City University of New York (CUNY)
CUNY Academic Works

All Dissertations, Theses, and Capstone Projects | Dissertations, Theses, and Capstone Projects

9-2017

Secure and Efficient Delegation of a Single and Multiple Exponentiations to a Single Malicious Server

Matluba Khodjaeva
The Graduate Center, City University of New York

How does access to this work benefit you? Let us know!

More information about this work at: https://academicworks.cuny.edu/gc_etds/2395
Discover additional works at: https://academicworks.cuny.edu

This work is made publicly available by the City University of New York (CUNY).
Contact: AcademicWorks@cuny.edu

Secure and Efficient Delegation of a Single and Multiple Exponentiations to a Single Malicious Server

by

Matluba Khodjaeva

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

2017

©2017

Matluba Khodjaeva

All Rights Reserved

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirements for the degree of Doctor of Philosophy.

Date

Delaram Kahrobaei
Chair of Examining Committee

Date

Robert Haralick
Executive Officer

Delaram Kahrobaei

Giovanni Di Crescenzo

Vladimir Shpilrain

Xiaowen Zhang

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

Secure and Efficient Delegation of a Single and Multiple Exponentiations to a Single Malicious Server

by

Matluba Khodjaeva

Advisor: Delaram Kahrobaei

Group exponentiation is an important operation used in many cryptographic protocols, specifically public-key cryptosystems such as RSA, Diffie Hellman, ElGamal, etc. To expand the applicability of group exponentiation to computationally weaker devices, procedures were established by which to delegate this operation from a computationally weaker client to a computationally stronger server. However, solving this problem with a single, possibly malicious, server, has remained open since a formal cryptographic model was introduced [17]. Several later attempts either failed to achieve privacy or only achieved constant security probability.

In this dissertation, we study and solve this problem for discrete log type groups and RSA type groups for single and multiple (batch) exponentiations and apply our solution in several protocols. Each of our protocols satisfies natural correctness, security, privacy, and efficiency requirements, where security holds with exponentially small probability.

Acknowledgements

First of all, I would like to express my sincere gratitude to my advisor Professor Delaram Kahrobaei who gave me the spirit to continue to pursue my PhD degree in December 2015. She introduced me to researchers and collaborators, led me to present papers at seminars and posters at NYU conferences, preparing me for my Exams. She presented me with many interesting problems to solve, and provided me with both advice and guidance during the composition of my thesis. I would like to thank my co-advisor, Professor Giovanni Di Crescenzo, for the continuous support of my Ph.D study and related research, as well as for his patience, motivation, and immense knowledge. Professor Di Crescenzo supported me throughout my research studies and his guidance helped me in writing this thesis. I could not have imagined having a better advisor and co-advisor for my Ph.D study.

I thank Professor Vladimir Shpilrain for the interesting problems, helpful suggestions and solutions, as well as guidance throughout my studies. Also, I thank Professor Xiowen Zhang for giving the necessary feedback on my thesis drafts and the many great conversations during my examinations. I would like to thank all of my committee members

again for their useful comments and suggestions throughout my examinations. Without a doubt, they played a key role in achieving my educational dreams.

A special thank you goes to Professor Robert Haralick, who accepted me into the computer science program and encouraged me to complete my coursework. Moreover, I would like to thank Dilvania Rodriguez, who performes awesome service supporting the computer science department.

I would like to thank my family: my husband, parents, brother and sisters for supporting me spiritually throughout the writing of this thesis and my whole life in general.

Last but not least, I acknowledge the ONR (Office of Naval Research) grant N000141210-758 for financial support.

Sincerely,

Matluba Khodjaeva

September 8, 2017

Contents

List of Tables

Chapter 1

Introduction

Radio frequency identification or RFID is an automatic identification system which consists of two parts: the tags and the reader. A tag contains, in its memory, information about its identification (ID) and a reader can read the IDs of tags by running a link-layer protocol wirelessly when placed sufficiently close to the tag. There are two different kinds of RFID tags: passive and active. Passive tags do not have a battery and therefore their lifetime is practically unlimited. They are also inexpensive to manufacture (only a few cents). The reader energizes these tags with RF continuously while reading information from them. In comparison with active RFID tags, or tags that have an internal energy source, writing an efficient protocol for passive RFID tags is more challenging. One must address such issues as limited memory storage, power, and computational resources.

This research is about implementing security and privacy in passive RFID communication. Secure communication requires encryption of information before it is transmitted. Only the intended receiver should be able to decrypt the information. An eavesdropper should not be able to do the same. There are two categories of encryption schemes:

asymmetric key and symmetric key. The public key and asymmetric protocol is the most challenging to implement (among the two protocols) as it demands more storage space and power than its counterpart. There is a need to design strong cryptographic schemes for computationally weaker devices such as RFID tags. This problem is solved in this thesis given the constraints below:

- a computationally weaker client holds an input and description of the function
- a computationally stronger server holds the same description of the function

Given the above constraints, the solution satisfies:

- *correctness*: If both the client and the server are honest, the client obtains the output of the function from the server in response to the input delivered from client to server.
- *security*: the server cannot convince the client of a wrong computational output.
- *privacy*: the server cannot learn any information about the client's input.
- *efficiency*: the client's computation time is significantly shortened by delegating the process to the server rather than performing the computation without client-server interaction.

This thesis describes a protocol for interacting with a single, possibly malicious server, whereby the client delegates single or multiple exponentiations of a multiplicative group

element to be processed by the aforementioned server. The goal is to have the client make a small number of group multiplications rather than a non-delegated group exponentiation.

1.1 Construction

First and foremost, the aim is to write down rigorous definitions based on [17, 15] for the correctness, security, privacy and efficiency requirements for delegated computation protocols in a single server(possibly malicious) setting. In this model, we construct several protocols that satisfy these constraints, while delegating group exponentiation to a single malicious server. Specifically, we describe protocols for group exponentiation, in which the base of the exponent is a fixed constant and the power of the exponent is a variable (fixed-base, variable-exponent), and vise versa (variable-base, fixed-exponent). Then we extend this idea to multiple (batch) exponentiations. First, we write the protocol for group exponentiation for fixed-base variable-exponent, and then implement a similar protocol for fixed exponent variable base using the inverse delegation from Appendix A as discussed in [10]. Most of the protocols satisfy exponentially small security probability, which is a function of the security parameter (e.g. if security parameter = 128, then security probability = = 2^{-128}). We also introduce protocols for general, as well as specific, groups such as cyclic groups, RSA-type groups, and prime order-groups.

Before we begin discussing our protocols, we would like to talk about the pseudo-random power generator, which is used in an offline phase for the protocols described

in Chapter 3. Alternatively, we could consider a model with an offline phase, where a client can precompute exponentiations to random exponents using the square-and-multiply algorithm (discussed in Appendix B), or Yao's algorithm (discussed in [34]) or another party can precompute these exponentiations and store them on the client's device.

Protocols for general groups are introduced in Chapter 3. In Section 3.1, we describe an efficient protocol with security probability = 1/2 for group exponentiations of the fixed-exponent variable-base and variable-exponent fixed-base varieties. The main idea for these protocols comes from [10]. In Section 3.2, we use direct parallel repetition of the protocol in Section 3.1 and thereby achieve better security probability. Our next protocol, presented in Section 3.3, is a parameterized class of protocols where, for some values of the parameters, the security probability is reduced more efficiently than by direct parallel repetition. Both protocols achieve the following efficiency trade off: they reduce the number of the client's group multiplications during the online phase, while increasing the number of group exponentiations of random exponents during the offline phase as well as the number of the server's group exponentiations.

The privacy and security properties attributed to our protocols presuppose the existence of a pseudo random power generator but have no additional complexity assumptions, as the adversary corrupting the server is not limited to run in polynomial time. These protocols are written for the functions: $F_{G,exp,g}(x) = g^x$ (i.e. fixed-base variable-exponent exponentiation function) and $F_{G,exp,k}(x) = x^k$ (i.e. variable-base fixed-exponent exponentiation function) for general groups. Note that for the function $F_{G,exp,k}(x) = x^k$, we need

the delegation of group inverses protocol, written in Appendix A. Overall, the resulting protocols in this chapter have constant security probability as atomic protocols. Then, using parallel repetition of these protocols, we can achieve the security probability of the client equal to $2^{-\lambda}$ for statistical security parameter λ. Tables 1.1 and 1.2 describe the parameters for the protocols of Chapter 3.

Table 1.1: Comparison of parameters in protocols of Chapter 3 and in [10] for the function $F_{G,exp,k}$ in general group.

	[10]	3.2.2	3.3.2, $c = 2$	3.3.2, $c = 9$	3.4.2, $c = 2$	3.4.2, $c = 9$
Mod. Mult. (on)	5	640	5	5	325	145
Mod. Exp. (off)	2	256	2	9	130	261
Mod. Inverse	1	1	1	1	1	1
Security Prob.	$\frac{1}{2}$	$\frac{1}{2^{128}}$	$\approx \frac{1}{4}$	≈ 0.045	$\frac{1}{2^{128}}$	$\frac{1}{2^{128}}$

Table 1.2: Comparison of parameters in protocols of Chapter 3 and in [10] for the function $F_{G,exp,g}$ in general group.

	3.1.1	3.2.1	3.3.1, $c = 2$	3.3.1, $c = 9$	3.4.1, $c = 2$	3.4.1, $c = 9$
Mod. Mult.(on)	1	128	1	1	65	29
Mod. Exp.(off)	2	256	2	9	130	261
Security Prob.	$\frac{1}{2}$	$\frac{1}{2^{128}}$	$\approx \frac{1}{4}$	≈ 0.045	$\frac{1}{2^{128}}$	$\frac{1}{2^{128}}$

In Chapter 4, we show delegation of exponentiation in **cyclic group**s to a single, possibly malicious, server for the function fixed-base variable-exponent, $F_{G,exp,g}$. In Section 4.1 the client only performs a number of group multiplications in linear time i.e. $(= 2\lambda+3)$ and the security probability is $\epsilon_s = 2^{-\lambda}$. If we fix $\lambda = 128$ then the client's online multiplications $= 259$ and the security parameter is $\frac{1}{2^{128}}$. The protocol does not have any complexity assumptions, as the adversary corrupting the server is not limited to run in polynomial time. In addition, the protocol only requires 2 precomputed exponentiations

(in the offline phase) and the communication of 4 group elements, thus the schemes are of much greater interest for practical applications. In Section 4.2, we introduce the protocol which reduces the client's online group multiplication to 100, with the same security parameter as Section 4.1 only with an additional 3 modular exponentiation in the offline phase. Table 1.3 describes the parameters of the protocols in Chapter 4.

Table 1.3: Comparison of parameters in protocols of Chapter 3 and 4 for the function $F_{G,exp,g}$.

	4.1	4.2	3.4.1,$c = 2$	3.4.1, $c = 9$
Mod. Mult.(on)	259	100	65	29
Mod. Exp.(off)	2	5	130	261
Security Prob.	$\frac{1}{2^{128}}$	$\frac{1}{2^{128}}$	$\frac{1}{2^{128}}$	$\frac{1}{2^{128}}$

In Chapter 5, we show delegation of exponentiation in **RSA-type group**s to a single, possibly malicious, server for the function $F_{G,exp,k}$ fixed-exponent variable-base. In Section 5.1 the client only performs a number of group multiplications in linear time i.e. $(= 4\lambda+9)$ and security probability is $\epsilon_s = 2^{-\lambda}$. If we fix $\lambda = 128$ then the client performs 521 online multiplications and the security parameter is $\frac{1}{2^{128}}$. The protocol does not require any complexity assumptions, as the adversary corrupting the server is not limited to run in polynomial time. In addition, the protocol only requires 2 precomputed exponentiations in the offline phase and the communication of 4 group elements, thus the schemes are of much greater interest for practical applications. In Section 5.2, we introduce the protocol which reduces the client's online group multiplication to 198, instead of 521, with the same security parameter only with an additional 3 modular exponentiations in the offline

phase. Table 1.4 describes the parameters of the protocols in Chapter 5.

Table 1.4: Comparison of parameters in protocols of Chapter 3 and 4 for the function $F_{G,exp,k}$.

	5.1	5.2	3.4.2,$c = 2$	3.4.2, $c = 9$
Mod. Mult.(on)	521	198	325	145
Mod. Exp.(off)	2	5	130	261
Mod. Inverse	1	1	0	0
Security Prob.	$\frac{1}{2^{128}}$	$\frac{1}{2^{128}}$	$\frac{1}{2^{128}}$	$\frac{1}{2^{128}}$

In chapter 6, we show batch verification tests and the application of these tests to the delegation of client - server exponentiations over Discrete Log type groups. Then we show two client-server protocols for secure batch delegation to a single (and possibly malicious) server of exponentiations in the q-order subgroup G_q of $\mathbb{Z}_p^*$, where $p = 2q + 1$ and p, q are large primes. These groups are often used in cryptosystems that base their security on the hardness of the discrete logarithm problem or related problems. We show two batch verification of exponentiation tests in Section 6.2 and then we apply these two verification tests to two delegation of client server protocols in Section 6.3. Lastly, we show performance in Section 6.4.

In Chapter 7, we show batch verification tests and the application of these tests to the delegation of client - server exponentiations over the RSA type groups, more specifically in the group $\mathbb{Z}_N^*$, where $N = pq$ and p, q are large primes of the form $p = 2p_1 + 1, q = 2q_1 + 1$, for primes p_1, q_1. We proceed similarly to the previous chapter for secure batch verification test and delegation to a single (and possibly malicious) server of exponentiations in $\mathbb{Z}_N^*$. We show two batch verifications of exponentiation tests in Section 7.2 and then we apply

these two verification tests to two delegation of client server protocols in Section 7.3 then we show performance in Section 7.4.

1.2 Related Work

Secure outsourcing or delegating exponentiation has continuously been a popular topic. Many papers proposed new ideas, but then the ideas were broken in follow-up papers. Papers [15] and [23] write good solutions for the delegation of any polynomial circuit to a single (semi – honest in [23]) server using garbled circuits and fully-homomorphic encryption. The solutions given in those papers are asymptotically efficient, but are not efficient for low power devices such as passive RFID tags.

Papers [10, 17, 12, 13, 22, 9, 18] write solutions for both the client and server using the pseudo-random-powers generation assumption, which we also employ in Chapter 3. In addition, the hidden-subset-sum assumption is used in [22, 21] and [20] under the (stronger) subset sum hardness assumption.

In [10, 17], modular exponentiation is motivated by the utility in RFID tags. Specifically, in [17], the authors presented two kinds of protocols where (1) the client delegates to two non-colluding cloud servers, of which at most one is untrusted, and (2) the client delegates to one cloud server which is trusted (on average). In this protocol the client has to perform $O(\log^2 l)$ multiplications, where $|G| = l$, and the security probability is $1/2$, where the group has prime order. The follow up paper, [12], has similar features to [17] but improves the efficiency of the client and the security probability decreases to $1/3$, the

authors stated that "an interesting open problem is whether there is an efficient algorithm for secure outsourcing modular exponentiation using only one untrusted cloud server", which is addressed in this thesis. The next follow up paper, [9], with the same definition as in paper [17] tries to improve the security probability to $1/\lambda^2$ where λ is the bit length of the exponent for the variable-base, variable-exponent exponentiation function, but we found a contradiction in both described protocols (i.e. the defined privacy requirement does not hold because given a_1 and a_2, the server can learn information about the private input a). In Table 1.5 we make a comparison between our scheme in Chapter 4 and the schemes in [22, 12, 10].

Table 1.5: Comparing of outsourcing single exponentiation

	[22]	[12]	[10]	Chapter 4
Function	ver-base ver-exp	fix-base ver-exp	ver-base fix-exp	fix-base ver-exp
$Rand_{G,exp,g}$	7	5	2	2
Mod. Mult.	12	7	5	2
Mod. Exp.	1(λ - bit)	0	0	1 (λ - bit)
Mod. Inv.	4	3	0	0
Queries to S	4	6	3	2
Privacy	$\surd$	$\surd$	$\surd$	$\surd$
Security	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2^\lambda}$
# of S	Single server	Two servers	Single server	Single server

The authors in [13] provide secure and efficient, asymptotic speed up of about $O(0.24(\frac{l}{\log l})^{\frac{2}{3}})$ over the square and multiply algorithm by assuming a prime order group platform. In addition, the base and exponent are known to the server and the adversary, thus the privacy requirement is not applicable in this paper. The authors in paper [22] present a protocol to delegate variable-exponent, variable- base exponentiations to a single

untrusted server. In [10], the authors write protocols for delegation of group inverse and delegation of group exponentiation for fixed-exponent, variable-base exponentiations with constant security probability. In this proposal, we improve the security probability of [22] and extend the work to variable-exponent, fixed base exponentiations.

Batch cryptography was first discussed in [27]. The problems in batch cryptography are primarily concerned with performing multiple cryptographic protocol executions at lower costs than independent executions. The results in the literature can be characterized as focusing on verifying multiple outputs of an atomic function (the most studied being group exponentiation, since it is a cornerstone of many cryptographic protocols), as well as on application domains like operations (e.g. generation, verification) within encryption or signature schemes. A first set of formal treatments and solutions emerged from the works in [28, 29, 6, 26, 30]. In particular, batch verification algorithms have been provided for exponentiation over prime-order groups and digital signatures [6, 26, 30]. The state-of-the-art batch verifier for group exponentiation, named the Random Subset and the Small Exponents tests in [6], has been studied in [28, 29, 6, 26]. We show novel extensions of the Random Subset and the Small Exponents test to efficiently, privately and securely verify exponentiations over Discrete Log and RSA groups by a malicious server.

Chapter 2

Definitions

In this chapter we write the basic definitions of correctness, security, privacy and efficiency requirements for a delegation protocol (such as in [17] and [15]).

