On Single-Server Delegation of RSA Decryption

G. Di Crescenzo*, M. Khodjaeva†, D. Kahrobaei‡, V. Shpilrain§

Abstract

In delegated computation research, the main problem asks how a computationally weaker client device can obtain help from one or more computationally stronger server devices to perform some computation. Desirable solution requirements include correctness of the computation, privacy of the inputs, high probability detection of any server malicious behavior, low client runtime in the online phase, low communication complexity, low client storage complexity, and minimal server trust.

We summarize the state of the art on delegated computation of RSA decryption, a particularly elusive problem which has been studied for 30+ years with several attempted solutions and successful attacks. We discuss some of the most recent progress, which includes privacy and security models, and provable solutions for related problems, such as delegating large-exponent RSA encryption, which imply a transformation to achieve the result security requirement. We also present three 'partial solution' protocols which provably meet all but one of the desired requirements; specifically, one protocol requires two non-colluding servers instead of one, and two other protocols require high communication complexity, with somewhat different performance tradeoffs. We conclude with directions for future research.

1 Introduction and Model

The area of server-aided cryptography, or delegation/outsourcing of cryptographic primitives, is mainly concerned with the following problem: "how can a computationally weaker client delegate cryptographic computations to computationally superior servers?"

This problem has been first discussed in [1, 2, 3] and a first formal model has been produced in [4]. In the past few years, this problem is seeing an increased interest because of the shift of modern computation paradigms towards cloud/fog/edge computing, large-scale computations over big data, Internet of Things, etc. A solution to this problem is an interactive protocol between a client and one or more servers, where the client holding an input x wants to delegate to the server(s) computation of a publicly known function F on input x, and the main desired properties of this delegated computation of $F(x)$ are:

1. *result correctness*: if client and server(s) honestly follow their instructions, at the end of the protocol the client obtains $F(x)$;
2. *input/output privacy*: only minimal or no information about x and/or $F(x)$ is revealed to the server(s); here, [4] formally defines privacy in the sense of simulatability of the client's messages (as in the area of secure multiparty computation), and [5, 6] considers input privacy in the sense of input indistinguishability; that is, even malicious servers cannot distinguish two different inputs from a protocol's execution, except possibly with very small probability ϵ_p;
3. *result ϵ_s-security*: even malicious server(s) should not be able, except possibly with very small probability (i.e., $\epsilon_s = 2^{-\lambda}$, for some statistical parameter λ) to convince the client to accept a result different than $F(x)$; and
4. *resource efficiency*: client's runtime t_C should be significantly smaller than the runtime t_F for computing $F(x)$ without delegation; usage of other resources like communication complexity cc, client's storage complexity sc and runtime of any offline phase t_P, should also be minimized.

Following, for instance, [7], protocols can be partitioned into (a) an offline phase, where input x is not yet known, but somewhat expensive computation can be performed by the client deployer or even the temporarily unconstrained client's device, and stored on the client's device, and (b) an online phase, where we assume the client's resources are limited, and thus the client needs the server's help to compute $F(x)$. To capture meaningfully distinct input scenarios in delegation, we say that an input x to F is

*Peraton Labs, Basking Ridge, NJ, USA. Email: gdicrescenzo@peratonlabs.com

†CUNY John Jay College of Criminal Justice. New York, NY, USA. Email: mkhodjaeva@jjay.cuny.edu

‡CUNY Graduate Center. New York, NY, USA. Email: dkahrobaei@gc.cuny.edu

§City University of New York. New York, NY, USA. Email: vshpilrain@ccny.cuny.edu

- *public-online* if x is unknown in the offline phase but known to both parties in the online phase;
- *public-offline* if x is known to both parties starting from the offline phase;
- *private-online* if x is unknown in the offline phase but known to C in the online phase;
- *private-offline* if x is known to C starting from the offline phase but unknown to S.

The RSA Functions. Let p, q be primes of the same length, let $n = p, q$, and let $\mathbb{Z}_n^*$ denote the set of integers coprime with n. We consider the group $(\mathbb{Z}_n^*, \cdot)$, where $\cdot$ denotes multiplication modulo n. Let e, d denote integers such that $\gcd(e, \phi(n)) = 1$ and $\gcd(d, \phi(n)) = 1$. In this group, with parameter values n, e, d, we define the following functions:

1. *RSA encryption exponentiation* as $\mathsf{eExp}_{n,e} : m \in \mathbb{Z}_n^* \rightarrow c \in \mathbb{Z}_n^*$, such that $c = m^e \mod n$.
2. *RSA decryption exponentiation* as $\mathsf{dExp}_{n,e,d} : c \in \mathbb{Z}_n^* \rightarrow y \in \mathbb{Z}_n^*$, such that $y = c^d \mod n$.

