dec}_{\mathsf{sk}}(c)$.
- **Online Stage.** The online stage of $\tilde{\mathsf{Del}}$ is identical to the online stage of Del.

Fig. 5. Transforming delegation scheme Del with non-interactive but inefficient offline stage into $\tilde{\mathsf{Del}}$ with an *efficient* but interactive offline stage

- Del *has perfect completeness and negligible soundness error.*
- *The offline stage consists of 4 messages, with* D *running in time* $\text{poly}(n, m, k, |M|, \log T)$ *and* W *running in time* $\text{poly}(T, |M|, k)$*.*
- *The offline stage produces a secret key of length* $\text{poly}(n, m, k)$ *for* D*, and no public key.*
- *In the (2-message) online stage,* D *runs in time* $\text{poly}(n, m, k)$ *and* W *runs in time* $\text{poly}(T, |M|, k)$*.*

Thus, D *and* W *are efficient in both stages.*

8.3 Pipelined Implementation of Del_5

As mentioned in the introduction, the soundness of our main schemes Del_4 and Del_5 is only guaranteed as long as the adversarial worker does not learn that the delegator has rejected a proof, as this may leak information about the delegator's secret key. Hence, the delegator needs to re-run the offline stage after rejection.

Our "pipelined" scheme avoids this issue by having the delegator keep c secret keys (for a constant c) and continually refresh them during the online stage. Recall that Del_5 has an efficient but 4-message offline stage where the delegator delegates the computation of his secret key to a worker. The idea is that, in each execution of the 2-message online stage, the delegator and the worker shall simultaneously run $2c$ copies of offline stages in the background. These are run in a pipelined fashion so that with each online stage, c copies of the offline stage are finished and can be used to refresh secret keys that are expired. We consider a secret key to be expired when it has been used in an online stage of Del_5 in which the delegator has rejected. Thus, the delegator will always have a fresh secret key available provided that for every c online stages in which there is an error (i.e. rejection), there are at least 2 consecutive errorless stages. We note that this implementation requires the worker and delegator to maintain state, and thus is most useful for settings in which the delegator is interacting with a single worker for many executions and wishes to avoid disruption from benign faults. (If the worker were truly cheating, then it seems prudent to halt the interaction and restart with a different worker...)

Acknowledgments

We are grateful to Boaz Barak for collaboration at the start of this research, and for sharing his insights with us.

References

And03. David P. Anderson. Public computing: Reconnecting people to science. In *Conference on Shared Knowledge and the Web*, 2003.

And04. David P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID*, pages 4–20, 2004.

Bab85. László Babai. Trading group theory for randomness. In *STOC*, pages 421–429, 1985.

Bar01. Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, pages 106–115, 2001.

BCC88. Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.

BFL91. László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

BFLS91. László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31, 1991.

BG02. Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *IEEE Conference on Computational Complexity*, pages 194–203, 2002.

CGH04. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4):557–594, 2004.

CKV10. Kai-Min Chung, Yael Kalai, and Salil Vadhan. Improved delegation of computation using fully homomorphic encryption. Cryptology ePrint Archive, Report 2010/241, 2010. `http://eprint.iacr.org/`.

FL93. Lance Fortnow and Carsten Lund. Interactive proof systems and alternating time-space complexity. *Theoretical Computer Science*, 113(1):55–73, 1993.

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

GGP09. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. Cryptology ePrint Archive, Report 2009/547, 2009. http://eprint.iacr.org/.

GK03. Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. pages 102–113, 2003.

GKR08. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

GMR89. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

IKO05. Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Sufficient conditions for collision-resistant hashing. In *TCC*, pages 445–456, 2005.

Kil92. Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

KR09. Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In *CRYPTO*, 2009.

LFKN92. Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

Mer07. The great internet mersenne prime search, project webpage. In *http://www.mersenne.org/*, 2007.

Mic94. Silvio Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453, 1994.

Mic00. Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

Sha92. Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

vAMM$^+$08. Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science*, 321(5895):1465–1468, 2008.