2.1 Basic notations

Let us define the

- $y \leftarrow T$: the probabilistic process of randomly and independently choosing y from the set T.
- $y \leftarrow A(x_1, x_2, \dots)$: y is the output of a (possibly probabilistic) algorithm A on inputs $x_1, x_2, \dots$ and any necessary random coins.
- $(z_A, z_B, tr) \leftarrow (A(x_1, x_2, \dots), B(y_1, y_2, \dots))$: z_A is the output from (possibly probabilistic) algorithm A on inputs $x_1, x_2, \dots$ and any necessary random coins. Similarly, z_B is the output of algorithm B, and tr is the sequence of messages exchanged by A and B.

2.2 System Scenario, Entities, and Protocol

Consider the system of parties, client and server, where the client's computational resources is more limited than the server, thus the client needs to delegate (outsource) the computation of a specific function to the servers. Let C be a single client and S be a single server in all of the outlined protocols.

Let σ denote the *computational security parameter* (i.e., the parameter derived from hardness considerations on the underlying computational problem), and let λ denote the *statistical security parameter* (i.e., a parameter such that events with probability $2^{-\lambda}$). Both parameters are expressed in unary notation (i.e., $1^{\sigma}, 1^{\lambda}$). For concreteness we use $\sigma = 2048$ and $\lambda = 128$, the currently recommended parameter settings in many cryptographic protocols and their applications.

Let $F : Dom(F) \rightarrow CoDom(F)$ be a function where $Dom(F)$ is the domain of F, $CoDom$ is the range of F and $desc(F)$ denotes the description of F. Let $(C(1^{\sigma}, 1^{\lambda}, desc(F), x_1, \ldots, x_n), S(1^{\sigma}, 1^{\lambda}, desc(F)))$ be the *client-server protocol for the delegated (n-instance) computation of* F, a two party communication protocol between C and S, where $desc(f)$ is known to both C and S, and the input x is known only to C. A *delegated computation of the value* $y_i = F(x_i)$ for $i = 1, \ldots, n$ is the execution, using independently chosen random bits for C and S, of the above client-server protocol and denoted as:

1. $pp \leftarrow C(1^{\sigma}, 1^{\lambda}, desc(F))$,

2. $((y_1, \ldots, y_n), tr) \leftarrow (C(1^\sigma, 1^\lambda, desc(F), x_1, \ldots, x_n), S(1^\sigma, 1^\lambda, desc(F))$

Case 1 is executed in an *offline phase*, when the inputs $x_1, \ldots, x_n$ to the function F are not yet available. If needed by the application, case 1 could be executed by a third party that does not collude with S. Case 2 is executed in the *online phase*, when the inputs $x_1, \ldots, x_n$ to the function F are available to C. At the end of both phases, C learns z_i (intended to be $y_i = F(x_i)$), and tr is the transcript of the communication between C and S. The delegated computation between client and server can be *sequential* (i.e. each execution begins after the previous one is finished) or *concurrent* (i.e. the server computes several executions at the same time from different clients). In this thesis we only consider sequential computation.

2.3 Correctness Requirements

Intuitively, the correctness requirement states that if the client and the server follow the protocol then at the end of the protocol the client receives some output which is equal to the n-tuple of values obtained by evaluating F on the client's m inputs.

First, consider the correctness Algorithm 1 below, denoted as $CorrExp_{F,A}(1^\sigma, 1^\lambda)$:

Definition. Let σ, λ be the security parameters, F be a function, and (C, S) be a client-server protocol for the delegated computation of F. We say that (C, S) satisfies δ_c-*correctness* if, for any x_i for $i = 1, \ldots, n$ in F's domain,

$$\Pr\left[out \leftarrow CorrExp_{F,A}(1^\sigma, 1^\lambda) : out = 1\right] \geq \delta_c$$

Algorithm 1 Correctness Algorithm: $CorrExp_{F,A}(1^\sigma, 1^\lambda)$

$pp \leftarrow C(desc(F))$
$(z_1, \ldots, z_m) \leftarrow (C(pp, x_1, \ldots, x_n), S)$
if $z_i = F(x_i)$ for all $i = 1, \ldots, n$ **then**
    return: 1
else
    return: 0
end if

for some δ_c close to 1.

2.4 Security Requirements

Intuitively, the basic security requirement states that if C follows the protocol, at the end of the protocol, a malicious adversary S cannot convince C to obtain output z_i such that $z_i \neq y_i$ where $y_i = F(x_i)$ and x_i is the input chosen by C for some $i = 1, \ldots, n$. For stronger and more realistic security we can extend the power of the adversary by letting him choose C's inputs $(x_1, \ldots, x_n)$ and execute the protocol polynomially many times, before trying to convince C to accept an incorrect output.

First, consider the security Algorithm 2 below, denoted as $SecExp_{F,A}(1^\sigma, 1^\lambda)$:

Algorithm 2 Security Algorithm: $SecExp_{F,A}(1^\sigma, 1^\lambda)$

$pp \leftarrow C(desc(F))$
$(x_1, \ldots, x_n, aux) \leftarrow A(desc(F))$
$(z_1, \ldots, z_n) \leftarrow (C(pp, x_1, \ldots, x_n), A(aux))$
if $z_i \in \{\perp, F(x_i)\}$ for all $i = 1, \ldots, n$ **then**
    return: 0
else
    return: 1
end if

Definition. Let σ, λ be the security parameters, F be a function, and (C, S) be a client-server protocol for the delegated computation of F. We say that (C, S) satisfies ϵ_s-**security against a malicious adversary** if, for any algorithm A,

$$\Pr\left[out \leftarrow SecExp_{F,A}(1^\sigma, 1^\lambda) : out = 1\right] \leq \epsilon_s$$

for some small ϵ_s.

2.5 Privacy Requirements

Intuitively, the basic privacy requirement states that if C follows the protocol, a malicious adversary corrupting S cannot get any information regarding C's input x from a protocol execution. In the cryptography literature privacy requirements have indistinguishable based approach, meaning the adversary can pick two inputs x_0, x_1 and one of these inputs is chosen randomly by C to be used in the protocol. Then, the adversary tries to guess which input C used in the protocol. For a stronger and more realistic privacy requirement we should extend the power of the adversary by letting him choose the input to C and then executing the protocol polynomially many times before attempting to guess C's input. We also consider a weaker privacy requirement, based on high residual entropy, where the protocol's transcript may leak some information about random inputs $x_1, \ldots, x_n$, but still leaves them undetermined among a very large set.

First, consider the privacy Algorithm 3 denote it as $PrivExp_{F,A}(1^\sigma, 1^\lambda)$:

Definition. Let σ, λ be the security parameters, F be a function, and (C, S) be a client-

Algorithm 3 Privacy Algorithm: $PrivExp_{F,A}(1^\sigma, 1^\lambda)$

$pp \leftarrow C(desc(F))$
$((x_{0,1} \ldots, x_{0,n}), (x_{1,1}, \ldots, x_{1,n}), aux) \leftarrow A(desc(F))$
$b \leftarrow \{0, 1\}$
$((z_1, \ldots, z_n), tr) \leftarrow (C(pp, (x_{b,1} \ldots, x_{b,n}), A(aux))$
$d \leftarrow A(aux)$
if $b = d$ **then**
    return: 1
else
    return: 0
end if

server protocol for the delegated computation of F. We say that (C, S) satisfies ϵ_p-**privacy (in the sense of indistinguishability) against a malicious adversary** if for any algorithm A,

$$\Pr\left[out \leftarrow PrivExp_{F,A}(1^\sigma, 1^\lambda) : out = 1\right] \leq \epsilon_p$$

for some small ϵ_p close to 0.

2.6 Efficiency Metrics and Requirements

Let (C, S) be a client-server protocol for the delegated computation of function F. We say that (C, S) has *efficiency parameters* $(t_F, t_P, t_C, t_S, cc, mc)$, where F can be computed using $t_F(\sigma, \lambda)$ atomic operations, C can run in the offline phase using $t_P(\sigma, \lambda)$ atomic operations and in the online phase using $t_C(\sigma, \lambda)$ atomic operations, S can be run $t_S(\sigma, \lambda)$ atomic operations, C and S exchange a total of at most mc messages, the length of those messages is cc. The atomic operation is considered as a group operation such as group

multiplication, inverse calculation, and/or exponentiation and are not considered atomic operations (e.g equality testing between two elements). In our analysis we only consider the most expensive group operations as atomic operations (e.g., group multiplications and/or exponentiation), and neglect lower-order operations (e.g., equality testing, additions and subtractions between group elements). The main goal is to have the protocols satisfy that $t_C(\sigma, \lambda)$ is smaller than $t_F(\sigma, \lambda)$ and $t_S(\sigma, \lambda)$ is not significantly larger than $t_F(\sigma, \lambda)$, with the following underlying assumptions:

1. group inverses require significantly more resources than group multiplication;

2. group exponentiation requires significantly more resources than group multiplication.

In addition, we want to minimize other protocol efficiency metrics, such as message complexity *mc* and communication complexity *cc*. We notice that according to the textbook for 'square-and-multiply' algorithm, $t_F(\sigma, \lambda)$ is, on average, equal to 1.5σ group multiplications.

Chapter 3

General Group Exponentiation

In this chapter we present protocols for delegation of exponentiation in general groups for the functions $F_{G,exp,g} : \mathbb{Z}_q \to G$, denoted as $F_{G,exp,x}(x) = g^x$, and $F_{G,exp,k} : G \to G$, denoted as $F_{G,exp,k}(x) = x^k$. In Section 3.1 we describe the basic protocol, which satisfies that the security parameter ϵ_s is equal to $1/2$ for both functions. In Section 3.2, we use the idea of *parallel repetition* of the protocol in Section 3.1 λ times, in order to improve the security probability to $2^{-\lambda}$. In the next protocol in Section 3.3, we introduce a class of protocols which depends on parameters c and m, where the security parameter ϵ_s reduces from $1/2$ to roughly $1/c$ without using parallel repetition. In Section 3.4 we use the protocol in Section 3.3, along with the method of parallel repetition, to get security probability $\epsilon_s = 2^{-\lambda}$. In these protocols we have the following conditions:

- the security probability, ϵ_s, decreases
- the client's group multiplication during the online phase decreases
- the client's group exponentiation during the offline phase increases

- the server's group exponentiation increases

The protocols' privacy and security properties use a pseudo random power generator assumption and no additional complexity assumptions. The server can be any malicious and server is not limited to run the protocol in polynomially many times.

3.1 Basic Delegating Exponentiation

This section describes the most efficient protocol with constant security probability for evaluating group exponentiations, specifically fixed-base variable exponent and fixed-exponent variable-base exponentiations. In further sections we write protocols to improve upon the security probability in this section.

3.1.1 Basic Delegating Exponentiation for Fixed-Base Variable-Exponent Group

In this section we write an efficient protocol for the function $F_{G,exp,g} : \mathbb{Z}_q \to G$, denoted as $F_{G,exp,x}(x) = g^x$, with constant security probability, specifically $\epsilon_s = 1/2$. First, we write the notation, theorem and a description of the protocol which satisfies the theorem for the function $F_{G,exp,g}$. Then, we prove the theorem by showing correctness, efficiency, privacy, and security requirements.

Notations: Let $l =: \lceil \log |G| \rceil$, the length of the binary representation of the element of group. We say group is *efficient* if its description is short, i.e. has length polynomial in l, its associated operation $*$ and inverse operation are efficient, i.e. they can be executed in time polynomial in l. Let σ be the security parameter and l be the group element

length where σ and l are typically the same value. Assume $(G, *)$ is an *efficient* group, and let $g \in G$ be an element with order $> 2q$ for some large integer q which is known to the client. Let $y = g^x$ denote the *exponentiation* of g to the x-th power; i.e. the value $y \in G$ such that $g * \cdots * g = y$, where the multiplication operation $*$ is applied $x - 1$ times. Let the function $F_{G,exp,g} : \mathbb{Z}_q \to G$ be denoted as $F_{G,exp,x}(x) = g^x$ where $x \in \mathbb{Z}_q = \{0, 1, \ldots, q-1\}$, $g \in G$ and $|g| > 2q$.

We say that $Rand_{G,g}$ is a *pseudo-random power generator* if it is a satisfiable, probabilistic polynomial-time algorithm with the following syntax and properties:

1. on input $i = 0$, $Rand_{G,g}$ returns an auxiliary state information aux;
2. on input $i > 0$ auxiliary state information aux, $Rand_{G,g}$ returns a pair (u_i, g^{u_i}), where $u_i \in G$ and update the state aux;
3. for any polynomial p, the tuple $\{(u_1, g^{u_1}), \ldots, (u_{p(\sigma)}, g^{u_{p(\sigma)}})\}$, obtained as part of the output of algorithm $Rand_{G,g}$, is computationally indistinguishable from the tuple $\{(z_1, g^{z_1}), \ldots, (z_{p(\sigma)}, g^{z_{p(\sigma)}})\}$, where $z_1, \ldots, z_{p(\sigma)}$ are random and independent elements from G.

A generator with these properties was first designed in [7], then refined in [21], and since has been used in a number of works, including previous work in outsourcing modular exponentiation [10]. This generator can be designed based on the hidden-subset-sum assumption in groups. The running time of $Rand_{G,k}$ is comparable to about m_r group multiplications where $m_r = O(\log^2 l)$.

Theorem 3.1.1. *Let σ be the security parameter, and let G be an efficient group. There exists (constructively) a client-server protocol (C, S) for delegated computation of the function $F_{G,exp,g}$ which satisfies*

1. *δ_c-correctness, for $\delta_c = 1$*
2. *ϵ_s-security, for $\epsilon_s = 1/2$*
3. *ϵ_p-privacy, for $\epsilon_p = 0$*
4. *efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where*
   - *t_F is $= 1$ group exponentiation in G*
   - *t_S is $= 2$ group exponentiations in G*
   - *t_P is $= 2$ group exponentiations with random exponents in G using pseudo random power generator ($\approx 2\log^2(l)$ group multiplications).*
   - *t_C is $= 1$ group multiplications in G*
   - *$cc = 2$ elements in G and 2 elements in $\mathbb{Z}_q$*
   - *$mc = 2$.*

We remark that Theorem 3.1.1 satisfies the constant security parameter requirement, i.e. $\epsilon_s = 1/2$. The protocol requirement only 1 group multiplication, in comparison to, the average number of group multiplications in the square-and-multiply algorithm discussed

in Appendix B (which performs 1.5σ multiplications, more specifically when $\sigma = 2048$, $1.5\sigma = 3072 >> 1$). Next we will show a protocol which satisfies Theorem 3.1.1.

Informal description of the protocol (C, S)**.** In our protocol we have both an offline phase and an online phase. In the offline phase, C generates, using pseudo random power generator, two pairs $(u_0, v_0), (u_1, v_1)$, where u_0, u_1 are random elements in $\mathbb{Z}_q$ and $v_i = g^{u_i} \mod q$ for $i = 0, 1$. In the online phase, one of these two pairs is used to verify that one of the two pairs sent by S is correct, and the other is used to mask C's input x and calculate $y = g^x$ with the help of S. Since S does not know which pair will be used by C for either of these two purposes, the security of this protocol is at most $1/2$, meaning that C can compute an incorrect output for the function $F_{G,exp,g}$ with probability at most $1/2$.

Formal description of the protocol (C, S)**.**

Input to S: 1^σ, $desc(F_{G,exp,g})$

Input to C: 1^σ, $desc(F_{G,exp,g})$, $g \in G$ and $x \in \mathbb{Z}_q$, $aux = Rand_{G,g}(0)$

Protocol instructions:

Offline phase instructions:

1. C computes $(u_i, v_i, aux) = Rand_{G,g}(i, aux)$, for $i = 0, 1$;

Online phase instructions:

1. C randomly chooses $b \in \{0, 1\}$;

   C sets $z_b := u_b,\ z_{1-b} := x - u_{1-b} \mod q$;

   C sends z_0, z_1 to S;

2. S computes $w_i := g^{z_i}$ for $i = 0, 1$;

   S sends w_0, w_1 to C

3. if $w_b \neq v_b$

   C **returns** $\perp$ and the protocol halts;

   C computes $y := w_{1-b} * v_{1-b}$ and **returns** y

Illustration of the Protocol (C, S)**:**

Client		**Server**
Offline Phase:		
$(u_i, v_i, aux) = Rand_{G,g}(i, aux)$ for $i = 0, 1$		
Online Phase:		
$b \in_R \{0, 1\}$		
$z_b := u_b,\ z_{1-b} := x - u_{1-b} \mod q$	$\xrightarrow{z_0, z_1}$	
	$\xleftarrow{w_0, w_1}$	$w_i = g^{z_i}$ for $i = 0, 1$
C checks if $w_b \stackrel{?}{=} v_b$ then		
If no, C **returns** $\perp$ and the protocol halts;		
Else C **returns** $y = w_{1-b} * v_{1-b}$		

Proof of Theorem 3.1.1:

Properties of protocol (C, S)**:**

1. *The efficiency properties* are verified by protocol inspection.

   - With respect to *round complexity:* only one round complexity
     - One message from C to S followed by one message from S to C (i.e. $mc = 2$.)
   - With respect to *communication complexity:* the protocol requires the transfer of 2 elements in $\mathbb{Z}_q$ and 2 elements in G (i.e. $z_0, z_1 \in \mathbb{Z}_q$ and $w_0, w_1 \in G$). Thus $cc = 4$.
   - With respect to *running time complexity:*
     - S runs 2 exponentiation operations.
     - C runs 2 offline group exponentiation, $t_P = 2$ using pseudo random generator and 1 online phase multiplication operation in G, i.e. $t_C = 1$.

2. *The correctness properties* are demonstrated by observing that if C and S follow the protocol, C's equality verification in step 3 will be satisfied, and thus C's output y is $\neq \perp$ and satisfies:

$$\begin{aligned} y =& w_{1-b} * v_{1-b} \\ =& g^{z_{1-b}} * g^{u_{1-b}} \\ =& g^{x-u_{1-b}} * g^{u_{1-b}} \\ =& g^{x} \end{aligned}$$

   which is $y = F_{G,exp,g}(x)$ for each $x \in \mathbb{Z}_q$.