We consider $\mathsf{eExp}_{n,e}$ as a base-private-online exponent-public-offline exponentiation function, and $\mathsf{dExp}_{n,e,c}$ as a base-public-online exponent-private-offline exponentiation function. The textbook algorithm to compute functions $\mathsf{eExp}_{n,e}$ and $\mathsf{dExp}_{n,e,c}$ is the *square-and-multiply* algorithm, which requires up to 2ℓ multiplications modulo n, but see, e.g. [8], for algorithms with a slightly improved constant.

This paper. Recently, single-server delegation protocols provably satisfying the defined properties of result correctness, input/output-privacy, result-security and resource efficiency, have been proposed for some operations often found in cryptographic protocols, including: exponentiation in discrete logarithm groups (see, e.g., [19]), large-exponent RSA encryption (see, e.g., [9]), pairings (see, e.g., [24, 26, 27]), modular multiplication (see, e.g., [28]). On the other hand, the problem of delegating RSA decryption remains particularly elusive, despite several attempts in the literature. Accordingly, we summarize the state of the art on delegated computation of RSA decryption, starting from the first protocols, proposed in [3], and going into more recent protocols and potential lower bound results. We then present three 'partial solution' protocols, based on recent work on delegating large-exponent RSA encryption [9], which provably meet all but one of the desired properties; specifically, one protocol, in Section 3.2, requires two non-colluding servers, and two additional protocols, in Section 3.3 and in Section 3.4, require high communication complexity. The latter of these two protocols slightly improves the communication complexity with only minor worsening of all other properties. Both protocols are also contrasted with previous literature proposals which also exhibited high communication complexity (i.e, [25]).

2 Previous Work

Delegation of RSA Decryption. In [3], the first paper proposing delegation of cryptographic algorithms, the authors presented two protocols for the delegation of RSA decryption. The basic idea of such a protocols was as follows: the client sends some randomized masking of exponent d to the server; the server computes exponentiations to exponents related to the masking, and sends the results to the client; finally, the client uses the mask computation to turn the received exponentiation results into the desired $x^d \mod n$ exponentiation. The specific masking used in the first of their protocols was

$$d = f_1 d_1 + \cdots + f_M d_M \mod \phi(n),$$

for some random integers $d_1, \ldots, d_M$ send by the client to the server, some random bits $f_1, \ldots, f_M$ kept secret by the client, and some small value M. While this might seem an interesting approach, in that recovering d might seem to require exhaustive search of all possible vectors $(d_1, \ldots, d_M)$, about 20 papers were published containing faster attacks to their protocols and/or proposing protocol variants and improvements, as well as faster attacks to these variants. The reader is referred to Section 2 of [10] and Section 1.2 of [20] for a detailed discussion of these papers. All of these results were published before the introduction of a formal delegation model [4], and therefore protocols were described without proofs for their properties, other than sometimes claiming security against all previous attacks. For some of these papers, it might be interesting to study if their techniques suffice to provably achieve some of the properties formally defined in the more recent delegation models. In particular, some of these attacks were based on an attacker's knowledge of signatures, which would not necessarily be part of the adversary model when considering decryption delegation.

More recently, protocols were proposed [11, 12, 13, 14, 15, 16] where the RSA exponent was hidden in one or more linear equations depending on a random value t in the exponent group. As we analyzed these protocols, we noted the following properties: (a) a full-domain value t would perfectly hide the RSA secret key from the server, but require client work comparable to non-delegated computation of RSA decryption; (b) a smaller-size value t would reduce the client's work but also proportionally reduce the work needed to derive the RSA secret key. Properties (a)

Table 1: Summary of lower bound results from Theorems in [22] for protocols for the delegation of public-online-base private-offline-exponent exponentiation in prime order groups, in the generic group model, where s is an arbitrary integer. Each row represents a barrier in the sense that a protocol with a better improvement factor than what written in the 2nd column, in a scenario with protocol parameters as described in columns 3-5, would imply an efficient attack that successfully violates input privacy.

Lower Bound	Improvement Factor (over non-deleg sq and mult alg)	Offline Client Exponentiations	Online Server Exponentiations	# of rounds
Theorem 2	4	$O(1)$	1	1
Theorem 3	$(s+1)/2$	$O(1)$	s	1
Theorem 4	$\ell+2$	$\ell = O(1)$	1	1
Theorem 5	8	$O(1)$	2	2
Theorem 6	$(4+(s+1)^2)/2$	$O(1)$	s	2
Theorem 7	2^{s+1}	$O(1)$	s	s

and (b) imply that these protocols do not simultaneously satisfy input privacy and client resource efficiency. Indeed, the variant of this approach used in [11] was broken by [22] using lattice-based cryptanalysis techniques.

Other exponent masking attempts were proposed in [17, 18], where the exponent was masked by a multiple of the group order. This would seem a potentially interesting and valid idea, since, on one hand the random value would mask d, and on the other hand the server's exponentiation to a multiple of the group order would cancel out and allow C to recover an exponentiation to the original exponent. However, these protocols were broken in [21] using lattice-based cryptanalysis techniques.

Finally, [22] proves, in the generic group model, lower bounds on the efficiency of delegation protocols for a class of functions, including one that has some similarity to RSA decryption: public-online-base private-offline-exponent exponentiation in prime-order groups. These results are summarized in Table 1. We note that it is yet unknown if these results can be extended to RSA groups. Even if they were, it would still be possible to design RSA decryption delegation protocols with non-trivial improvements over non-delegated computation.

Delegation of RSA Encryption. Early papers in the area mentioned that low-exponent RSA encryption is much more efficient than RSA decryption, and did not attempt delegation protocols for it. Recent results in [6, 23, 9] include protocols for the delegation of large-exponent exponentiation in RSA encryption. All of these latter results formally prove protocol properties, also captured in Table 2. In particular, in [9], a client performs about 3λ multiplications if result $2^{-\lambda}$-security is desired. We later use these results so that the verification component of the RSA decryption delegation protocols in Sections 3.2, 3.3 and 3.4 do not need to assume that RSA encryption was computed using a small exponent.

In what follows, we denote by $\mathcal{P}_e$ a delegation protocol for the RSA encryption (large-exponent) exponentiation function $\mathsf{eExp}_{\mathsf{n,e}} : x \rightarrow x^e \mod n$, satisfying correctness, input privacy, and ϵ-security. $\mathcal{P}_e$ can be instantiated using any of the protocols in Table 2.

3 RSA Decryption Delegation: Partial Solution Protocols

In this section we present protocols that only partially solve the RSA decryption delegation problem, as previously stated.

We start by showing a transformation that transforms a protocol that does not achieve the result security property into one that does, using protocol $\mathcal{P}_e$. We then show 3 protocols where the server(s) are honest-but-curious: a 2-server protocol with efficient communication complexity (in Section 3.2), a single-server protocol with somewhat inefficient communication complexity (in Section 3.3), and another single-server protocol with a somewhat different performace tradeoff that results in slighty improved communication complexity (in Section 3.4).

3.1 A Transformation to Achieve the Result Security Property

We observe that if there exists a delegation protocol $\mathcal{P}$ for the RSA decryption exponentiation function $\mathsf{dExp}_{\mathsf{n,e,c}} : c \rightarrow c^d \mod n$ satisfying correctness and input privacy, then we can construct a delegation protocol $\mathcal{P}'$ for the same function, satisfying correctness, input privacy, and result ϵ-security. The transformation goes as follows: in $\mathcal{P}'$, C and S run one execution of protocol $\mathcal{P}$ on the same inputs, thus returning m

Table 2: Results on provable delegation of large-exponent RSA Encryption where t_C is C's runtime in the online phase; t_P is C's runtime in the offline phase; cc is the communication exchanged between C and S; mc is the number of messages exchanged between C and S; sc is the storage complexity of C; $t_{exp}(\ell)$ is an exponentiation to an ℓ-bit exponent; we can set m as an arbitrary integer (e.g., $m \leq 100$) and $c \in \{4, \ldots, 9\}$.

	t_C		t_P		cc	sc	mc	ϵ_s
	$t_{exp}(\lambda)$	# mult	$t_{exp}(\sigma)$	# inv				
[23]	0	2	c	1	$2m$	$m+c$	2	$\sim O(1/c)$
[9]	2	5	2	1	4	4	2	$2^{-\lambda}$
[6]	0	5	2	0	6	4	2	1/2

to C; at the end of this execution, C checks the obtained result m by using delegation protocol $\mathcal{P}_e$ for $\mathsf{eExp}_{\mathsf{n,e}} : x \rightarrow x^e \mod n$, setting $x = m$, and thus obtaining the result c'; finally, C checks that $c' = c$.

We note that $\mathcal{P}'$ preserves the efficiency of $\mathcal{P}$ in all metrics; in particular, if C is efficient in $\mathcal{P}$, then C is also efficient in $\mathcal{P}'$, where it only performs 2 additional exponentiations to a λ-bit exponent in the online phase, to achieve the result λ-security property. In practice λ can be set much smaller than $|n|$; e.g., $\lambda = 50$. Finally, note that with respect to $\mathcal{P}$, protocol $\mathcal{P}'$ requires two additional messages, the computation of 2 exponentiations in the offline phase, and C's computation of 2 λ-bit-exponent exponentiations in the online phase, when requiring result $2^{-\lambda}$-security.