3. *The privacy property* is obeyed by observing that the messages z_0, z_1 sent by C do not leak any information about $x \in \mathbb{Z}_q$. Notice that (u_0, u_1) is computationally indistinguishable from a pair of random elements in $\mathbb{Z}_q$, which complies with property 3 of the pseudo-random (G, g)-power generator. The pair (z_0, z_1) is also computationally indistinguishable because $z_b = u_b$ and $z_{1-b} = x - u_{1-b} \mod q$ for $b \in_R \{0, 1\}$. Therefore, messages z_0, z_1 sent by C do not leak any information about x.

4. To show that the *security property* is satisfied, we consider three cases for w_0, w_1 sent to the client from the server:

   (a) If both inequalities $w_0 \neq g^{z_0}$ and $w_1 \neq g^{z_1}$ hold.

   (b) If one of the two inequalities $w_0 \neq g^{z_0}$ and $w_1 \neq g^{z_1}$ hold.

   (c) If both equalities $w_0 = g^{z_0}$ and $w_1 = g^{z_1}$ holds.

   In case a) C returns $\perp$ because in step 3 of the protocol, C checks if $w_b \neq v_b$ then the protocol halts. Therefore, the adversary is successful in convincing C to return an output $\neq F_{G,exp,g}(x)$ with probability 0.

   In case b) either C returns $\perp$ in step 3, or C computes $y \neq F_{G,exp,g}(x)$. The goal of the adversary is to pass the check in step 3 and C computes $y \neq F_{G,exp,g}(x)$, and this occurs if the adversary correctly guesses b. This happens with probability $1/2$.

In case c) the server is honest. Therefore, the adversary is successful in convincing C to return an output $\neq F_{G,exp,g}(x)$ with probability 0.

Hence, the adversary is successful in convincing C to return an output $\neq F_{G,exp,g}(x)$ with probability at most 1/2. □

3.1.2 Basic Delegating Exponentiation for Fixed Exponent Variable Base Group

In this section we describe a similar protocol as in the previous subsection, for the function $F_{G,exp,k} : G \to G$ denoted as $F_{G,exp,k}(x) = x^k$, introduced in [10]. First we write the notation, theorem, and description of the protocol satisfying the theorem for the function $F_{G,exp,k}$. Then, we prove the theorem by showing correctness, efficiency, privacy, and security requirements.

Notations: Let $(G, *)$ be a commutative group, let $l = \lceil \log |G| \rceil$ and let $k > 0$ be an integer (assumed for simplicity $k < |G|$).

Let σ be a security parameter. Similarly as before, $Rand_{G,k}$ is a *pseudo-random power generator* if it satisfies probabilistic polynomial-time algorithm with the following syntax and properties:

1. on input $i = 0$, $Rand_{G,k}$ returns an auxiliary state information aux;
2. on input $i > 0$ auxiliary state information aux, $Rand_{G,k}$ returns a pair (u_i, u_i^k), where $u_i \in G$ and updates the state aux;
3. for any polynomial p, the tuple $\{(u_1, u_1^k), \ldots, (u_{p(\sigma)}, u_{p(\sigma)}^k)\}$, obtained as part of the

output of algorithm $Rand_{G,k}$, is computationally indistinguishable from the tuple $\{(z_1, z_1^k), \ldots, (z_{p(\sigma)}, z_{p(\sigma)}^k)\}$, where $z_1, \ldots, z_{p(\sigma)}$ are random and independent elements from G.

Now we write the formal theorem:

Theorem 3.1.2. *Let σ be a security parameter, let k be a positive integer and assume the existence of a pseudo-random (G, k)-powers generator. There exists (constructively) a client-server protocol (C, S) for delegated computation of function $F_{G,exp,k}$ which satisfies:*

1. *δ_c-correctness, for $\delta_c = 1$*
2. *ϵ_s-security, for $\epsilon_s = 1/2$*
3. *ϵ_p-privacy, for $\epsilon_p = 0$*
4. *efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where*
   - *t_F is $= 1$ group exponentiation in G;*
   - *t_S is $= 2$ group exponentiations and $= 1$ group inverse in G;*
   - *t_P is $= 2$ group exponentiations using pseudo random power generator in G;*
   - *t_C is $= 5$ group multiplications in G;*
   - *$cc = 6$ elements in G and $mc = 2$.*

We remark that Theorem 3.1.2 satisfies that the security requirement is constant for the function $F_{G,exp,k}$, i.e. $\epsilon_s = 1/2$ and C calculates only 5 group multiplications in

comparison to the average number of group multiplications in the square-and-multiply algorithm, discussed in Appendix B (which performs 1.5σ multiplications, more specifically when $\sigma = 2048$, $1.5\sigma = 3072 >> 5$). Next we will show the protocol which satisfies Theorem 3.1.2.

Informal description of protocol (C, S)**.** In this protocol, C uses the procedure $Rand_{G,k}$ to generate two pairs (u_0, v_0), (u_1, v_1) of random group elements u_0, u_1, and their k-th powers v_0, v_1, respectively. Then, one of these two pairs is used to verify that the answers from S are correct and the other pair is used to mask C's input x and allow C to compute a k-th power of x, using the answers received from S. Division is delegated to S by using one group multiplication and the inverse delegation from the protocol in Appendix A. The privacy property follows from the fact that the message sent by C to S is computationally indistinguishable from random elements in G with their k-th powers, in turn based on the properties of $Rand_{G,k}$, and thus leaks no information about x. The security property follows from the fact that the message sent by C to S does not reveal which of the two pairs of group elements is used for verification and which is used for computation. Therefore, any dishonest answer from S will be determined by C with probability at least $1/2$.

Formal description of protocol (C, S)**.** We will use Theorem 10 from Appendix A on (C_{inv}, S_{inv}) for delegated computation of inverses in group G. For instance, on input value x in G to be inverted, C_{inv} returns a group value d to send to S_{inv}; then on input d, S_{inv} returns a group value e to be sent to C_{inv}; finally based on x, d, e the algorithm

C_{inv} computes the value x^{-1}.

Also, let $Rand_{G,k}$ denote a pseudo-random (G, k)-powers generator. We assume that C computes $aux = Rand_{G,k}(0)$ once at set up time, before running any delegated computation protocol.

Input to S: $1^\sigma, 1^\lambda$, $desc(F_{G,exp,k})$

Input to C: $1^\sigma, 1^\lambda$, $desc(F_{G,exp,k}), x \in G, aux = Rand_{G,k}(0)$

Protocol instructions:

Offline phase instructions:

1. C computes $(u_i, v_i, aux) = Rand_{G,g}(i, aux)$, for $i = 0, 1$;

Online phase instructions:

1. C randomly chooses $b \in \{0, 1\}$;

   C sets $z_b = u_b,\ z_{1-b} = x * u_{1-b}$;

   C runs C_{inv} on input v_{1-b}, to obtain d;

   C sends z_0, z_1, d to S;

2. S computes $w_i = z_i^k$ for $i = 0, 1$;

   S runs S_{inv} on input d, thus obtains e;

   S sends w_0, w_1, e to C

3. C runs C_{inv} on input v_{1-b}, d, e to compute v_{1-b}^{-1};

   if this execution of (C_{inv}, S_{inv}) returns $\perp$ as output

C **returns** $\perp$ and the protocol halts;

if $w_b \neq v_b$ $(w_b = z_b^k = u_b^k = v_b^k)$

C **returns** $\perp$ and the protocol halts;

C computes $y = w_{1-b} * v_{1-b}^{-1}$ and **returns** y

Illustration of the Protocol (C, S)**:**

Client		**Server**
Offline Phase:		
$(u_i, v_i, aux) = Rand_{G,k}(i, aux)$ for $i = 0, 1$		
Online Phase:		
$b \in_R \{0, 1\}$		
$z_b := u_b,\ z_{1-b} := x * u_{1-b}$		
$C_{inv}(v_{1-b}) = d$ ($v_{1-b} * c = d,\ c \in_R G$)	$\xrightarrow{z_0, z_1, d}$	
		$w_i = z_i^k$ for $i = 0, 1$
	$\xleftarrow{w_0, w_1, e}$	$S_{inv}(d) = e$ $(d^{-1} = e)$
$C_{inv}(v_{1-b}, d, e) = v_{1-b}^{-1}$		
($c * e = c * d^{-1} = c * (v_{1-b} * c)^{-1} = v_{1-b}^{-1}$)		
if (C_{inv}, S_{inv}) returned $\perp$ as output		
C **returns** $\perp$ and protocol halts;		
if $w_b \neq v_b$ then		

C **returns** $\perp$ and protocol halts;

C computes $y = w_{1-b} * v_{1-b}^{-1}$and **returns** y

Proof of Theorem 3.1.2:

Properties of protocol (C, S)**:**

1. *The efficiency properties* are verified by protocol inspection.

   - With respect to *round complexity:* only one round complexity
     - One message from C to S followed by one message from S to C (i.e. $mc = 2$.)
   - With respect to *communication complexity:* the protocol requires the transfer of 6 elements (i.e. $cc = 6$)
   - With respect to *running time complexity:*
     - S runs 2 exponentiation operations and 1 inversion operation.
     - C runs 2 multiplication operations in G, and 1 execution of the inverse delegation protocol (requires 3 multiplications)
     - $t_P = 2$, because in the offline phase C executes the procedure $Rand_{G,k}$.

2. *The correctness properties* are demonstrated by observing that if C and S follow the protocol, C's equality verification in step 3 will be satisfied, and thus C's output y is $\neq\perp$ and satisfies:

$$y = w_{1-b} * v_{1-b}^{-1}$$

$$
\begin{aligned}
&= (z_{1-b})^k * ((u_{1-b})^k)^{-1} \\
&= (x * u_{1-b})^k * (u_{1-b})^{-k} \\
&= x^k * (u_{1-b})^k * (u_{1-b})^{-k} \qquad \text{(by commutativity)} \\
&= x^k
\end{aligned}
$$

which is $y = F_{G,exp}(x)$ for each $x \in G$.

3. *The privacy property* follows from the following two observations:

   (a) on a single execution of (C, S) the message z_0, z_1, d sent by C does not leak any information about x.

   (b) seeing multiple executions of (C, S) does not help the adversary to obtain information about the input x in a new execution, even when C's input is chosen by the adversary.

   To show the observation (a), first notice that (u_0, u_1) is computationally indistinguishable from a pair of random group elements by the property 3 of pseudo-random (G, k)-power generator. Thus, the pair (z_0, z_1) is also computationally indistinguishable, because $z_b = u_b$ and $z_{1-b} = x * u_{1-b}$ for $b \in_R \{0, 1\}$. In addition, the value d does not depend on x. Therefore, the message z_0, z_1, d sent by C does not leak any information about x. This follows from Theorem 3.1.1.

   To show observation (b), notice that the protocol (C, S) is one round complexity and if the protocol executed multiple times, it does not change the result.

4. The *security property* follows by combining the following two observations:

   (a) in each execution of (C, S), if S follows the protocol, then the equality $y = F_{G,exp,k}(x)$ holds for each $x \in G$;

   (b) seeing multiple executions of (C, S) does not help the adversary violate the equality $y = F_{G,exp,k}(x)$ in a future execution, even when C's input in these executions are chosen by adversary.

   To show the observation (a), first consider a single execution of (C, S), where C follows the protocol, and, for any probabilistic polynomial adversary corrupting S, consider the values w_0, w_1, e returned by the adversary to C.

   - The value e is associated with an execution of the inverse delegation protocol from Appendix A, which calculates the inverse of d and is secure by Theorem A.0.1. If the adversary deviates from the protocol in computing an e, C will detect the division and return the failure symbol $\perp$.

   - The value w_0, w_1, where $w_i = z_i^k$ for $i \in \{0, 1\}$. Since $v_b = u_b^k$, assuming property 3 of the pseudo-random (G, k)-powers generator, S cannot guess the random bit b, and the verification $w_b = v_b$ will be passed with probability at most 1/2. If $w_b \neq v_b$ then C returns the failure symbol.

     If $w_b = v_b$ then C returns $y = w_{1-b} * v_{1-b}^{-1}$ which is equal to x^k for each x by the correctness property.

To show observation (b), that multiple executions of (C, S) does not help the adversary to increase the probability of verification $w_b = v_b$ in the next execution by assuming property 3 of the pseudo-random (G, k)-powers generator. □

3.2 Delegating Exponentiation Using Parallel Repetition

In the last section we introduced a protocol which satisfies the security probability $\epsilon_s = 1/2$. In this section we improve the security probability ϵ_s using the idea of *parallel repetition* of the same protocol as seen in the previous section for both functions $F_{G,exp,g}$ and $F_{G,exp,k}$.

3.2.1 Delegating Exponentiation Using Parallel Repetition for Fixed-Base Variable-Exponent Group

In this sub-section, we want to decrease $\epsilon_s = 1/2$ to $\epsilon_s = 1/2^\lambda$ for a fixed security parameter λ for the function $F_{G,exp,g} : \mathbb{Z}_q \to G$, denoted as $F_{G,exp,g}(x) = g^x$. Now we extend the protocol (C, S) using the parallel repetition method. We will write the theorem and the satisfying protocol. Then we prove the theorem by showing correctness, efficiency, privacy, and security requirements.

Theorem 3.2.1. *Let σ, λ be security parameters, and let G be an efficient group. There exists (constructively) a client-server protocol (C, S) for delegated computation of the function $F_{G,exp,g}$ which satisfies*

1. *δ_c-correctness, for $\delta_c = 1$*

2. *ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$*

3. *ϵ_p-privacy, for $\epsilon_p = 0$*

4. *efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where*

   - *t_F is $= 1$ group exponentiation in G*
   - *t_S is $= 2\lambda$ group exponentiations in G*
   - *t_P is $= 2\lambda$ group exponentiations with random exponents in $\mathbb{Z}_q$ using pseudo random power generatorλ subtractions in $\mathbb{Z}_q$*
   - *t_C is $= \lambda$ group multiplications in G.*
   - *$cc = 2\lambda$ elements in G and 2λ elements in $\mathbb{Z}_q$*
   - *$mc = 2$.*

We remark that Theorem 3.2.1 satisfies a much better security probability, $\epsilon_s = 2^{-\lambda}$, than the previous section where the security probability is $\epsilon_s = 2^{-1}$. However, to get a low security probability the client needs to calculate more group multiplications, more specifically if λ gets very large the efficiency decreases because C has to calculate λ times more, than in Theorem 3.1.1. However, this is still better than using the square-and-multiply algorithm what discussed in Appendix B. More specifically, if $\lambda = 128, \sigma = 2048$, then C performs 256 group multiplications in Theorem 3.2.1, compared to $1.5\sigma = 3072$ multiplications using the square-and-multiply algorithm.

Informal description of the protocol (C, S)**.** In our protocol we have two phases, an offline phase and an online phase as in the previous section. In the offline phase, C generates 2λ pairs $(u_{i,0}, v_{i,0}), (u_{i,1}, v_{i,1})$ using pseudo random power generator, where $u_{i,0}, u_{i,1}$ are random elements in $\mathbb{Z}_q$ and $v_{i,j} = g^{u_i} \mod q$ for $i = 1, \ldots, \lambda$ and $j = 0, 1$. In the online phase, λ of these pairs is used to verify that λ of the 2λ pairs sent by S is correct. The other λ of these pairs is used to mask C's input x and calculate $y = g^x$ using the help of S. S does not know which pair will be used by C for any of these two purposes. Thus, security of this protocol is at most $2^{-\lambda}$, meaning that C can compute an incorrect output for the function $F_{G,exp,g}$ in this protocol with at most probability $2^{-\lambda}$.

Formal description of the protocol (C, S)**.**

Input to S: 1^σ, 1^λ, $desc(F_{G,exp,g})$, $g \in G$

Input to C: 1^σ, 1^λ, $desc(F_{G,exp,g})$, $g \in G$, $x \in \mathbb{Z}_q$, $aux = Rand_{G,g}(0)$

Protocol instructions:

Offline phase instructions:

1. C computes $(u_{i,j}, v_{i,j}, aux) = Rand_{G,g}(i_j, aux)$, for $i = 1, \ldots, \lambda$ and $j = 0, 1$;

Online phase instructions:

1. For each $i = 1, \ldots, \lambda$

   C randomly chooses $b_i \in \{0, 1\}$;

   C sets $z_{i,b_i} := u_{i,b_i}$, $z_{i,1-b_i} := x - u_{i,1-b_i} \mod q$;

   C sends $z_{i,0}, z_{i,1}$ to S;

2. For each $i = 1, \ldots, \lambda$

   S computes $w_{i,j} := g^{z_{i,j}}$ for $j = 0, 1$;

   S sends $w_{i,0}, w_{i,1}$ to C

3. For each $i = 1, \ldots, \lambda$,

   if $w_{i,b_i} \neq v_{i,b_i}$

   C **returns** $\perp$ and the protocol halts;

   C computes $y_i := w_{i,1-b_i} * v_{i,1-b_i}$

   if $y_1 = \cdots = y_\lambda$

   C **returns** y_1

   else C **returns** $\perp$

Illustration of the Protocol (C, S)**:**

Client **Server**

Offline Phase:

$(u_{i,j}, v_{i,j}, aux) = Rand_{G,g}(i_j, aux)$ for $i = 1, \ldots, \lambda$, $j = 0, 1$

Online Phase:

$b_i \in_R \{0, 1\}$

$z_{i,b_i} := u_{i,b_i}$, $z_{i,1-b_i} := x - u_{i,1-b_i} \mod q$

$\xrightarrow{z_{0,1},\cdots,z_{0,\lambda},z_{1,1},\cdots,z_{1,\lambda}}$ for $i = 1, \ldots, \lambda$, $j = 0, 1$

$\xleftarrow{w_{0,1},\cdots,w_{0,\lambda},w_{1,1},\cdots,w_{1,\lambda}}$ $w_{i,j} = g^{z_{i,j}}$

if $w_{i,b_i} \neq v_{i,b_i}$ for some $i = 1, \ldots, \lambda$ then

C **returns** $\perp$ and protocol halts;

C computes $y_i := w_{i,1-b_i} * v_{i,1-b_i}$ for $i = 1, \ldots, \lambda$

if $y_1 = \cdots = y_\lambda$ then

C **returns** y_1

else C **returns** $\perp$

Proof of Theorem 3.2.1:

Properties of protocol (C, S)**:**

1. *The efficiency properties* are verified by protocol inspection.

   - With respect to *round complexity:* only one round complexity
     - One message from C to S followed by one message from S to C (i.e. $mc = 2$.)
   - With respect to *communication complexity:* the protocol requires the transfer of 2λ elements in $\mathbb{Z}_q$ and 2λ elements in G (i.e. $z_{0,1}, \cdots, z_{0,\lambda}, z_{1,1}, \cdots, z_{1,\lambda} \in \mathbb{Z}_q$ and $w_{0,1}, \cdots, w_{0,\lambda}, w_{1,1}, \cdots, w_{1,\lambda} \in G$). Thus $cc = 4\lambda$.
   - With respect to *running time complexity:*
     - S runs 2λ exponentiation operations.
     - C runs 2λ offline phase group exponentiation, $t_P = 2\lambda$ using pseudo random generator. C runs λ online phase multiplication operations in G, $t_C = \lambda$.

2. *The correctness properties* are demonstrated by observing that if C and S follow the protocol, C's equality verification in step 3 will be satisfied, and thus C's output is $\neq \perp$ and satisfies:

$$\begin{aligned} y_1 =& w_{1,1-b} * v_{1,1-b} \\ =& g^{z_{1,1-b}} * g^{u_{1,1-b}} \\ =& g^{x-u_{1,1-b}} * g^{u_{1,1-b}} \\ =& g^x \end{aligned}$$

which is $= F_{G,exp,g}(x)$ for each $x \in \mathbb{Z}_q$.

3. *The privacy property* follows by observing that the messages $z_{0,1}, \cdots, z_{0,\lambda}$, $z_{1,1}, \cdots, z_{1,\lambda}$ sent by C do not leak any information about $x \in \mathbb{Z}_q$. Notice that $u_{0,1}, \cdots, u_{0,\lambda}$, $u_{1,1}, \cdots, u_{1,\lambda}$ are computationally indistinguishable from random elements in $\mathbb{Z}_q$, which follows by property 3 of pseudo-random (G, g)-power generator. Thus, it is true for the values $z_{0,1}, \cdots, z_{0,\lambda}, z_{1,1}, \cdots, z_{1,\lambda}$, because for all $i = 0, \ldots, \lambda$, C sets $z_{i,b_i} := u_{i,b_i}$ and $z_{i,1-b_i} := x - u_{i,1-b_i} \mod q$ for $b_i \in_R \{0, 1\}$. Therefore, the messages $z_{0,1}, \cdots, z_{0,\lambda}, z_{1,1}, \cdots, z_{1,\lambda}$ sent by C do not leak any information about x.

4. To show that the *security property* is satisfied, we consider three cases for the messages $w_{0,i}, w_{1,i}$ where $i = 1, \ldots, \lambda$, which S sent to C:

   (a) There exist $i \in \{1, ..., \lambda\}$ such that both inequalities $w_{i,0} \neq g^{z_{i,0}}$ and $w_{i,1} \neq g^{z_{i,1}}$

hold.

(b) For all $i \in \{1, ..., \lambda\}$ such that one of the inequalities, either $w_{i,0} \neq g^{z_{i,0}}$ or $w_{i,1} \neq g^{z_{i,1}}$, holds.

(c) For all $i \in \{1, ..., \lambda\}$ both equalities $w_{i,0} = g^{z_{i,0}}$ and $w_{i,1} = g^{z_{i,1}}$ hold.

In case a), C returns $\perp$ since in step 3 of the protocol above, C checks if $w_{i,b_i} \neq v_{i,b_i}$ then C halts the protocol. Therefore, the adversary is successful to convincing C to return an output $\neq F_{G,exp,g}(x)$ with probability 0.

In case b), either C returns $\perp$, or C computes $y \neq F_{G,exp,g}(x)$ as in step 3. The goal of the adversary is to pass the check in step 3, and C computes an incorrect output, i.e. $y \neq F_{G,exp,g}(x)$. This can happen if the adversary correctly guesses the values $b_i \in \{0, 1\}$, for all $i = 1, \ldots, \lambda$. This happens with probability $2^{-\lambda}$.

In case c), the server is honest. Therefore, the adversary is successful in convincing C to return an output $\neq F_{G,exp,g}(x)$ with probability 0.

Hence, the adversary is successful in convincing C to return an output $\neq F_{G,exp,g}(x)$ with probability at most $2^{-\lambda}$. □

3.2.2 Delegating Exponentiation Using Parallel Repetition for Fixed Exponent Variable Base Group

Similar to the previous subsection, we want to decrease the security probability $\epsilon_s = 1/2$ to $\epsilon_s = 1/2^{\lambda}$, for a fixed security parameter λ for the function $F_{G,exp,k} : G \to G$ denoted as $F_{G,exp,k}(x) = x^k$. Thus, we extend the protocol (C, S) using the parallel repetition method

in Section 3.1.2. We write the theorem and satisfying protocol then prove correctness, efficiency, privacy, and security requirements.

Theorem 3.2.2. *Let σ, λ be security parameters and let k be a positive integer. Assume the existence of a pseudo-random (G, k)-powers generator. There exists (constructively) a client-server protocol (C, S) for delegated computation of the function $F_{G,exp,k}$ which satisfies:*

1. *δ_c-correctness, for $\delta_c = 1$*
2. *ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$*
3. *ϵ_p-privacy, for $\epsilon_p = 0$*
4. *efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where*
    - *t_F is $= 1$ group exponentiation in G*
    - *t_S is $= 2\lambda$ group exponentiations in G*
    - *t_P is $= 2\lambda$ group exponentiations with random exponents in G using pseudo random power generator subtractions in $\mathbb{Z}_q$*
    - *t_C is $= 5\lambda$ group multiplications in G*
    - *$cc = 6\lambda$ elements in G and $mc = 2$.*

We remark that this Theorem 3.2.2 satisfies a significanlly better security probability, i.e. $\epsilon_s = 2^{-\lambda}$, than Theorem 3.1.2, where $\epsilon_s = 2^{-1}$. However, it requires more calculation

by C, specifically if λ gets very large, the efficiency decreases because C has to calculate 5λ multiplications rather than 5. But, this is still better than using the square-and-multiply algorithm which discussed in Appendix B. More specifically, in Theorem 3.2.2 if $\lambda = 128, \sigma = 2048$, then C performs 640 group multiplications, compared to $1.5\sigma = 3072$ multiplications using the square-and-multiply algorithm.

Informal description of the protocol (C, S)**.** In our protocol we have two phases, an offline phase and an online phase as in the previous section. In the offline phase, C generates 2λ pairs $(u_{i,0}, v_{i,0}) using pseudorandom power generator, (u_{i,1}, v_{i,1})$, where $u_{i,0}, u_{i,1}$ are random elements in G and $v_{i,j} = u_{i,j}^k$ for $i = 1, \ldots, \lambda$ and $j = 0, 1$. In the online phase, λ of these pairs is used to verify that λ of the 2λ pairs sent by S are correct. From the other, one pair is used to mask C's input x and calculate $y = x^k$ using the help of S. S does not know which pair will be used by C for any of these two purposes. Thus, the security of this protocol is at most $2^{-\lambda}$, meaning that C can compute an incorrect output for the function $F_{G,exp,k}$ in this protocol with probability at most $2^{-\lambda}$.

Formal description of the protocol (C, S)**.** We will use the Theorem in Appendix A on (C_{inv}, S_{inv}) for delegated computation of inverse in group G.

Input to S: 1^σ, 1^λ, $desc(F_{G,exp,k})$

Input to C: 1^σ, 1^λ, $desc(F_{G,exp,k})$, $x \in G$, $aux = Rand_{G,k}(0)$

Protocol instructions:

Offline phase instructions:

1. C computes $(u_{i,j}, v_{i,j}, aux) = Rand_{G,k}(i_j, aux)$, for $i = 1, \ldots, \lambda$ and $j = 0, 1$;

Online phase instructions:

1. For each $i = 1, \ldots, \lambda$

   C randomly chooses $b_i \in \{0, 1\}$;

   C sets $z_{i,b_i} := u_{i,b_i},\ z_{i,1-b_i} := x * u_{i,1-b_i}$;

   C runs C_{inv} on input $v_{i,1-b}$, thus obtaining d_i;

   C sends $z_{i,0}, z_{i,1}, d_i$ to S;

2. For each $i = 1, \ldots, \lambda$

   S computes $w_{i,j} := z_{i,j}^k$ for $j = 0, 1$;

   S runs S_{inv} on input d_i, thus obtains e_i;

   S sends $w_{i,0}, w_{i,1}, e_i$ to C

3. For each $i = 1, \ldots, \lambda$,

   C runs C_{inv} on input $v_{i,1-b_i}, d_i, e_i$ to compute $v_{i,1-b_i}^{-1}$;

   if this execution of (C_{inv}, S_{inv}) returned $\perp$ as output for some $i = 1, \ldots, \lambda$

      C **returns** $\perp$ and the protocol halts;

   if $w_{i,b_i} \neq v_{i,b_i}$

      C **returns** $\perp$ and the protocol halts;

   C computes $y_i := w_{i,1-b_i} * v_{i,1-b_i}^{-1}$

   if $y_1 = \cdots = y_\lambda$

C **returns** y_1

else C **returns** $\perp$

Illustration of the Protocol (C, S)**:**

Client **Server**

Offline Phase:

$(u_{i,j}, v_{i,j}, aux) = Rand_{G,k}(i_j, aux)$ for $i = 1, \ldots, \lambda$, $j = 0, 1$

Online Phase:

$b_i \in_R \{0, 1\}$

$z_{i,b_i} := u_{i,b_i}$, $z_{i,1-b_i} := x * u_{i,1-b_i}$

$C_{inv}(v_{i,1-b_i}) = d_i$ $\xrightarrow{z_{0,i}, z_{1,i}, d_i}$ for $i = 1, \ldots, \lambda$, and $j = 0, 1$

$w_{i,j} = z^k_{i,j}$

$\xleftarrow{w_{0,i}, w_{1,i}, e_i}$ $S_{inv}(d_i) = e_i$

$C_{inv}(v_{i,1-b_i}, d_i, e_i) = v^{-1}_{i,1-b_i}$

if (C_{inv}, S_{inv}) returned $\perp$ as output for some $i = 1, \ldots, \lambda$

C **returns** $\perp$ and protocol halts;

if $w_{i,b_i} \neq v_{i,b_i}$ for some $i = 1, \ldots, \lambda$ then

C **returns** $\perp$ and protocol halts;

C computes $y_i := w_{i,1-b_i} * v^{-1}_{i,1-b_i}$ for $i = 1, \ldots, \lambda$

if $y_1 = \cdots = y_\lambda$ then

C **returns** y_1

else C **returns** $\perp$

Proof of Theorem 3.2.2: Proof of this Theorem follows similarly as proof of Theorem 3.1.2 and Theorem 3.2.1. □

3.3 Delegating Exponentiation with Improved Security Error Reduction on Atomic Execution

In this section, we improve efficiency and security parameters which we showed in Section 3.1 and 3.2 by studying computation-efficient of reduction of the security parameter ϵ_s and efficiency parameter t_P and t_C for both functions $F_{G,exp,g}$ and $F_{G,exp,k}$. First, we introduce an atomic protocol with improved constant security parameter ϵ_s. Then, we show our atomic protocol which consists of a class of protocols depending two additional parameters.

3.3.1 Delegating Exponentiation with Improved Security Error Reduction on Atomic Execution for Fixed-Base Variable-Exponent Group

In this sub-section we work on the function $F_{G,exp,g} : \mathbb{Z}_q \to G$ denoted as $F_{G,exp,g}(x) = g^x$. We write the theorem and satisfying protocol. Then we prove the theorem by showing correctness, efficiency, privacy, and security requirements.

Theorem 3.3.1. *Let σ, λ be security parameters and c, m be protocol parameters. There exists (constructively) a client-server protocol (C, S) for delegated computation of function*

$F_{G,exp,g}$ which satisfies

1. *δ_c-correctness, for $\delta_c = 1$*

2. *ϵ_s-security, where ϵ_s is constant that depends on parameter c*

   (see Table 3.1 for exact values, ranging between 0.10763*, for* $c = 2, m = 100$*, to* 0.04080 *for* $c = 9, m = 100$*)*

3. *ϵ_p-privacy, for $\epsilon_p = 0$*

4. *efficiency with parameter $(t_P, t_F, t_S, t_P, , cc, mc)$, where*

   - *t_F is $= 1$ group exponentiation in G*
   - *t_S is $= m$ group exponentiations in G*
   - *t_P is $= c$ group exponentiations with random exponent in $\mathbb{Z}_q$ using pseudo random power generator*
   - *t_C is $= 1$ group multiplication in G*
   - *$cc = m$ elements in G and $= m$ elements in $\mathbb{Z}_q$*
   - *$mc = 2$*

We remark that Theorem 3.3.1 satisfies much better security probability in the atomic protocol when C only performs 1 group multiplication in G during the online phase of the protocol, than the atomic protocol in Theorem 3.1.1, where the security probability is $\epsilon_s = 2^{-1}$. However, the other metrics increase, specifically the number of group exponentiations

by S (i.e. t_S) and the number of pre-computed group exponentiations with random exponents using pseudo random power generator increase.

Informal description of protocol (C, S)**.** Comparing Sections 3.1 and 3.2, we notice that in Section 3.1 has security parameter $\epsilon_s = \frac{1}{2}$ and in Section 3.2 with the idea of *parallel repetition*, we run of the protocol in Section 3.1 λ times then check that all repetitions of the protocol returns the same output. If the values are equal, we know the protocol in Section 3.2 gives the security parameter $\epsilon_s = \frac{1}{2^\lambda}$. In this section we want to study a more computationally-efficient protocol for the client in order to reduce the security probability ϵ_s.

Before describing the protocol, we would like to write the inspiration for this protocol. First, we begin with the same protocol as in Section 3.1, the only changes are, that we include two random 'decoy' values, elements of $\mathbb{Z}_q$, in C's message in the online phase. Then, we shuffle these two 'decoy' elements and the two other elements described in Section 3.1, and send those four elements to S. Then the protocol follows similarly as in Section 3.1. In this way we obtain the following properties:

1. It does not increase C's number of multiplications (i.e. t_C)

2. It slightly increases S's number of computation (i.e. t_S)

3. The security parameter ϵ_s decreases from 1/2 to 1/3

In this case ϵ_s decreases from 1/2 to 1/3 without increasing C's number multiplications. Now, we want to generalize this idea of using m random decoy values in $\mathbb{Z}_q$. We will

show that the generalization of this idea to using m random decoy elements decreases the security parameter from $1/2$ to at most $1/7$, which may not be very satisfactory. In order to decrease the security parameter ϵ_s, we should increase the number of equality checks (in Section 3.1 we have only 1 equality check). In the following protocol we introduce the parameter c which represents an upper bound on the number of equality checks that C should execute. More specifically, in the resulting protocol, we have the following steps:

1. c number of pre-computing exponentiations that C calculates in the offline phase where
   - one value is used to compute the function output
   - $c-1$ values are used to perform equality checks
2. m values in $\mathbb{Z}_q$ sent by C to S where
   - $m-c$ values are decoy elements in $\mathbb{Z}_q$
   - c random values in $\mathbb{Z}_q$, generated in the offline phase using the pseudo random power generator, one of which is used to mask the input of C.
3. The resulting protocol achieves security parameter $\epsilon_s \approx \frac{1}{c}$

This protocol, which we describe below, is *the class* of protocols with varying parameters c and m. We analyze the security parameter, ϵ_s, for each of the values $c \in \{2, \ldots, 9\}$ and m.

Formal description of protocol (C, S)**.** We describe the client-server (C, S) protocol for the function $F_{G,exp,g} : \mathbb{Z}_q \to G$ denoted as $F_{G,exp,g}(x) = g^x$, where $g \in G$.

Input to S: 1^σ, $desc(F_{G,exp,g})$, $g \in G$, parameters $1^c, 1^m$

Input to C: 1^σ, $desc(F_{G,exp,g})$, $g \in G$ and $x \in \mathbb{Z}_q$, $aux = Rand_{G,g}(0)$ parameters $1^c, 1^m$

Protocol instructions:

Offline phase instructions:

1. C computes $(u_j, v_j, aux) = Rand_{G,g}(j, aux)$, for $j = 1, \ldots, c$;

Online phase instructions:

1. C randomly chooses distinct $j_1, \ldots, j_m \in \{1, \ldots, m\}$;

   C sets $z_{j_1} := u_1, \ldots, z_{j_{c-1}} := u_{c-1},\ z_{j_c} := x - u_c \mod q$;

   C randomly and independently chooses $z_{j_{c+1}}, \ldots, z_{j_m} \in \mathbb{Z}_q$ when $c < m$;

   C sends $z_1, \ldots, z_m$ to S;

2. S computes $w_j := g^{z_j}$ for $j = 1, \ldots, m$;

   S sends $w_1, \ldots, w_m$ to C

3. if $w_{j_1} \neq v_{j_1}$ or $w_{j_2} \neq v_{j_2}$ or $\ldots$ or $w_{j_{c-1}} \neq v_{j_{c-1}}$

   C **returns** $\perp$ and the protocol halts;

   C computes $y := w_{j_c} * v_{j_c}$ and **return:** y

Illustration of the Protocol (C, S)**:**

Client **Server**

Offline Phase:

$(u_j, v_j, aux) = Rand_{G,g}(j, aux)$ for $j = 1, \ldots, c$

Online Phase:

$j_1, \ldots, j_m \in_R \{1, \ldots, m\}$

$z_{j_1} := u_1, \ldots, z_{j_{c-1}} := u_{c-1},\ z_{j_c} := x - u_c \mod q$

$z_{j_{c+1}}, \ldots, z_{j_m} \in_R \mathbb{Z}_q$ "decoy elements" $\xrightarrow{z_1, \cdots, z_m}$ for $j = 1, \ldots, m$

$\xleftarrow{w_1, \cdots, w_m}$ $w_j := g^{z_j}$

if $w_{j_1} \neq v_{j_1}$ or $w_{j_2} \neq v_{j_2}$ or $\ldots$ or $w_{j_{c-1}} \neq v_{j_{c-1}}$ then

C **returns** $\perp$ and protocol halts;

C computes $y := w_{j_c} * v_{j_c}$

Proof of Theorem 3.3.1:

Properties of protocol (C, S)**:**

1. *The efficiency properties* are verified by protocol inspection.