3.2 A 2-Server Delegation Protocol

We describe a 2-server protocol $\mathcal{P}_1 = (C, S_1, S_2)$ for the delegated computation of $\mathsf{dExp}_{\mathsf{n,e,c}} : c \rightarrow c^d \mod n$.

Informally, protocol $\mathcal{P}_1$ goes as follows. C splits the secret exponent d into two random values that sum to d modulo $\phi(n)$; then, C asks each server for an exponentiation of c to one of these values; and finally, C returns the product mod n of the values received by the two servers.

A formal description of protocol $\mathcal{P}_1$ and its properties follows.

Input to C and $S = (S_1, S_2)$: n, e, c, $desc(\mathsf{dExp}_{n,e,c})$

Input to C: $d \in \{1, \ldots, \phi(n) - 1\}$, $\phi(n)$, $c \in \mathbb{Z}_n^*$

Protocol Steps:

1. C randomly chooses $u \in \{2, \ldots, \phi(n) - 1\}$, C sends u to S_1 and $z := d - u \mod \phi(n)$ to S_2.
2. S_1 computes and sends $w_1 := c^u \mod n$ to C
3. S_2 computes and sends $w_2 := c^z \mod n$ to C
4. C computes $m = w_1 \cdot w_2 \mod n$.

Protocol Properties: The *efficiency* properties are verified by protocol inspection. In particular,

- *C's runtime complexity* consists of 1 multiplication in $\mathbb{Z}_n^*$;
- *S's runtime complexity* consists of 1 exponentiation in $\mathbb{Z}_n^*$ for each server;
- with respect to *round complexity,* $\mathcal{P}_1$ only requires 2 messages between C and each server; and
- with respect to *communication complexity,* $\mathcal{P}_1$ only requires the transfer of 2 values in $\mathbb{Z}_n^*$ and 2 values in $\mathbb{Z}_{\phi(n)}$.

The *correctness* property follows by showing that if C, S_1 and S_2 follow the protocol, C always outputs $m = c^d \mod n$ since $m = w_1 \cdot w_2 = c^u \cdot c^z = c^u \cdot c^{d-u} = c^d \mod n$.

The *1-server input privacy* property is verified by showing that no information about d is leaked to any single server from the message received by C, since each one between u and $d - u \mod \phi(n)$ is uniformly distributed in $\mathbb{Z}_n^*$, and we assume the two servers do not collide.

Discussion. We note that the *result security* property can be achieved for this protocol by combining it with the transformation in Section 3.1 (i.e., C delegates the computation of $c' = m^e \mod n$ using any of the protocols in Table 2 and checks that $c' = c$). The properties of the resulting protocol are summarized in Table 4, where they are also compared with 2 protocols presented in [25], which turn out to have somewhat similar properties. This combination of protocols $\mathcal{P}_1$ and $\mathcal{P}_e$ comes close to solving the RSA decryption delegation problem, except for requiring two non-colluding servers. It is also not hard to extend it into a protocol which requires s servers, of which up to $s - 1$ can collide. However, known approaches do not seem to help in transforming this protocol into a 1-server protocol, as we now explain. First, note that both w_1 and w_2 do depend on c which is a public-online input and thus cannot be precomputed by C. Next, note that C cannot ask S_1 to compute both w_1 and w_2 as that would leak d. Finally, as discussed in section 2, two literature approaches to mask d in a message from C to a single server S, using linear random equations and/or multiples of the group order, have resulted in broken protocols.

3.3 A Time-Efficient Communication-Inefficient Protocol

We describe a 1-server protocol $\mathcal{P}_2 = (C, S)$ for the delegated computation of $\mathsf{dExp}_{\mathsf{n,e,d}} : c \to c^d \mod n$.

Informally, protocol $\mathcal{P}_2$ goes as follows. First, secret exponent d is considered in its b-ary representation, for some parameter $b \geq 2$. By viewing this representation as a k-term summation, S computes all possible b exponentiations to each different power of b as exponent, and sends all these $k(b-1)$ exponentiations to C. Finally, C can use the exponent representation coefficients to select the appropriate k values among the received exponentiations, and multiply them to obtain $c^d \mod n$.

A formal description of protocol $\mathcal{P}_2$ and its properties follows.

Input to C and S: n, e, c and parameter base b; also, let k be the min value such that $b^k \geq n$

Private input to C: $d \in \mathbb{Z}_{\phi(n)}$

Offline phase instructions:

1. Write private exponent d in base b; i.e., $d = (d_{k-1}, \ldots, d_1, d_0)_b = \sum_{i=0}^{k-1} d_i \cdot b^i$, where $d_i \in [0, b-1]$ for $i \in \{0, \ldots, k-1\}$.
2. C stores $(d_{k-1}, \ldots, d_1, d_0)_b$ on its device.

Online phase instructions:

1. S sets $w_{0,1} := c$ and $w_{0,j} := w_{0,1} \cdot w_{0,j-1} \mod n$, for $j = 2, \ldots, b-1$
   For $i = 1, \ldots k-1$
   S sets $w_{i,1} := w_{i-1,1}^b \mod n$
   For $j = 2, \ldots b-1$
   S sets $w_{i,j} := w_{i,1} \cdot w_{i,j-1} (= w_{i,1}^j) \mod n$
   S sends $w_{i,j}$ for $i = 0, .., k-1$, and $j = 1, .., b-1$
2. C sets $m = w_{0,d_0}$
   For $i = 1, \ldots k-1$
   C sets $m = m \cdot w_{i,d_i} \mod n$

Protocol Properties: The *efficiency* properties are verified by protocol inspection; in particular,

- *C's runtime complexity* consists of $k-1$ multiplications, which improves over non-delegated multiplication by a multiplicative factor of $\geq \log b$;
- *S's runtime complexity* consists of $k-1$ exponentiations of power b and $(k-1) \cdot (b-2)$ multiplications in $\mathbb{Z}_n^*$;
- the *offline runtime complexity* only takes time linear in $|d|$;
- with respect to *round complexity,* $\mathcal{P}_2$ only requires 2 messages between C and S; and
- with respect to *communication complexity*, the online phase of $\mathcal{P}_2$ requires the transfer of $k \cdot (b-1) + 1$ values in $\mathbb{Z}_n^*$.

The *correctness* property follows by showing that if C and S follow the protocol, C always outputs $m = c^d \mod n$ since $d = (d_{k-1}, \ldots, d_1, d_0)_b = \sum_{i=0}^{k-1} d_i \cdot b^i$ and c^d is

$$= c^{\sum_{i=0}^{k-1} d_i \cdot b^i} = \prod_{i=0}^{k-1} (c^{b^i})^{d_i} = \prod_{i=0}^{k-1} (w_{i,1})^{d_i} = \prod_{i=0}^{k-1} w_{i,d_i}.$$

The *input privacy* property holds with parameter $\epsilon_p = 0$ since C does not send any message to S during this protocol.

Discussion. We note that the *result security* property can be achieved for protocol $\mathcal{P}_2$ by combining it with the transformation in Section 3.1 (i.e., C delegates the computation of $c' = m^e \mod n$ using any of the protocols in Table 2 and checks that $c' = c$). The properties of the resulting protocol are summarized in Table 4. This combination of protocols $\mathcal{P}_2$ and $\mathcal{P}_e$ comes close to solving the RSA decryption delegation problem, except for requiring very high communication complexity; specifically, about $k(b-1)$ group values where parameters b, k are chosen so that $b^k \geq n$, and C's runtime in the online phase consists of $k-1$ multiplications mod n. However, known approaches do not seem to help in transforming this protocol into a communication-efficient protocol, as we now explain. First, note that choosing a smaller/larger value for b improves/worsens the communication complexity while worsening/improving C's runtime. Second, note that shorter vector-type representations of d have appeared in many past efforts, starting with [3], and none of them has been proved secure, as discussed in Section 2.

3.4 A Time-Efficient Communication-Improved Protocol

We show a 1-server protocol $\mathcal{P}_3 = (C, S)$ for the delegated computation of $\mathsf{dExp}_{\mathsf{n,e,d}} : c \to c^d \mod n$.

Informally, protocol $\mathcal{P}_3$ goes as follows. As before, secret exponent d is considered in its b-ary representation $d = \sum_{i=0}^{k-1} d_i \cdot b^i$, for some parameter $b \geq 2$. Note that $c^d \mod n$ can thus be seen as a product of exponentiations with bases c^{d_i} and powers of b as exponent. In this protocol C sends random masks of these bases and asks S to compute the product of exponentiation to the known power-of-b exponents. Upon receiving values from S, C can remove the masks by using a product of exponentiations computed in the offline phase, and thus recover $c^d \mod n$.

A formal description of protocol $\mathcal{P}_2$ and its properties follows. Let G be a pseudo-random generator.

Input to C and S: n, e, c and parameter base b; also, let k be the min value such that $b^k > n$.