   - With respect to *round complexity:* only one round complexity
     - One message from C to S followed by one message from S to C (i.e. $mc = 2$.)

- With respect to *communication complexity:* the protocol requires the transfer of m elements in $\mathbb{Z}_q$ and m elements in G (i.e. $z_m, \cdots, z_m \in \mathbb{Z}_q$ and $w_m, \cdots, w_m \in G$). Thus $cc = 2m$.
- With respect to *running time complexity:*
  - S runs m exponentiation operations.
  - C runs c offline phase group exponentiations, $t_P = c$, using pseudo random power generator and 1 online phase multiplication operation in G, $t_C = 1$.

2. *The correctness properties* are demonstrated by observing that if C and S follow the protocol, C's equality verification in step 3 will be satisfied, and thus C's output is $\neq \perp$ and satisfies:

$$\begin{aligned} y =& w_{j_c} * v_{j_c} \\ =& g^{z_{j_c}} * g^{u_{j_c}} \\ =& g^{x-u_{j_c}} * g^{u_{j_c}} \\ =& g^x \end{aligned}$$

   which is $y = F_{G,exp,g}(x)$ for each $x \in \mathbb{Z}_q$.

3. *The privacy property* follows by observing that the messages $z_1, \ldots, z_m$ sent by C do not leak any information about $x \in \mathbb{Z}_q$. Notice that $u_1, \cdots, u_c$ are computationally indistinguishable from random elements in $\mathbb{Z}_q$ by the property 3 of pseudo-random (G, g)-power generator. Thus, the values $z_1, \cdots, z_m$ are also indistinguishable from

elements in $\mathbb{Z}_q$, because C sets $z_{j_1} := u_1, \ldots, z_{j_{c-1}} := u_{c-1},\ z_{j_c} := x - u_c \mod q$ and C randomly and independently chooses $z_{j_{c+1}}, \ldots, z_{j_m} \in \mathbb{Z}_q$ when $c < m$. Therefore, the messages $z_1, \cdots, z_m$ sent by C do not leak any information about x.

4. To show the *security property* against any malicious S, we first write several properties and then write the proof for the security property with several cases of different values of c. In order to prove the security property, we need to find an upper bound for ϵ_s, where ϵ_s stands for the probability that S convinces C to output a y such that $y \neq F_{G,exp,g}(x)$. We define the following events on input $x \in \mathbb{Z}_q$ to C, for the protocol (C, S):

   - $e_{y \neq F}$, meaning that 'C's outputs y such that $y \neq F_{G,exp,g}(x)$'
   - $e_{y=F}$, meaning that 'C's outputs y such that $y = F_{G,exp,g}(x)$'
   - $e_{\perp}$, meaning that 'C's outputs $\perp$'

   Note that in the protocol (C, S), the events $e_{y \neq F}$, $e_{y=F}$, $e_{\perp}$ are mutually exclusive and one of them must occur at the end of the protocol. The following properties holds for those events:

   Observation 3.3.1. *Event $e_{y \neq F}$ holds if and only if event $(e_{y=F} \cup e_{\perp})^c$ (or using De morgan Law we can writing $(e_{y=F})^c \cap\ (e_{\perp})^c$) holds.*

   Recall that the message $(z_1, \ldots, z_m)$ is sent by C to S in step 1 during the online phase instruction and the message $(w_1, \ldots, w_m)$ is sent by S to C in step 2, where

S calculates $w_j := g^{z_j} (= F_{G,exp,g}(z_j))$. Let us define

- $nW_=$ as the set $\{j | w_j = F_{G,exp,g}(z_j)\}$, the number of equality for all w_js
- $nW_{\neq}$ as the set $\{j | w_j \neq F_{G,exp,g}(z_j)\}$, the number of inequality for all w_js.

Notice that:

(a) $(nW_=, nW_{\neq})$ is a partition of $\{1, \ldots, m\}$.

(b) C does **not** output $\perp$ whenever for all $j_1 \in nW_=, \ldots, j_{c-1} \in nW_=$, because C checks in last step if $w_{j_1} \neq v_{j_1}$ or $w_{j_2} \neq v_{j_2}$ or $\ldots$ or $w_{j_{c-1}} \neq v_{j_{c-1}}$ then returns $\perp$.

(c) C does **not** output $y = F_{G,exp,g}(x)$ whenever $j_c \in nW_{\neq}$.

The following observation holds as described above in mathematical terminology:

Observation 3.3.2. *It holds that:*

(a) $\Pr[(e_\perp)^c] = \Pr[j_1 \in nW_= \cap, \ldots, \cap j_{c-1} \in nW_=]$,

(b) $\Pr[(e_{y=F})^c] = \Pr[j_c \in nW_{\neq}]$

Since S can be malicious, let us observe the following:

- Could $nW_=$ be the empty set?

  If $nW_= = \emptyset$ then S sends to C incorrect w_i's for all $i = 1, \ldots, m$. But then the protocol halts by sending the failure symbol, $\perp$, since C's check in the last step.

- Could a malicious S choose the w_i as some arbitrary function of the message $(z_1, \ldots, z_m)$, as well as other public information?

  Observe that S **cannot** choose these messages because C randomly chooses $j_1, \ldots, j_m$ from the set of $\{1, \ldots, m\}$ in step 1 of the protocol. In other words, C "shuffles" all indices of z_j before sending values to S. So, S gets elements $(z_1, \ldots, z_m)$, which are uniformly and independently distributed in $\mathbb{Z}_q$. The following observation holds:

  Observation 3.3.3. *The distribution of values $w_1, \ldots, w_m \in G$ is independent from the distribution of values $j_1, \ldots, j_m \in \{1, \ldots, m\}$.*

  This observation is important while studying the strategies available S, this observation implies that S cannot compute w_i based on the random values $j_1, \ldots, j_m$ chosen by C in step 1.

In the remainder of the proof of security, we compute the upper bound for ϵ_s, meaning maximal probability for S to convince C of an incorrect output, i.e. the maximum probability of the event $e_{y \neq F}$. We prove the bound by proving in four different cases, where each case depends on a different value of c in the protocol (C, S).

Case 1. $c = 2$.

In this case

- C has to generate 2 precomputed group exponentiations with random

exponents in $\mathbb{Z}_q$ using the pseudo random power generator. To construct the message $z_1, \ldots, z_m$, one of the values is used to mask the input x in step 1 and the other value is used in the inequality check in step 3.

- C has to choose $m-2$ random decoy elements in $\mathbb{Z}_q$ to construct the message $z_1, \ldots, z_m$ in step 1 of the protocol.

We calculate the probability of the event $e_{y \neq F}$ as follows:

$$\begin{aligned}
\Pr[e_{y\neq F}] &= \Pr[(e_{y=F})^c \cap\ (e_\perp)^c] && \text{by Observation 1} \\
&= \Pr[(e_\perp)^c]\Pr[(e_{y=F})^c | (e_\perp)^c] && \text{by conditional probability} \\
&= \Pr[j_1 \in nW_=]\Pr[j_2 \in nW_{\neq} | j_1 \in nW_=] && \text{by Observation 2} \\
&= \frac{|nW_=|}{m}\frac{|nW_{\neq}|}{m-1} && \text{by Observation 3} \\
&= \frac{|nW_=|(m-|nW_=|)}{m(m-1)} && \text{by definitions of } nW_=, nW_{\neq}
\end{aligned}$$

Note that since S can be malicious, it can choose $(w_1, \ldots, w_m)$ arbitrarily. But we look for the maximum probability of the above calculation, thus we want to find the maximum probability of $\frac{|nW_=|(m-|nW_=|)}{m(m-1)}$. Let us relabel this probability as the function $f(j) := \frac{j(m-j)}{m(m-1)}$, and use calculus to find the maximum of this function.

- First take the derivative: $f'(j) = \frac{m-2j}{m(m-1)}$
- Set the derivative to zero (i.e. $f'(j) = \frac{m-2j}{m(m-1)} = 0$) in order to find the critical point. The critical point can occur at $j = \left\lceil \frac{m}{2} \right\rceil$ as well as $j = \left\lfloor \frac{m}{2} \right\rfloor$.

Note that we consider the floor or ceiling of $m/2$ because we need an integer value.

- We find that the function $f(j)$ has an absolute maximum at the point $j = \lceil \frac{m}{2} \rceil$ as well as $j = \lfloor \frac{m}{2} \rfloor$.

Therefore, the probability can reach the maximum when $|nW_{=}| = \lfloor \frac{m}{2} \rfloor$ (or $|nW_{=}| = \lceil \frac{m}{2} \rceil$), in which case we obtain that $\epsilon_s = \Pr[e_{y \neq F}] = \frac{m}{4(m-1)}$. Hence, as m increases ϵ_s gets closer to $1/4$.

Comparing to Theorem 3.1.1, we can conclude that by using $m-2$ random decoy elements in $\mathbb{Z}_q$ in C's message reduces the probability from $1/2$ to almost $1/4$, while requiring *no additional computation* of group multiplication from C.

Case 2. $c = 3$

In this case:

- C has to generate 3 precomputed group exponentiations with random exponents in $\mathbb{Z}_q$ using the pseudo random power generator to construct the message $z_1, \ldots, z_m$. One of these values is used to mask the input x in step 1, and the other two values are used to contact the two inequality checks in step 3.
- C has to choose $m-3$ random decoy elements in $\mathbb{Z}_q$ to creat the message $z_1, \ldots, z_m$ in step 1 of the protocol.

We calculate the probability of the event $e_{y\neq F}$ as follows:

$$\begin{aligned}
\Pr[e_{y\neq F}] &= \Pr[(e_{y=F})^c \cap \ (e_\perp)^c] \\
&= \Pr[(e_\perp)^c] \Pr[(e_{y=F})^c | (e_\perp)^c] \\
&= \Pr[j_1, j_2 \in nW_=] \Pr[j_3 \in nW_{\neq} | j_1, j_2 \in nW_=] \\
&= \Pr[j_1 \in nW_=] \Pr[j_2 \in nW_=] \Pr[j_3 \in nW_{\neq} | j_1 \in nW_=, j_2 \in nW_=] \\
&= \frac{|nW_=|}{m} \frac{(|nW_=| - 1)}{m-1} \frac{|nW_{\neq}|}{m-2} \\
&= \frac{|nW_=|(|nW_=| - 1)(m - |nW_=|)}{m(m-1)(m-2)} \\
&= \frac{(m - |nW_=|)}{(m-2)} \frac{\binom{|nW_=|}{2}}{\binom{m}{2}}
\end{aligned}$$

Note that since S can be malicious, it can choose $(w_1, \ldots, w_m)$ arbitrarily. We want to calculate the maximum of this probability. Thus, we find the maximum probability of $\frac{|nW_=|(|nW_=|-1)(m-|nW_=|)}{m(m-1)(m-2)}$. Let us relabel this fraction as a function

$$f(j) := \frac{j(j-1)(m-j)}{m(m-1)(m-2)} = \frac{(m-j)}{(m-2)} \frac{\binom{j}{2}}{\binom{m}{2}}$$

and calculus to find the maximum.

- First take the derivative: $f'(j) = \frac{2mj - 3j^2 - m + 2j}{m(m-1)(m-2)}$
- Set $f'(j)$ to zero (i.e. $f'(j) = \frac{m-2j}{m(m-1)} = 0$) in order to find the critical point. The critical point can happen at $j = \frac{m+1 \pm \sqrt{m^2 - m + 1}}{3}$.
- We find that the function $f(j)$ has an absolute maximum at the point $j = \frac{m+1+\sqrt{m^2-m+1}}{3}$.

Therefore, the probability can reach its maximum when

$|nW_{=}| = \frac{m+1+\sqrt{m^2-m+1}}{3}$, in which case we obtain that

$$\epsilon_s = \Pr[e_{y \neq F}] = 0.15041 < \frac{1}{6} \qquad \text{when } m = 100$$

$$\epsilon_s = \Pr[e_{y \neq F}] = 0.14837 \qquad \text{when } m = 1000$$

Hence, as m increases, ϵ_s gets closer to $1/7$.

Comparing this with Theorem 3.1.1, we can conclude that by using $m - 3$ random decoy elements in $\mathbb{Z}_q$ as part of C's message along with 1 additional precomputed exponentiation of a random exponent reduces the probability from $1/2$ to almost $1/7$, while requiring *no additional computation* of group multiplication from C in the online phase.

Case 3. $c = 4, 5, \ldots, m-1$

In this case

- C has to generate c precomputed group exponentiations with random exponents using the pseudo random power generator in $\mathbb{Z}_q$ to its message $z_1, \ldots, z_m$. One of the values is used to mask the input x in step 1 and the other $c-1$ are used to construct inequality checks in step 3.
- C has to choose $m - c$ random decoy elements in $\mathbb{Z}_q$ to its message $z_1, \ldots, z_m$ in step 1.

We calculate the probability of the event $e_{y \neq F}$ similarly as we did for $c = 2$

and $c = 3$ as follows:

$$\begin{aligned}
\Pr[e_{y\neq F}] &= \Pr[(e_{y=F})^c \cap\ (e_\perp)^c] \\
&= \Pr[(e_\perp)^c]\Pr[(e_{y=F})^c|(e_\perp)^c] \\
&= \Pr[j_1, j_2 \ldots, j_{c-1} \in nW_=]\Pr[j_c \in nW_{\neq}|j_1, \ldots, j_{c-1} \in nW_=] \\
&= \Pr[j_1 \in nW_=]\Pr[j_2 \in nW_=]\cdots\Pr[j_{c-1} \in nW_=]\Pr[j_c \in nW_{\neq}|j_1, \ldots, j_{c-1} \in nW_=] \\
&= \frac{|nW_=|}{m}\frac{(|nW_=|-1)}{m-1}\cdots\frac{(|nW_=|-(c-2))}{m-(c-2)}\frac{|nW_{\neq}|}{m-(c-1)} \\
&= \frac{|nW_=|}{m}\frac{(|nW_=|-1)}{m-1}\cdots\frac{(|nW_=|-(c-2))}{m-(c-2)}\frac{(m-|nW_=|)}{m-(c-1)} \\
&= \frac{(m-|nW_=|)}{(m-c+1)}\frac{\binom{|nW_=|}{c-1}}{\binom{m}{c-1}}
\end{aligned}$$

As in the previous cases, note that since S can be malicious, he can choose $(w_1, \ldots, w_m)$ arbitrarily. But we want to calculate maximum probability of the above calculation. Thus, we need to find the maximum of $\frac{(m-|nW_=|)}{(m-c+1)}\frac{\binom{|nW_=|}{c-1}}{\binom{m}{c-1}}$. Let us relabel this fraction as the function

$$\begin{aligned}
\epsilon_s &= \max_{c-1\leq j\leq m-1} f(j) \\
&= \max_{c-1\leq j\leq m-1} \frac{j}{m}\frac{(j-1)}{m-1}\cdots\frac{(j-(c-2))}{m-(c-2)}\frac{(m-j)}{m-(c-1)} \\
&= \max_{c-1\leq j\leq m-1} \frac{(m-j)}{(m-c+1)}\frac{\binom{j}{c-1}}{\binom{m}{c-1}}
\end{aligned}$$

To analyze the function $f(j)$

- using differentiation does not work with the high-degree polynomial.
- using the upper and lower bound of the binomial coefficient are too loose.

- using tool based trend analysis shows that this ratio approximately behaves like $O(\frac{1}{c})$ when studied as the function of c.

When we evaluated ϵ_s (the above equation) for all values of c numerically, we noticed

- improved efficiency on the number t_C (i.e. C's group multiplication during the protocol)
- number of t_P (i.e. C's group exponentiation with random exponent during offline phase) increases slightly
- by looking of all values of c, the obtained ϵ_s is smaller than what could be obtained by parallel repetition of $\lfloor \frac{c}{2} \rfloor$ executions of the atomic protocol from Section 3.1
- for the values $c = 4, 5, 6, 7, 8$ and 9 guarantee improved efficiency on t_C without making t_P much worse
- staring from $c = 10$, the dependency of this protocol for the value t_P gets larger than the one for the protocol in Section 3.2

The Table 3.1 reports ϵ_s when c and m change:

Note that when $c = 4, \ldots, 9$, the value ϵ_s in the Table 3.1 above is strictly smaller than the value $2^{\lfloor c/2 \rfloor}$ that could be obtained by using Section 3.2. However, when $c = 10$, the value $\epsilon_s = 0.03894 > .0312 = 2^{-5}$. Thus the protocol from Section 3.2 is better than using the above protocol. Thus, we are

Table 3.1: Values of ϵ_s for protocol (C, S) when $c = 2, \ldots, 10$ and different m

$c =$	2	3	4	5	6	7	8	9	10
$m = 10$	.27778	.17500	.13333	.11111	.10000	.10000	.10000	.10000	.10000
$m = 20$	.26316	.15965	.11739	.09391	.07982	.06842	.06316	.05789	.05263
$m = 40$	.25641	.15395	.11106	.08912	.07249	.06219	.05465	.04918	.04442
$m = 60$	.25424	.15196	.10912	.08551	.07053	.06024	.05117	.04692	.04232
$m = 80$	.25316	.15095	.10818	.08457	.06963	.05930	.05169	.04588	.04135
$m = 100$	.25252	.15041	.10763	.08403	.06906	.05874	.05117	.04538	.04080
$m = 1000$	.25025	.14837	.10568	.08212	.06718	.05685	.04928	.04350	.03894

only interested in this protocol when $c < 10$. We compared the two protocols numerically in Table 3.2 below.

Table 3.2: Comparison of this protocol for $m = 100$ and the protocol from Section 3.2. Let R be the number of repetitions of the same protocol.

Table 3.3: This Protocol

R	$c = t_P$	t_C	ϵ_s
1	2	1	$.\overline{25}$
1	4	1	.1076274
1	6	1	.067181234
1	8	1	.051172087
1	10	1	.040799532

Table 3.4: Protocol in Section 3.2

R	t_P	t_C	ϵ_s
1	2	1	.5
2	4	2	.25
3	6	3	.125
4	8	4	.0625
5	10	5	.03125

Comparing this protocol with Theorem 3.1.1, we can conclude that by using $m - c$ random elements (i.e. decoy elements) in $\mathbb{Z}_q$ as part of C's message and $c - 2$ additional precomputed exponentiations of a random exponent reduces the probability from 1/2 to approximately $O(\frac{1}{c})$, while requiring *no additional computation* of group multiplication from C.

Case 4. $c = m$

In this case

- C has to generate m precomputed group exponentiations with random exponents using the pseudo random power generator in $\mathbb{Z}_q$ to construct the message $z_1, \ldots, z_m$. One of these values is used to mask C's input x in step 1 and the another $m-1$ are used to construct $m-1$ inequality checks in step 3.
- C does not have to choose any random decoy elements in $\mathbb{Z}_q$ to its message $z_1, \ldots, z_m$ in step 1.

We calculate the probability of the event $e_{y \neq F}$ as above for $c = 2, \ldots, m-1$ as follows:

$$\begin{aligned}
\epsilon_s = \Pr[e_{y \neq F}] &= \Pr[(e_{y=F})^c \cap \ (e_\perp)^c] \\
&= \Pr[(e_\perp)^c] \Pr[(e_{y=F})^c | (e_\perp)^c] \\
&= \Pr[j_1, \cdots, j_{m-1} \in nW_=] \Pr[j_m \in nW_{\neq} | j_1, \cdots, j_{m-1} \in nW_=] \\
&= 1 \cdot \frac{1}{m} = \frac{1}{m}
\end{aligned}$$

However, the protocol requires m precomputed exponentiations of random exponents which is higher than the cost incurred by the protocol in Section 3.2, in order to reach the same security probability.

Therefore, we complete the proof of correctness, efficiency, privacy and security properties in this protocol. □

3.3.2 Delegating Exponentiation with Improved Security Error Reduction an Atomic Execution for Fixed-Exponent Variable-Base Group

This subsection is similar to the previous subsection, the difference is we use the function $F_{G,exp,k} : G \to G$ denoted as $F_{G,exp,k}(x) = x^k$. We will write the theorem and protocol which satisfies the theorem.

Theorem 3.3.2. *Let σ, λ be security parameters, c, m be protocol parameters and let k be a positive integer and assume the exitance of a pseudo-random (G, k)-powers generator. There exists (constructively) a client-server protocol (C, S) for delegated computation of function $F_{G,exp,k}$ which satisfies*

1. *δ_c-correctness, for $\delta_c = 1$*

2. *ϵ_s-security, where ϵ_s is constant that depends on parameter c*

   (see Table 3.1 for exact values, ranging between 0.10763, for $c = 2, m = 100$, to 0.04080 for $c = 9, m = 100$)

3. *ϵ_p-privacy, for $\epsilon_p = 0$*

4. *efficiency with parameter $(t_P, t_F, t_S, t_P, , cc, mc)$, where*

   - *t_F is $= 1$ group exponentiation in G*
   - *t_S is $= m$ group exponentiations in G and $= 1$ inversion in G*
   - *t_P is $= c$ group exponentiations using $Rand_{G,k}$.*

- t_C *is* $= 5$ *group multiplication in* G
- $cc = 2m + 2$ *elements in* G *and* $mc = 2$

We remark that Theorem 3.3.2 satisfies much better security probability achievement in an atomic protocol on the function $F_{G,exp,k} : G \to G$ denoted as $F_{G,exp,k}(x) = x^k$. C only performs 5 group multiplications in G during the protocol, compared to the atomic protocol in Theorem 3.1.2, where the security probability is $\epsilon_s = 2^{-1}$. However, the other metrics increase, specifically the number of group exponentiations by S (i.e. t_S) and the number of pre-computed group exponentiations with random exponents using the pseudo random power generator. In addition, there would be described in the protocol the existence of other comparisons for specific values of parameter c as in previous subsection.

Formal description of protocol (C, S)**.** We describe the client- server (C, S) protocol for the function $F_{G,exp,k}$. We will use Theorem A.0.1 in Appendix A on (C_{inv}, S_{inv}) for delegated computation of inverses in the group G.

Input to S: 1^σ, $desc(F_{G,exp,k})$, parameters $1^c, 1^m$

Input to C: 1^σ, $desc(F_{G,exp,k})$, $x \in G$, $aux = Rand_{G,k}(0)$ parameters $1^c, 1^m$

Protocol instructions:

Offline phase instructions:

1. C computes $(u_j, v_j, aux) = Rand_{G,k}(j, aux)$, for $j = 1, \ldots, c$;

Online phase instructions:

1. C randomly chooses distinct $j_1, \ldots, j_m \in \{1, \ldots, m\}$;

   C sets $z_{j_1} := u_1, \ldots, z_{j_{c-1}} := u_{c-1},\ z_{j_c} := x * u_c$;

   C randomly and independently chooses $z_{j_{c+1}}, \ldots, z_{j_m} \in G$ when $c < m$;

   C runs C_{inv} on input v_c, thus obtaining d;

   C sends $z_1, \ldots, z_m, d$ to S;

2. S computes $w_j := z_j^e$ for $j = 1, \ldots, m$;

   S runs S_{inv} on input d, thus obtains e;

   S sends $w_1, \ldots, w_m, e$ to C;

3. C runs C_{inv} on input v_c, d, e to compute v_c^{-1};

   if this execution of (C_{inv}, S_{inv}) returned $\perp$ as output

   C **returns** $\perp$ and the protocol halts;

   if $w_{j_1} \neq v_{j_1}$ or $w_{j_2} \neq v_{j_2}$ or $\ldots$ or $w_{j_{c-1}} \neq v_{j_{c-1}}$

   C **returns** $\perp$ and the protocol halts;

   C computes $y := w_{j_c} * v_{j_c}$ and **return:** y

Illustration of the Protocol (C, S)**:**

Client	**Server**
Offline Phase:	
$(u_j, v_j, aux) = Rand_{G,k}(j, aux)$ for $j = 1, \ldots, c$	
Online Phase:	

$j_1, \ldots, j_m \in_R \{1, \ldots, m\}$

$z_{j_1} := u_1, \ldots, z_{j_{c-1}} := u_{c-1},\ z_{j_c} := x * u_c$

$z_{j_{c+1}}, \ldots, z_{j_m} \in_R G$ "decoy elements"

$C_{inv}(v_c) = d$ $\xrightarrow{z_1, \cdots, z_m, d}$ $w_j := z_j^k$ for $j = 1, \ldots, m$

$\xleftarrow{w_1, \cdots, w_m, e}$ $S_{inv}(d) = e$

$C_{inv}(v_c, d, e) = v_c^{-1}$

if (C_{inv}, S_{inv}) returned $\perp$ as output

C **returns** $\perp$ and protocol halts;

if $w_{j_1} \neq v_{j_1}$ or $w_{j_2} \neq v_{j_2}$ or $\ldots$ or $w_{j_{c-1}} \neq v_{j_{c-1}}$

C **returns** $\perp$ and protocol halts;

C computes $y := w_{j_c} * v_{j_c}^{-1}$

Proof of Theorem 3.3.2: Proof follows similarly as the proof of Theorem 5 □

3.4 Delegating Exponentiation with Improved Security Error Reduction with Multiple Execution

In this section we decrease the security probability to some $\epsilon_s = 2^{-\lambda}$ while requiring a small number of group multiplications by the client using previous section. We can make a protocol similar to the previous section, using the idea of parallel repetition of Section 3.2. In this case, we can a get security probability of $\epsilon_s = 2^{-\lambda}$ and not too many group multiplications from C (i.e. t_C), but we require more precomputed exponentiation of

random exponent t_P using the pseudo random power generator.

Let us define the client-server protocol as (C_R, S_R) which stands for parallel repetition of R executions of client-server protocol (C, S) in Section 3.3.

3.4.1 Delegating Exponentiation with Improved Security Error Reduction with Multiple Execution for Fixed-Base Variable Exponent Group

In this subsection, we work on the function $F_{G,exp,g} : \mathbb{Z}_q \to G$, denoted as $F_{G,exp,g}(x) = g^x$. We will write the theorem and the protocol which satisfies this theorem. Then, we prove the theorem by showing correctness, efficiency, privacy, and security requirements.

Theorem 3.4.1. *Let σ, λ be security parameters and c, m be protocol parameters. There exists (constructively) a client-server protocol (C_R, S_R) for delegated computation of function $F_{G,exp,g}$ which satisfies*

1. *δ_c-correctness, for $\delta_c = 1$*
2. *ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$*
3. *ϵ_p-privacy, for $\epsilon_p = 0$*
4. *efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where $R = \left\lceil \frac{\lambda}{\log_2(\frac{1}{\epsilon_s})} \right\rceil$ and*
   - *t_F is $= 1$ group exponentiation in G*
   - *t_S is $= m \cdot R$ group exponentiations in G*
   - *t_P is $= c \cdot R$ group exponentiations with random exponents in $\mathbb{Z}_q$*

- t_C *is* $= R$ *group multiplications in* G
- $cc = m \cdot R$ *elements in* G *and* $m \cdot R$ *elements in* $\mathbb{Z}_q$
- $mc = 2$.

Formal description of the protocol (C, S)**.**

Input to S*:* 1^σ, 1^λ, $desc(F_{G,exp,g})$, $g \in G$

Input to C*:* 1^σ, 1^λ, $desc(F_{G,exp,g})$, $g \in G$, $x \in \mathbb{Z}_q$, $aux = Rand_{G,g}(0)$

Protocol instructions:

Offline phase instructions:

1. C computes $(u_{i,j}, v_{i,j}, aux) = Rand_{G,g}(i_j, aux)$, for $i = 1, \ldots, \lambda$ and $j = 1, \ldots, c$;

Online phase instructions:

1. For each $i = 1, \ldots, \lambda$

   C randomly chooses distinct $j_1, \ldots, j_m \in \{1, \ldots, m\}$;

   C sets $z_{i,j_1} := u_{i,1}, \ldots, z_{i,j_{c-1}} := u_{i,c-1},\ z_{i,j_c} := x * u_{i,c}$;

   C randomly and independently chooses $z_{i,j_{c+1}}, \ldots, z_{i,j_m} \in G$ when $c < m$;

   C sends $z_{i,1}, \ldots, z_{i,m}$ to S;

2. For each $i = 1, \ldots, \lambda$

   S computes $w_{i,j} := g^{z_{i,j}}$ for $j = 1, \ldots, m$;

   S sends $w_{i,1}, \ldots, w_{i,m}$ to C

3. For each $i = 1, \ldots, \lambda$

   if $w_{i,j_1} \neq v_{i,j_1}$ or $w_{i,j_2} \neq v_{i,j_2}$ or $\ldots$ or $w_{i,j_{-1}} \neq v_{i,j_{c-1}}$

   C **returns** $\perp$ and the protocol halts;

   C computes $y_i := w_{i,j_c} * v_{i,j_c}$

   if $y_1 = \cdots = y_\lambda$

   C **returns** y_1

   else C **returns** $\perp$

Illustration of the Protocol (C, S)**:**

Let $i = 1, \ldots, \lambda$

Client **Server**

Offline Phase:

$(u_{i,j}, v_{i,j}, aux) = Rand_{G,g}(i_j, aux)$ for $j = 1, \ldots, c$

Online Phase:

$j_1, \ldots, j_m \in_R \{1, \ldots, m\}$

$z_{i,j_1} := u_{i,1}, \ldots, z_{i,j_{c-1}} := u_{i,c-1},\ z_{i,j_c} := x - u_{i,c} \mod q$

$z_{i,j_{c+1}}, \ldots, z_{i,j_m} \in_R G$ random elements $\xrightarrow{z_{i,1}, \cdots, z_{i,m}}$

$\xleftarrow{w_{i,1}, \cdots, w_{i,m}}$ $w_{i,j} := g^{z_{i,j}}$ for $j = 1, \ldots, m$

if $w_{i,j_1} \neq v_{i,j_1}$ or $w_{i,j_2} \neq v_{i,j_2}$ or $\ldots$ or $w_{i,j_{c-1}} \neq v_{i,j_{c-1}}$ then

C **returns** $\perp$ and protocol halts;

C computes $y_i := w_{i,j_c} * v_{i,j_c}$

if $y_1 = \cdots = y_\lambda$ then

C **returns** y_1

else C **returns** $\perp$

Proof of Theorem 3.4.1: Properties of the protocol is similar to the proof of Theorem 3.1.2 and Theorem 3.2.1. □

We report numerical evaluations of the main parameters of our protocol for $c = 2$ applied in Section 3.2, and for $c = 2, 3, 4, 5, 6, 7, 8, 9$ for this section in the following Tables 3.5-3.7. We write three similar Tables for parameter m, specifically for $m = 10, 40, 100$. As you may notice, when m increases, the number of repetitions decrease for $\epsilon_s = 2^{-128}$, which is a fixed number. Number of repetitions is dependent on ϵ_s for atomic operations, given in Table 3.1 in the previous section.

Table 3.5: Efficiency parameters of (C_R, S_R) when $m = 10$ and $c = 2, \ldots, 9$.

	R	ϵ_s	t_P	t_C	t_S
$c = 2$ (Sec.3.2)	128	2^{-128}	256	128	256 exponentiations
$c = 2$ (Sec.3.4)	70	2^{-128}	140	70	$70m$ exponentiations
$c = 3$	51	2^{-128}	153	51	$51m$ exponentiations
$c = 4$	45	2^{-128}	180	45	$45m$ exponentiations
$c = 5$	41	2^{-128}	205	41	$41m$ exponentiations
$c = 6$	39	2^{-128}	234	39	$39m$ exponentiations
$c = 7$	39	2^{-128}	273	39	$39m$ exponentiations
$c = 8$	39	2^{-128}	312	39	$39m$ exponentiations
$c = 9$	39	2^{-128}	351	39	$39m$ exponentiations

In these Tables we include only $c = 2, 3, 4, 5, 6, 7, 8, 9$. We could calculate for $c =$

Table 3.6: Efficiency parameters of (C_R, S_R) when $m = 40$ and $c = 2, \ldots, 9$.

	r	ϵ_s	t_P	t_C	t_S
$c = 2$ (Sec.3.2)	128	2^{-128}	256	128	256 exponentiations
$c = 2$ (Sec.3.4)	66	2^{-128}	132	66	$66m$ exponentiations
$c = 3$	48	2^{-128}	144	48	$48m$ exponentiations
$c = 4$	41	2^{-128}	164	41	$41m$ exponentiations
$c = 5$	36	2^{-128}	180	36	$36m$ exponentiations
$c = 6$	34	2^{-128}	204	34	$34m$ exponentiations
$c = 7$	32	2^{-128}	224	32	$32m$ exponentiations
$c = 8$	31	2^{-128}	248	31	$31m$ exponentiations
$c = 9$	30	2^{-128}	270	30	$30m$ exponentiations

Table 3.7: Efficiency parameters of (C_R, S_R) when $m = 100$ and $c = 2, \ldots, 9$.

	r	ϵ_s	t_P	t_C	t_S
$c = 2$ (Sec.3.2)	128	2^{-128}	256	128	256 exponentiations
$c = 2$ (Sec.3.4)	65	2^{-128}	130	65	$65m$ exponentiations
$c = 3$	47	2^{-128}	141	47	$47m$ exponentiations
$c = 4$	40	2^{-128}	160	40	$40m$ exponentiations
$c = 5$	36	2^{-128}	180	36	$36m$ exponentiations
$c = 6$	34	2^{-128}	204	34	$34m$ exponentiations
$c = 7$	32	2^{-128}	224	32	$32m$ exponentiations
$c = 8$	30	2^{-128}	240	30	$30m$ exponentiations
$c = 9$	29	2^{-128}	261	29	$29m$ exponentiations

$10, 11, \ldots, m$ numerically, so the protocol reduces the number of multiplication (i.e. t_C) and number of repetitions. However, as mentioned before, starting $c = 10$ (please refer to Table 3.2), the protocol offers more group exponentiations with a random exponent (i.e. t_P) from the offline phase than the protocol from Section 3.2.

3.4.2 Delegating Exponentiation with Improved Security Error Reduction with Multiple Execution for Fixed-Exponent Variable-Base Group

This subsection is similar to the previous subsection, the only difference is that we apply the previous protocol on the function $F_{G,exp,k} : G \to G$ denoted as $F_{G,exp,k}(x) = x^k$. We will write the theorem and the protocol which satisfies the theorem.

Theorem 3.4.2. *Let σ, λ be security parameters and c, m be protocol parameters and let k be a positive integer and assume the exitance of a pseudo-random (G, k)-powers generator. There exists (constructively) a client-server protocol (C_R, S_R) for delegated computation of function $F_{G,exp,k}$ which satisfies*

1. *δ_c-correctness, for $\delta_c = 1$*
2. *ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$*
3. *ϵ_p-privacy, for $\epsilon_p = 0$*
4. *efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where $R = \left\lceil \frac{\lambda}{\log_2(\frac{1}{\epsilon_s})} \right\rceil$ and*
   - *t_F is $= 1$ group exponentiation in G*
   - *t_S is $= m \cdot R$ group exponentiations in G*
   - *t_P is $= c \cdot R$ group exponentiations with random exponents in G using pseudo-random (G, k)-powers generator.*
   - *t_C is $= 5 \cdot R$ group multiplications in G*

- $cc = (2m + 2) \cdot R$ *elements in* G *and* $mc = 2$.

Formal description of the protocol (C, S)**.** We describe the client-server (C, S) protocol for the function $F_{G,exp,k}$. We use Theorem A.0.1 in Appendix A on (C_{inv}, S_{inv}) for delegated computation of inverses in a group G.