Private input to C: $d \in \mathbb{Z}_{\phi(n)}$

Offline phase instructions:

Table 3: Performance results of our RSA Decryption delegation protocols with result security holding against a *malicious server* S, where t_C is C's runtime in the online phase; t_P is C's runtime in the offline phase; cc is the communication exchanged between C and S; mc is the number of messages exchanged between C and S; sc is the storage complexity of C; ϵ_s is the result security parameter, ϵ_p is the input privacy parameter; $t_{exp}(\ell)$ is an exponentiation to an ℓ-bit exponent, and $t_{prod,k,exp}(\sigma)$ is the time to compute a product of k exponentiations of group elements.

Protocol	t_C		t_P			cc	sc	mc	ϵ_s	ϵ_p	# Servers
	$t_{exp}(\lambda)$	mult	$t_{prod,k,exp}(\sigma)$	$t_{exp}(\sigma)$	inv						
$\mathcal{P}_1 + \mathcal{P}_e$	2	6	0	2	1	8	4	6	$2^{-\lambda}$	0	2
$\mathcal{P}_2 + \mathcal{P}_e$	2	$k+4$	0	2	1	$k(b-1)+5$	k	3	$2^{-\lambda}$	0	1
$\mathcal{P}_3 + \mathcal{P}_e$	2	$k+6$	1	2	2	$b+k+3$	$2k+5$	5	$2^{-\lambda}$	0	1

Table 4: Performance results of RSA Decryption delegation protocols from this paper as well as [25], when result security only holds against a "honest but curious" server S, and where we use the same parameters as in Table 3.

Protocol	t_C # of mult.	t_P # of exp.	t_S # of exp.	cc	sc	mc	ϵ_p	# Servers
$\mathcal{P}_1$	1	No	2	4	0	4	0	2
[25] [§2A]	$2b+k-4$	No	$k-1$	$k+1$	0	2	0	1
[25] [§2B]	$b+k$	No	0	$\sigma+1$	0	2	0	1
$\mathcal{P}_2$	$k-1$	No	$k-1$	$k(b-1)+1$	1	1	0	1
[25] [§2C]	$k+1$	Yes	$2b-2$	$b+k+1$	b	3	0	1
$\mathcal{P}_3$	$k+1$	Yes	k	$b+k-2$	3	3	negl	1

1. Write private exponent d in base b; i.e., $d = (d_{k-1}, \ldots, d_1, d_0)_b = \sum_{i=0}^{k-1} d_i \cdot b^i$, where $d_i \in [0, b-1]$ for $i \in \{0, \ldots, k-1\}$
2. C pseudo-randomly chooses $u_0, \ldots, u_{k-1} \in \mathbb{Z}_n^*$ using G on input a random seed s
   C sets $v_0 := (\prod_{i=0}^{k-1} u_i^{b^i})^{-1} \mod n$
3. C stores $((d_{k-1}, \ldots, d_1, d_0)_b, s, v_0)$ on its device.

Online phase instructions:

1. S sets $B_1 := c$ and computes $B_j := B_{j-1} \cdot c \mod n$, for $j = 2, \ldots, b-1$
   S sends $B_2, \ldots, B_{b-1}$
2. C sets $B_0 := 1$, $B_1 := c$, and computes $z_i := B_{d_i} \cdot u_i \mod n$ for $i = 0, \ldots k-1$
   C sends $z_0, \ldots, z_{k-1}$
3. S computes $w_0 := \prod_{i=0}^{k-1} z_i^{b^i} \mod n$ and S sends w_0
4. C **returns** $m := w_0 \cdot v_0 \mod n$,

Protocol Properties: The *efficiency* properties are verified by protocol inspection; in particular,

- *C's online runtime complexity* consists of $k+1$ multiplications, which improves over non-delegated multiplication by a multiplicative factor of $\geq \log b$;
- *S's runtime complexity* consists of 1 product of k exponentiations, and $b-2$ multiplications in $\mathbb{Z}_n^*$;
- the *offline runtime complexity* consists of 1 product of k exponentiations with random k bases, 1 inversion in $\mathbb{Z}_n^*$,and time linear in $|d|$;
- with respect to *round complexity,* $\mathcal{P}_3$ requires 3 messages between C and S; and
- with respect to *communication complexity*, the online phase of $\mathcal{P}_3$ requires the transfer of $b+k-1$ values in $\mathbb{Z}_n^*$.