Input to S: 1^σ, 1^λ, $desc(F_{G,exp,k})$

Input to C: 1^σ, 1^λ, $desc(F_{G,exp,k})$, $x \in G$, $aux = Rand_{G,k}(0)$

Protocol instructions:

Offline phase instructions:

1. C computes $(u_{i,j}, v_{i,j}, aux) = Rand_{G,k}(i_j, aux)$, for $i = 1, \ldots, \lambda$ and $j = 1, \ldots, c$;

Online phase instructions:

1. For each $i = 1, \ldots, \lambda$

   C randomly chooses distinct $j_1, \ldots, j_m \in \{1, \ldots, m\}$;

   C sets $z_{i,j_1} := u_{i,1}, \ldots, z_{i,j_{c-1}} := u_{i,c-1},\ z_{i,j_c} := x * u_{i,c}$;

   C randomly and independently chooses $z_{i,j_{c+1}}, \ldots, z_{i,j_m} \in G$ when $c < m$;

   C runs C_{inv} on input $v_{i,c}$, thus obtaining d_i;

   C sends $z_{i,1}, \ldots, z_{i,m}, d_i$ to S;

2. For each $i = 1, \ldots, \lambda$

   S computes $w_{i,j} := z_{i,j}^k$ for $j = 1, \ldots, m$;

S runs S_{inv} on input d_i, thus obtains e_i;

S sends $w_{i,1}, \ldots, w_{i,m}, e_i$ to C

3. For each $i = 1, \ldots, \lambda$

   C runs C_{inv} on input $v_{i,c}, d_i, e_i$ to compute $v_{i,c}^{-1}$;

   if this execution of (C_{inv}, S_{inv}) returned $\perp$ as output for some $i = 1, \ldots, \lambda$

      C **returns** $\perp$ and the protocol halts;

   if $w_{i,j_1} \neq v_{i,j_1}$ or $w_{i,j_2} \neq v_{i,j_2}$ or $\ldots$ or $w_{i,j_{c-1}} \neq v_{i,j_{c-1}}$

      C **returns** $\perp$ and the protocol halts;

   C computes $y_i := w_{i,j_c} * v_{i,j_c}$

   if $y_1 = \cdots = y_\lambda$

      C **returns** y_1

   else C **returns** $\perp$

Illustration of the Protocol (C, S)**:**

Let $i = 1, \ldots, \lambda$

Client **Server**

$(u_{i,j}, v_{i,j}, aux) = Rand_{G,k}(i_j, aux)$ for $j = 1, \ldots, c$

$j_1, \ldots, j_m \in_R \{1, \ldots, m\}$

$z_{i,j_1} := u_{i,1}, \ldots, z_{i,j_{c-1}} := u_{i,c-1}, \ z_{i,j_c} := x * u_{i,c}$

$z_{i,j_{c+1}}, \ldots, z_{i,j_m} \in_R G$ random elements

$C_{inv}(v_{i,c}) = d_i$ $\xrightarrow{z_{i,1},\cdots,z_{i,m},d_i}$ $w_{i,j} := z_{i,j}^k$ for $j = 1, \ldots, m$

$\xleftarrow{w_{i,1},\cdots,w_{i,m},e_i}$ $S_{inv}(d_i) = e_i$

$C_{inv}(v_{i,c}, d_i, e_i) = v_{i,c}^{-1}$

if (C_{inv}, S_{inv}) returned $\perp$ as output for some $i = 1, \ldots, \lambda$

C **returns** $\perp$ and protocol halts;

if $w_{i,j_1} \neq v_{i,j_1}$ or $w_{i,j_2} \neq v_{i,j_2}$ or $\ldots$ or $w_{i,j_{c-1}} \neq v_{i,j_{c-1}}$ then

C **returns** $\perp$ and protocol halts;

C computes $y_i := w_{i,j_c} * v_{i,j_c}^{-1}$

if $y_1 = \cdots = y_\lambda$ then

C **returns** y_1

else C **returns** $\perp$ ∎

Proof of Theorem 3.4.2: properties of the protocol satisfies similar as the proof of Theorem 3.2.1 and Theorem 3.3.1. □

Chapter 4

Cyclic Groups Exponentiation

In this chapter we show a client-server protocol (C, S) which satisfies correctness, security, privacy and efficiency requirements for delegated computation of exponentiation in the cyclic group, in the model where the adversary corrupting the server can be malicious. We present an efficient atomic protocol in Section 4.1 using the idea of probabilistic check which satisfies four requirements where the security probability is $2^{-\lambda}$. Next, we present a protocol in Section 4.2, which is similar as the protocol in Section 4.1, but the number of C's multiplication in the online phase is much lower; this protocol achieves the following efficiency tradeoff: it reduces the number of the client's group multiplications during the online phase, while it increases the number of group exponentiations of random exponents during the offline phase and the number of the server's group exponentiations. Both of these protocols achieve much better efficiency and security properties in one atomic protocol, in comparison with the protocols in Chapter 3. In other words in Chapter 3, we have to use the idea of parallel repetition of an atomic protocol in order to achieve $\epsilon_s = \frac{1}{2^\lambda}$, but this protocol can run only once and achieve $\epsilon_s = \frac{1}{2^\lambda}$. We start the presentation of the

two protocols with group notations that are relevant to both protocols.

Group notations. Let ℓ denote the length of the binary representation of a group's elements. We say that a group is *efficient* if its description is short (i.e., has length polynomial in ℓ), its associated operation $*$ and the inverse operation are efficient (i.e., they can be executed in time polynomial in ℓ). Let $(G, *)$ be an efficient cyclic group of order q with generator g. Let $y = g^x$ denote the *exponentiation (in G)* of g to the x-th power; i.e., the value $y \in G$ such that $g * \cdots * g = y$, where the multiplication operation $*$ is applied $x - 1$ times. Let $\mathbb{Z}_q = \{0, 1 \ldots, q - 1\}$, and let $F_{G,exp,g} : \mathbb{Z}_q \to G$ denote the function that maps every $x \in \mathbb{Z}_q$ to the exponentiation (in G) of g to the x-th power.

4.1 Basic Delegation of Exponentiation in Cyclic Groups

In this section we present a client-server protocol for delegated computation of group exponentiation in a cyclic group. First, we formally state the result of the protocol, then we describe the delegation of the protocol, and finally prove its correctness, security, privacy and efficiency properties.

Formal theorem statement. We show the following

Theorem 4.1.1. Let $(G, *)$ be an efficient cyclic group of order q, let σ be its computational security parameter, and let λ be a statistical security parameter. There exists (constructively) a client-server protocol (C, S) for delegated computation of function $F_{G,exp,g}$ which satisfies

1. δ_c-correctness, for $\delta_c = 1$;

2. ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$;

3. ϵ_p-privacy, for $\epsilon_p = 0$;

4. efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where

   - $t_F = t_{exp}(\sigma)$;
   - $t_S = 2 \cdot t_{exp}(\sigma)$;
   - $t_P = 2 \cdot t_{exp}(\sigma)$ with random exponents in $\mathbb{Z}_q$;
   - $t_C = t_{exp}(\lambda) + 3$;
   - $cc = 4$ elements in G and $mc = 2$.

We remark that Theorem 4.1.1 satisfies significantly better efficiency, to achieve security parameter $\epsilon_s = 2^{-\lambda}$. Specifically,

- the client's number of exponentiations with random exponent using the pseudo random power generator during the offline phase is small constant (i.e. $= 2$)

- the client's number of multiplications using square and multiply algorithm is at most $2\lambda + 3 = 259$ (or using the algorithm described in [8], or [24] is at most 177 or 149 respectively) when $\lambda = 128$. Note that $2\lambda + 3 = 259$ is much smaller than the non delegated square and multiply algorithm $2\sigma = 4096$ where σ equals to 2048 which is the recommended parameter setting.

Now we describe the protocol satisfying Theorem 4.1.1.

Informal description of protocol (C, S)**.** In previous protocols, the verification of server computations were based on deterministic tests and repetitions of an atomic protocol with (in almost all cases) constant security probability. In this section, we consider a family of protocols which differs on both of these aspects: there is no repetition of atomic protocols, and probabilistic tests are used. This is helpful in some ways, for instance: the lack of repetition of an atomic protocol keeps a low number of exponentiations computed in the offline phase, and the use of probabilistic tests allows a significant reduction of the security probability with at most a small number of test equations. Specifically, for this section's protocol in cyclic groups, C injects an additional random element into the inputs on which S is asked to compute the value of the function F to satisfy the following properties: (a) if S returns correct computations of F, then C can use these random values in a probabilistic test to correctly compute y, (b) if S returns incorrect computations of F, then S can pass C's verification only for very few possible values of the random elements, (c) C's messages hide the values of the random element as well as C's input to the function. By choosing a large domain from which this random value is chosen, the protocol achieves a very small security probability, without the need for repetition.

The probabilistic test we use is based on an equation that involves the answer from S, a correct exponentiation computed in the offline phase, and the potential function output computed using the answer from S and another correct exponentiation computed in the offline phase. Only one of these values is exponentiated to an exponent that is $\leq 2^{\lambda}$ by

C, which can be much smaller than $|G|$. In itself, this probabilistic test is not always successful, but we characterize the condition under which it fails, and this condition can be expressed as 2 computationally simple deterministic tests: a group membership and a value distinctness test, both of which can be efficiently checked by C. The distinctness test removes from this protocol the capability of delegating exponentiation to the exponent 0, which is not a practical concern.

Formal description of protocol (C, S)**.**

Input to S: $1^\sigma, 1^\lambda$, $desc(F_{G,exp,g})$, $g \in G$

Input to C: $1^\sigma, 1^\lambda$, $desc(F_{G,exp,g})$, $x \in \mathbb{Z}_q$, $g \in G$

Offline phase instructions:

1. C randomly chooses $u_i \in \mathbb{Z}_q$, for $i = 0, 1$
2. C sets $v_i = g^{u_i}$, for $i = 0, 1$

Online phase instructions:

1. If $x = 0$

   C randomly chooses z_0, z_1 from $\mathbb{Z}_q$ and sends z_0, z_1 to S

   C randomly chooses $b \in \{1, \ldots, 2^\lambda\}$

   C sets $z_0 := (x - u_0) \mod q$, $z_1 := (b \cdot x + u_1) \mod q$

   C sends z_0, z_1 to S

2. S computes $w_i := g^{z_i}$ for $i = 0, 1$

   S sends w_0, w_1, to C

3. If $x = 0$

   C **returns** $y = 1$ and the protocol halts

   C computes $y := w_0 * v_0$

   C checks that $y \neq 1$, also called the 'distinctness test'

   C checks that $w_1 = y^b * v_1$, also called the 'probabilistic test'

   C checks that $w_0, w_1 \in G$, also called the 'G-membership test'

   if any one of these tests is not satisfied then

   C **returns** $\perp$ and the protocol halts

   C **returns** y

Proof of Theorem 4.1.1:

Properties of protocol (C, S)**:**

1. *The efficiency properties* are verified by protocol inspection such that

   (a) t_F is $= 1$ exponentiation in G (because we need to calculate $F_{G,exp,g}(x) = g^x$);

   (b) S required to calculate 2 exponentiations (i.e. t_S is $= 2$ exponentiation in G);

   (c) C is required to calculate 2 exponentiations during the offline phase (i.e. t_P is $= 2$ exponentiation in G);

   (d) During the online phase, C is required to calculate 3 multiplications and 1 exponentiations to a random exponent b which can be at most 2^λ. If C uses

the square and multiply algorithm it takes 1.5σ to calculate in the average case and 2σ in the worst case. C performs at most $2\lambda + 3$ multiplications during online phase (i.e. t_C is $\leq 2\lambda + 3$ multiplications). Specifically, when $\lambda = 128$ and $\sigma = 2048$, then $t_C \leq 259$ group multiplications by C during the online phase, but using the square and multiply method, C has to perform, on average, 3072 multiplications, and 4096 multiplications in the worst case.

(e) With respect to *round complexity*, the protocol requires one round complexity, consisting of one message from C to S followed by one message from S to C (i.e. $mc = 2$). With respect to *communication complexity* the protocol requires two elements in $\mathbb{Z}_q$ from C and two elements in G from S (i.e. $cc = 4$).

2. *Correctness* properties are satisfied if C and S follow the protocol. All three verification in step 3 will be satisfied, as follows

(a) "distinctness test": $y \stackrel{?}{\neq} 1$. Since $x \neq 0$ and g is generator of G we have

$$y = w_0 * v_0 = g^{z_0} * g^{u_0} = g^{x-u_0} * g^{u_0} = g^x \neq 1$$

which implies that $u_0 \neq (x - u_0) \mod q$

(b) "probabilistic test": $w_1 \stackrel{?}{=} y^b * v_1$. Since

$$w_1 = g^{z_1} = g^{bx+u_1} = (g^x)^b * g^{u_1} = y^b * v_1$$

(c) "G-membership test": $w_0, w_1 \stackrel{?}{\in} G$. Since $w_i = g^{z_i}$ for $i = 0, 1$ and g is a

generator of G, then we must have $w_i \in G$ for $i = 0, 1$.

and thus C outputs y if the protocol does not halt (i.e. $\neq \perp$) and $y = g^x$ as we show in the "distinctness test".

3. *The privacy property* against a malicious S follows by observing that the messages (z_0, z_1) sent by C do not leak any information about $x \in \mathbb{Z}_q$. This message is a pair (z_0, z_1) where $z_0 = (x - u_0) \mod q$ and $z_1 = b \cdot x + u_1 \mod q$, u_0 and u_1 are uniformly and independently distributed in $\mathbb{Z}_q$, and thus so are z_0 and z_1. Therefor, we can generate two properties from the above reasoning, which we use to prove security:

   (a) (z_0, z_1) are uniformly and independently distributed in $\mathbb{Z}_q$ for any input x.

   (b) (z_0, z_1) does not leak any information about b for any input x.

4. To show *security property* against any malicious S we first write several definitions and properties and then write the proof of security by finding the upper bound for ϵ_s, where ϵ_s stands for the security probability for that S convinces C to output a y such that $y \neq F_{G,exp,g}(x)$. Similarly, as in chapter 3, we define the following events on the input $x \in \mathbb{Z}_q$ of C on the protocol (C, S):

   - $e_{y \neq F}$, meaning 'C outputs y such that $y \neq F_{G,exp,g}(x)$'
   - $e_{\perp}$, meaning 'C outputs $\perp$'

By protocol inspection we observe that if C outputs y where $y \neq F_{G,exp,g}(x)$ then C does **not** output $\perp$. Thus we obtain the following observation

Observation 4.1.1. *If the event $e_{y \neq F}$ holds then the event $(e_{\perp})^c$ holds.*

Now let us define the following events which we use to prove the security requirement:

- $e_{1,b}$ defined as "there exist exactly one b such that the message of S is (w_0, w_1) which satisfies $w_1 = (w_0 * v_0)^b * v_1$"
- $e_{>1,b}$ defined as "there exist more than one b such that the message of S is (w_0, w_1) which satisfies $w_1 = (w_0 * v_0)^b * v_1$".

By the above definitions, events $e_{1,b}$ and $e_{>1,b}$ are complements of each other. In the proof of the privacy property of this protocol (C, S), we prove that "for any x, the message (z_0, z_1) of C does not leak any information about b". From this statement we can note that all values $\{1, \ldots, 2^{\lambda}\}$ are equally likely to be chosen for b under the condition that the event $e_{1,b}$ is true. From this fact we obtain the following observation:

Observation 4.1.2. $\Pr[(e_{\perp})^c | e_{1,b}] \leq \frac{1}{2^{\lambda}}$

The main proof of the security property is to show that *any S cannot produce values w_0, w_1 in step 2 satisfying all three checks of C in step 3 for two values of $b \in \{1, \ldots, 2^{\lambda}\}$ where $2^{\lambda} < q = |G|$, say b_1, $b_2 \in \{1, \ldots, 2^{\lambda}\}$.*

To prove this statement: we know that S can be malicious, in step 2 it can send correct answer or incorrect answer. In other words, S can send w_i' for $i = 0, 1$ where $w_i' = w_i$ or $w_i' \neq w_i$ where $w_i = g^{z_i}$. Since the group G is cyclic and g is a generator of G and C checks that $w_i' \in G$ in step 3, we can write

$$w_0' = g^u * w_0 \text{ and } w_1' = g^v * w_1 \text{ for some } u, v \in \mathbb{Z}_q$$

then $y = w_0' * v_0 = g^u * w_0 * v_0 = g^u * g^x$. The goal of a malicious S is to pass three checks and C's output $y \neq g^x$ then $u \neq 0 \mod q$. Now, consider the 'probabilistic check':

$$\begin{aligned} w_1' &= y^b * v_1 \\ g^v * w_1 &= (g^u * g^x)^b * g^{u_1} \\ g^v * g^{z_1} &= g^{ub} * g^{bx+u_1} \\ g^v * g^{bx+u_1} &= g^{ub} * g^{bx+u_1} \\ g^v &= g^{ub} \\ v &= ub \mod q \end{aligned}$$

Notice that if $u = 0 \mod q$ then $v = 0 \mod q$, from the above calculation, which implies that S is honest then $\epsilon_s = 0$. Now consider when S is dishonest. Then $u \neq 0 \mod q$. We want to show that b is unique in this case. If there exist two distinct b_1

and b_2 such that

$$ub_1 = v \mod q \text{ and } ub_2 = v \mod q$$

then $u(b_1 - b_2) = 0 \mod q$ then $b_1 - b_2 = 0 \mod q$ (i.e. $b_1 = b_2$) because $u \neq 0 \mod q$. Which shows that b is unique. Thus we obtain the following observation

Observation 4.1.3. $Pr[e_{>1,b}] = 0$

Now, we will prove the upper bound ϵ_s on the probability of the event $e_{y \neq F}$ as follows:

$$\begin{aligned}
\epsilon_s = \Pr[e_{y \neq F}] &\leq \Pr[(e_\perp)^c] \quad \text{by Observation 4.1.1} \\
&= \Pr[e_{1,b} \cap \ (e_\perp)^c] + \Pr[e_{>1,b} \cap \ (e_\perp)^c] \\
&= \Pr[e_{1,b}] \cdot \Pr[(e_\perp)^c | e_{1,b}] + \Pr[e_{>1,b}] \cdot \Pr[(e_\perp)^c | e_{>1,b}] \\
&= \Pr[e_{1,b}] \cdot \Pr[(e_\perp)^c | e_{1,b}] \quad \text{by Observation 4.1.3} \\
&\leq \Pr[e_{1,b}] \cdot \frac{1}{2^\lambda} \quad \text{by Observation 4.1.2} \\
&\leq \frac{1}{2^\lambda}
\end{aligned}$$

The above calculation shows that the security probability can be at most $\frac{1}{2^\lambda}$ in one atomic execution of the protocol. □

4.2 Delegation of Exponentiation in Cyclic Groups with Tradeoff Runtime Performance

In this section we present a client-server protocol for delegated computation of group exponentiation which is similar to the previous section's protocol for the function $F_{G,exp,g}(x)$ in a cyclic group G. In this section, the goal is to improve the number of C's multiplication in the online phase (i.e. to decrease t_C). Note that in the last section the most expensive operation for C is to evaluate one exponentiation with λ bit. In this section we introduce the protocol where C calculates multiple exponentiations with smaller bit than λ and evaluates exponentiations by using the techniques introduced in [8, 24].