The *correctness* property follows by showing that if C and S follow the protocol, C always outputs $m = c^d \mod n$ since $d = (d_{k-1}, \ldots, d_1, d_0)_b = \sum_{i=0}^{k-1} d_i \cdot b^i$ and

$$\begin{aligned} m &= w_0 \cdot v_0 = \prod_{i=0}^{k} z_i^{b^i} \cdot (\prod_{i=0}^{k} u_i^{b^i})^{-1} \\ &= \prod_{i=0}^{k} (B_{d_i} \cdot u_i)^{b^i} \cdot \prod_{i=0}^{k} u_i^{-b^i} = \prod_{i=0}^{k} (B_{d_i})^{b^i} \\ &= \prod_{i=0}^{k} (c^{d_i})^{b^i} = c^{\sum_{i=0}^{k} d_i \cdot b^i} = c^d \end{aligned}$$

The *privacy* property of the protocol against a malicious S follows by observing that C's message to S is a sequence of pseudo-random values in $\mathbb{Z}_n^*$ and is thus pseudo-independent from d. Specifically, the values sent by C to S are $z_0, \ldots, z_k$ where $z_i = B_{d_i} \cdot u_i \mod n$ for all $i = 0, \ldots, k-1$ and $z_k = m \cdot u_k \mod n$. Note that as $u_0 \ldots, u_k$ are chosen as pseudo-random values in $\mathbb{Z}_n^*$ using G's output on input a random seed s, even $z_0, \ldots, z_k$ are pseudo-random values in $\mathbb{Z}_n^*$. Thus, under the assumption that G is pseudo-

random, the probability that any efficient malicious S learns some additional information about private exponent d, is negligible.

Discussion. Note that the *result security* property can be achieved for this protocol by C delegates the computation of $c' = m^e \mod n (= w_1 \cdot v_1 \mod n)$ through steps 4 to 6 and checks that $c' = c$. The properties of the resulting protocol are summarized in Table 4, where they are also compared with a protocol presented in [25], which turn out to have somewhat similar properties.

This combination of protocols $\mathcal{P}_3$ and $\mathcal{P}_e$ comes close to solving the RSA decryption delegation problem, except that it still requires somewhat high communication complexity for typical parameter values of b, k. Similarly as noted at the end of Section 3.3, shorter vector-type representations of d have appeared in many past efforts, starting with [3], and none of them has been proved secure, as also discussed in Section 2.

4 Conclusions and Directions for Future Research

The problem of delegating the computation of RSA decryption has attracted a large amount of solution proposals and attacks for 30+ years, despite advances in delegation protocols for other computations of use in cryptographic protocols. Desirable requirements by a solution to this problem include correctness of ciphertext computation, privacy of RSA secret key, high probability detection of a malicious server trying to convince the client of an incorrect RSA ciphertext, efficient online phase runtime for both the client (i.e., significantly smaller than the best known non-delegated computation) and the server (i.e., not much worse than the best known non-delegated computation), low communication complexity, low client storage complexity, and minimum number of servers.

We presented 3 provable solutions satisfying all above requirements but one: a scheme requiring 2 non-colluding servers, and a scheme requiring high communication complexity. One open direction is to reduce the number of servers or reduce communication complexity in these 2 protocols. Another interesting direction is to extend the lower bounds from [22], and reviewed in Table 1, from prime-order groups, to RSA groups. Even if this latter problem were solved, we note that such lower bounds would not be ruled out the possibility of designing protocols with better communication complexity performance (or overall better performance tradeoffs) than those presented here.

We discussed that much research has been produced on attacks and variations of the protocols proposed in [3]. As many of the attacks considered an adversary model where the adversary obtains signatures with the same RSA secret key, one direction worth investigating is whether techniques in these protocols can be used to provably delegate the computation of RSA decryption, where no signatures would be available to the adversary.

References

[1] J. Feigenbaum, *Encrypting Problem Instances: Or ..., Can You Take Advantage of Someone Without Having to Trust Him?*, in Proc. of CRYPTO 1985: 477-488

[2] M. Abadi, J. Feigenbaum, J. Kilian, *On Hiding Information from an Oracle*, in J. Comput. Syst. Sci. 39(1): 21-50 (1989)

[3] T. Matsumoto, K. Kato, H. Imai, *Speeding Up Secret Computations with Insecure Auxiliary Devices.* In Proc. of CRYPTO 1988: 497-506

[4] S. Hohenberger, A. Lysyanskaya, *How to securely outsource cryptographic computations.* In Proc. of TCC 2005, pp. 264–282, Springer.

[5] R. Gennaro, C. Gentry, B. Parno, *Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers*, in Proc. of CRYPTO 2010: 465-482, Springer.

[6] B. Cavallo, G. Di Crescenzo, D. Kahrobaei, V. Shpilrain, *Efficient and secure delegation of group exponentiation to a single server.* In Proc. of RFIDSec Workshop, pp. 156-173, Springer, 2015.

[7] H. Wasserman, M. Blum: *Software reliability via run-time result-checking.* In J. ACM 44(6): 826-849, Proc. of IEEE FOCS 94, 2019.

[8] S. Galbraith, *Mathematics of Public-Key Cryptography*, Cambridge Press, 2018, version 2.0.

[9] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, V. Shpilrain, *Secure Delegation to a Single Malicious Server: Exponentiation in RSA-type Groups.* In Proc. of IEEE CNS 2019: 1-9.

[10] G. Horng, *A Secure Server-Aided RSA Signature Computation Protocol for Smart Cards*, In J. of Information Science and Engineering 16, 847-855 (2000).

[11] Y. Wang, Q. Wu, D. Wong, B. Qin, S. Chow, Z. Liu, X. Tan, *Securely Outsourcing Exponentiations with Single Untrusted Program for Cloud Storage*, in Proc. of ESORICS (1) 2014, LNCS 8712, Springer, pp. 326-343

[12] J. Ye and J. Wang, *Secure Outsourcing of Modular Exponentiation with Single Untrusted Server*, In 2015 18th Int. Conf. on Network-Based Information Systems, 2015.

[13] Y. Ding, Z. Xu, J. Ye, K.-K.R. Choo, *Secure outsourcing of modular exponentiations under single untrusted program model.* In J. Comput. Syst. Sci., 90, 1-13, 2017.

[14] A. Fu, S. Li, S. Yu, Y. Zhang, Y. Sun, *Privacy-preserving composite modular exponentiation outsourcing with optimal checkability in single untrusted cloud server.* In J. of Network and Comp. App., vol.118, pp. 102–112, 2018.

[15] A. Fu, Y. Zhu, G. Yang, S. Yu, Y. Yu, *Secure outsourcing algorithms of modular exponentiations with optimal checkability based on a single untrusted cloud server.* In Cluster Computing 21, pp. 1933–1947, 2018.

[16] Y. Ren, M. Dong, Z. Qian, X. Zhang, G. Feng, *Efficient Algorithm for Secure Outsourcing of Modular Exponentiation with Single Server*. In IEEE Transactions on Cloud Computing, 145–154, 2021

[17] J. Rangasamy, L. Kuppusamy, *Revisiting single-server algorithms for outsourcing modular exponentiation*, in Proc. of INDOCRYPT 2018, LNCS 11356, pp. 3-20.

[18] Q. Su, R. Zhang, and R. Xue, *Secure outsourcing algorithms for composite modular exponentiation based on single untrusted cloud*, in The Computer Journal, vol. 63, 2020

[19] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, V. Shpilrain, *Practical and Secure Outsourcing of Discrete Log Group Exponentiation to a Single Malicious Server*. In Proc. of 9th ACM CCSW, pp. 17–28, 2017.

[20] T. Mefenza and D. Vergnaud, *Cryptanalysis of Server-Aided RSA Protocols with Private-Key Splitting*, in The Computer Journal, 62(8): 1194-1213, 2019

[21] C. Bouillaguet, F. Martinez, and D. Vergnaud, *Cryptanalysis of Modular Exponentiation Outsourcing Protocols*, in The Computer Journal, in press.

[22] C. Chevalier, F. Laguillaumie, D. Vergnaud, *Privately Outsourcing Exponentiation to a Single Server: Cryptanalysis and Optimal Constructions*. Algorithmica 83, 72–115, 2021; also, Proc. ESORICS '16: 261-278, Springer.

[23] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, V. Shpilrain, *Efficient and Secure Delegation of Exponentiation in General Groups to a Single Malicious Server*. In Math. in Comput. Sci. 14(3): 641-656 (2020). Also in IMCS 2018.

[24] S. Canard, J. Devigne, and O. Sanders, *Delegating a pairing can be both secure and efficient*. In Proc. of ACNS 2014, LNCS vol 8479, Springer.

[25] S. Kawamura, A. Shimbo, *Fast server-aided secret computation protocols for modular exponentiation*. In IEEE Journal on Selected Areas in Communications, Vol. 11, No.5, pp. 778-784, 1993.

[26] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, V. Shpilrain, *Secure and Efficient Delegation of Elliptic-Curve Pairing*. In: Proc. of ACNS 2020. LNCS, vol. 12146. Springer, Cham.

[27] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, V. Shpilrain, *Secure and Efficient Delegation of Pairings with Online Inputs*. In: Proc. of CARDIS 2020. LNCS, vol. 12609. Springer.

[28] G. Di Crescenzo, M. Khodjaeva, V. Shpilrain, D. Kahrobaei, R. Krishnan, *Single-Server Delegation of Ring Multiplications from Quasilinear-time Clients*. In 14th Int. Conf. on Security of Information and Networks (SIN), 2021, pp. 1-8.

[29] G. Di Crescenzo, M. Khodjaeva, D. Kahrobaei, V. Shpilrain, *A Survey on Delegated Computation*. In Proc. of DLT 2022. LNCS, vol. 13257. Springer.