Formal theorem statement. We show the following

Theorem 4.2.1. Let σ, λ be security parameters along with an integer m, as protocol parameters. There exists (constructively) a client-server protocol (C, S) for delegated computation of the function $F_{G,exp,g}$ which satisfies

1. δ_c-correctness, for $\delta_c = 1$;
2. ϵ_s-security, for $\epsilon_s = \frac{1}{2^\lambda}$;
3. ϵ_p-privacy, for $\epsilon_p = 0$;
4. efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where
   - $t_F = t_{exp}(\sigma)$;

- $t_S = (m+1) \cdot t_{exp}(\sigma)$;
- $t_P = (m+1) \cdot t_{exp}(\sigma)$ with random exponents in $\mathbb{Z}_q$;
- $t_C = m \cdot t_{exp}(\lambda/m) + 2m + 1$;
- $cc = m + 1$ elements in $\mathbb{Z}_q$ and $m + 1$ elements in G
- $mc = 2$.

Informal description of protocol (C, S)**.** In the previous section we present a correct, efficient protocol, which satisfies the privacy requirement and the security probability $\epsilon_s = 2^{-\lambda}$. Specifically, when $\lambda = 128$, the security probability is $\epsilon_s = 2^{-128}$ and C's online calculation is $t_C = 259$ modular multiplications using the square and multiply algorithm, which is slightly more work for computationally weaker devises. In this section, we present similar a protocol as that is in previous section, which keeps the same security probability (i.e. $\epsilon_s = 2^{-\lambda}$) and improves the number of online multiplications by C, specifically the protocol decreases C's online multiplication up to 63 i.e. $t_C = 63$ modular multiplications. In the previous protocol C sends two elements to S in order to get exponentiations for both of the elements; one is used to evaluate the function $F_{G,exp,g}$ and the other is used to check the probabilistic test. The most expensive calculation for C is to calculate one exponentiation in the probabilistic test, which is $\leq 2^{\lambda}$ and it costs at most 2λ modular multiplications for C. In this section, C sends $m + 1$ elements to S in order to get exponentiations for those elements; one element is used to evaluate the function $F_{G,exp,g}$ and m elements are used to check the m probabilistic tests. C performs m exponentiations

with a fixed base and with exponent $\leq 2^{\lceil \frac{\lambda}{m} \rceil}$ which costs for C, in the online phase, only 63 multiplications by using the techniques in [8, 24]. This is compared to the 259 modular multiplications discussed in the previous section. The protocol achieves the following efficiency tradeoff: it reduces the number of group multiplications by C during the online phase, while increasing the number of group exponentiations of random exponents during the offline phase and the number of the server's group exponentiations. Now we describe the protocol as follows:

Formal description of protocol (C, S)**.**

Input to S: $1^\sigma, 1^\lambda$, $desc(F_{G,exp,g})$ $g \in G$, parameters $1^m, 1^b$

Input to C: $1^\sigma, 1^\lambda, desc(F_{G,exp,g})$, q, and $x \in \mathbb{Z}_q$, $g \in G$, parameters $1^m, 1^b$

Offline phase instructions:

1. For $j = 0, \ldots, m$,

   C randomly chooses $u_j \in \mathbb{Z}_q$,

   C sets $v_j = g^{u_j}$

Online phase instructions:

1. If $x = 0$

   C randomly chooses $z_0, \ldots, z_m$ from $\mathbb{Z}_q$ and sends them to S

   For $i = 1, \ldots, m$,

   C randomly chooses $b(i) \in \left\{1, \ldots, 2^{\lceil \frac{\lambda}{m} \rceil}\right\}$

C sets $z_0 = (x - u_0) \mod q$ and $z_i = (b(i) \cdot x + u_i) \mod q$

C sends $z_0, \ldots, z_m$ to S

2. For $j = 0, \ldots, m$,

   S computes $w_j = g^{z_j}$ and sends w_j to C

3. If $x = 0$

   C **returns** $y = 1$ and the protocol halts

C computes $y = w_0 * v_0$

For $i = 1, \ldots, m$,

C checks that $y \neq 1$, also called the 'distinctness test'

C checks that $w_i = y^{b(i)} * v_i$, also called the 'probabilistic test'

C checks that $w_0, w_i \in G$, also called the 'G-membership test'

if any one of these tests is not satisfied then

C **returns** $\perp$ and the protocol halts

C **returns** y

Properties of protocol (C, S)**:** *The efficiency properties* are verified by protocol inspection. 1. *With respect to round complexity,* the protocol only requires one round, consisting of one message from C to S followed by one message from S to C. 2. *With respect to communication complexity*, the protocol requires the transfer of $m+1$ elements in G from

S to C, and $m+1$ elements in $\mathbb{Z}_q$ from C to S. 3. *With respect to runtime complexity*, if $x = 0$, C randomly chooses $m+1$ elements from $\mathbb{Z}_q$ and returns $y = 1$ without any checks, otherwise C performs $m+1$ exponentiations with random exponents in the offline phase and S performs $m+1$ exponentiations. In the online phase, C performs $2m+1$ modular multiplications and m exponentiations, which is at most $\lceil \frac{\lambda}{m} \rceil$ bits.

The correctness properties are demonstrated by observing that if C and S follow the protocol, and if $x = 0$, C returns $y = 1 = g^0$. Otherwise C performs three tests which pass with high probability and C will output y such that $y = g^x$. To see that this is the correct output, note that

$$y = w_0 * v_0 = g^{z_0} * g^{u_0} = g^{x-u_0} * g^{u_0} = g^x.$$

The G-membership test is always passed since $w_j = g^{z_j}$, for $j = 0, 1, \ldots, m$, and g is a generator of the group G. The probabilistic test is always passed for $i = 1, \ldots, m$ since

$$w_i = g^{z_i} = g^{b(i)x+u_i} = (g^x)^{b(i)} * g^{u_i} = y^{b(i)} * v_i.$$

The distinctness test is always passed, since we assume $x \neq 0$ then $-u_0 \neq (x - u_0)$ mod q and using the fact that g is a generator of the group G of order q, then we have $v_0^{-1} = g^{-u_0} \neq g^{x-u_0} = g^{z_0} = w_0$, equivalently saying that $y = w_0 * v_0 \neq 1$. This implies that C never returns $\perp$, and thus returns y which is equal to $F_{G,exp,g}$ by above calculation.

The privacy against a malicious S of the protocol follows, similarly as for the previous protocols, by observing that C's only message to S does not leak any information about

x. This message is $(z_0, z_1, \ldots, z_m)$ where $z_0 = (x - u_0) \mod q$, $z_i = (b(i) \cdot x + u_i) \mod q$, u_0 and u_i are uniformly and independently distributed in $\mathbb{Z}_q$, and thus so are z_0 and z_i for all $i = 1, \ldots, m$. By the same reasoning it satisfies the following two properties for all $j = 0, \ldots, m$: (1) for any x, z_j are uniformly and independently distributed in $\mathbb{Z}_q$; and (2) for any x, z_j do not leak any information about $b(i)$. We will use both facts in the proof of the security property.

To prove *the security property* against a malicious S we need to compute an upper bound ϵ_s on the security probability that i.e. the probability S convinces C to output y such that $y \neq F_{G,exp,g}(x)$. If $x = 0$, C can calculate $F_{G,exp,g}(x) = g^0 = 1$ and does not need to check whether S is honest or dishonest. Thus $\epsilon_s = 0$ when $x = 0$. Now, we assume that $x \neq 0$. In the online phase of step 1, C chooses randomly and independently $b(i)$ from from the set of $\{1, \ldots, 2^{\lceil \frac{\lambda}{m} \rceil}\}$ for each $i = 1, \ldots, m$. We obtain the following facts.

Observation 4.2.1. *The distribution of values* $b(1), \ldots, b(m) \in \{1, \ldots, 2^{\lceil \frac{\lambda}{m} \rceil}\}$ *is randomly and independently chosen by* C.

With respect to a random execution of (C, S) where C uses x as input, we define the following events for all $i = 1, \ldots, m$:

- $e_{1,b(i)}$, defines as '$\exists$ exactly one $b(i)$ such that S's message $(w_0, \ldots, w_m)$ satisfies $w_i = (w_0 * v_0)^{b(i)} * v_i$'

- $e_{>1,b(i)}$, defines as '$\exists$ more than one $b(i)$ such that S's message $(w_0, \ldots, w_m)$ satisfies $w_i = (w_0 * v_0)^{b(i)} * v_i$'.

By definition, events $e_{1,b(i)}, e_{>1,b(i)}$ are complement of each other. By Observation 4.2.1, C chooses randomly and independently $b(i)$ from the set of $\{1, \ldots, 2^{\lceil \frac{\lambda}{m} \rceil}\}$ for each $i = 1, \ldots, m$.

In our proof of the privacy property for (C, S), we show that for any x, C's message $z_0 \ldots, z_m$ does not leak any information about $b(1), \ldots, b(m)$. This implies that all values in $\{1, \ldots, 2^{\lceil \frac{\lambda}{m} \rceil}\}$ are equally likely events for each $b(i)$ when conditioning over the message $(z_0, \ldots, z_m)$. Then, if the event $e_{1,b(i)}$ is true, the probability that S's message $(w_0, \ldots, w_m)$ satisfies the probabilistic test is 1 divided by the number $2^{\lceil \frac{\lambda}{m} \rceil}$ of values of $b(i)$ that are still equally likely even when conditioning over message $(z_0, \ldots, z_m)$. We obtain the following observation

Observation 4.2.2. $\Pr[(e_\perp)^c | e_{1,b(i)}] \leq 2^{-\lceil \frac{\lambda}{m} \rceil}$

We now show the main technical claim, saying that if S is malicious then in step 2 of the protocol it cannot produce values $w'_0, \ldots, w'_m$ satisfying all of C's three tests relative to two distinct values $b_1(i), b_2(i) \in \{1, \ldots, 2^{\lceil \frac{\lambda}{m} \rceil}\}$ for each $i = 1, \ldots, m$:
Since S can be malicious, in step 2 it can send a correct answer or an incorrect answer. In other words, it can send w'_j where $w'_j = w_j$ or $w'_j \neq w_j$ where $w_i = g^{z_j}$ for each $j = 0, \ldots, m$. Since the group G is cyclic, g is a generator of G and C checks in step 3

that $w'_j \in G$, we can write

$$w'_j = g^{\alpha_j} * w_j \text{ for some } \alpha_j \in \mathbb{Z}_q \text{ and } j = 0, \ldots, m$$

then $y = w'_0 * v_0 = g^{\alpha_0} * w_0 * v_0 = g^{\alpha_0} * g^x$. The goal of a malicious S is to pass all three checks and C's output $y \neq g^x$ then $\alpha_0 \neq 0 \mod q$. Now, consider 'probabilistic check' for each $i = 1, \ldots, m$:

$$w'_i = y^{b(i)} * v_i$$

$$g^{\alpha_i} * w_i = (g^{\alpha_0} * g^x)^{b(i)} * g^{u_i}$$

$$g^{\alpha_i} * g^{z_i} = g^{\alpha_0 b(i)} * g^{b(i)x+u_i}$$

$$g^{\alpha_i} * g^{b(i)x+u_i} = g^{\alpha_0 b(i)} * g^{b(i)x+u_i}$$

$$g^{\alpha_i} = g^{\alpha_0 b(i)}$$

$$\alpha_i = \alpha_0 \cdot b(i) \mod q$$

Notice that if $\alpha_0 = 0 \mod q$ then $\alpha_i = 0 \mod q$ for all $i = 1, \ldots, m$ from the above calculation, and if S is honest then $\epsilon_s = 0$. Now, the reverse direction. If $\alpha_i = 0 \mod q$ then $\alpha_0 = 0$ because $b(i) \neq 0$ for $i \in \{1, \ldots, m\}$. If there exists some $j \in \{1, \ldots, m\}$ such that $\alpha_j \neq 0$, we want to show that $b(j)$ is unique. Toward contradiction, if there exist two distinct $b_1(j)$ and $b_2(j)$ such that

$$\alpha_j b_1(j) = \alpha_0 \mod q \text{ and } \alpha_j b_2(j) = \alpha_0 \mod q$$

then $\alpha_j(b_1(j) - b_2(j)) = 0 \mod q$ then $b_1(j) - b_2(j) = 0 \mod q$ (i.e $b_1(j) = b_2(j)$) because

$\alpha_j \neq 0 \mod q$. Which shows that each $b(i)$ is unique for all $i = 1, \ldots, m$. From this we obtain the following observation

Observation 4.2.3. $\Pr[e_{>1,b(i)}] = 0$ *for all* $i = 1, \ldots, m$

The rest of the proof consists of computing an upper bound ϵ_s on the probability of event $e_{y \neq F}$. We have the following

$$
\begin{aligned}
\Pr[e_{y\neq F}] &\leq \Pr[(e_\perp)^c] \\
&= \Pr[(e_\perp)^c \cap (e_{1,b(1)} \cap \ldots \cap e_{1,b(m)})] + \Pr[(e_\perp)^c \cap (e_{1,b(1)} \cap \ldots \cap e_{1,b(m)})^c] \\
&= \Pr[((e_\perp)^c \cap e_{1,b(1)}) \cap \ldots \cap ((e_\perp)^c \cap e_{1,b(m)})] + \Pr[(e_\perp)^c \cap (e_{>1,b(1)} \cup \ldots \cup \ e_{>1,b(m)})] \\
&= \Pr[((e_\perp)^c \cap e_{1,b(1)}) \cap \ldots \cap ((e_\perp)^c \cap e_{1,b(m)})] \\
&\quad + \Pr[((e_\perp)^c \cap e_{>1,b(1)}) \cup \ldots \cup \ ((e_\perp)^c \cap e_{>1,b(m)})] \\
&\leq \prod_{i=1}^{m} \Pr[(e_\perp)^c \cap e_{1,b(i)}] + \sum_{i=1}^{m} \Pr[(e_\perp)^c \cap e_{>1,b(i)}] \\
&= \prod_{i=1}^{m} \Pr[e_{1,b(i)}] \cdot \Pr[(e_\perp)^c | e_{1,b(i)}] + \sum_{i=1}^{m} \Pr[e_{>1,b(i)}] \cdot \Pr[(e_\perp)^c | e_{>1,b(i)}] \\
&\leq \prod_{i=1}^{m} \Pr[e_{1,b(i)}] \cdot 2^{-\lceil \frac{\lambda}{m} \rceil} \\
&\leq (2^{-\lceil \frac{\lambda}{m} \rceil})^m \\
&\leq \frac{1}{2^\lambda},
\end{aligned}
$$

where the first inequality follows from Observation 4.1.1 in the previous section, and the

first, second and third equalities follow from partitioning the event $\neg\, e_{\perp}$ into two disjoint events using the definition of events $e_{1,b(i)}$ and $e_{>1,b(i)}$, then simplifying the expressions. The second inequality follows from Observation 4.2.1 The fourth equality follows from the conditioning rule in probability. The third inequality follows from Observation 4.2.3. Thus, we obtain that $\epsilon_s = 2^{-\lambda}$ and conclude the proof for the security property for execution of (C, S). □

4.3 Performance Analysis

The performance of our protocols has been expressed in terms of group multiplications and parameterized by functions t_{exp} and $t_{exp,m}$ where $t_{exp,m} := m \cdot t_{exp}$ by definition. We have two different settings to evaluate group exponentiations to group multiplications in our protocols:

1. **Basic settings** uses the square and multiply algorithm to evaluate group exponentiation i.e.

   - $t_{exp}(\ell) = 2\ell$
   - $t_{exp,m}(\ell) = 2m\ell$

2. **Improved settings** uses improved algorithms from the literature to evaluate group exponentiation. We use the closed-form estimates in [34], also in [8, 19, 33, 35], for other algorithms claiming improvements without closed-form evaluations. By studying these we obtain that:

- $t_{exp,G}(\ell) \approx \ell(1 + \frac{1}{\log \ell})$ using Brauer's 1939 algorithm
- $t_{exp,G,m}(\ell) \approx \ell(1 + \frac{m}{\log \ell})$ using Yao's 1976 algorithm

Table 4.1 below compares the performance of efficiency parameters in protocols of Sections 4.1 and 4.2 with non-delegated computation of the client under both basic (B) and improved (I) settings.

Table 4.1: Performance of the protocols (C, S)

Perf	**Basic Imp.**	$F_{G,exp,g}$ **with no Delegation**	$F_{G,exp,g}$ **with Delegation**	
			Section 4.1	*Section 4.2*
t_F	B	2σ	2σ	2σ
	I	$\sigma(1 + \frac{1}{\log \sigma})$	$\sigma(1 + \frac{1}{\log \sigma})$	$\sigma(1 + \frac{1}{\log \sigma})$
t_P	B	0	4σ	$2\sigma(m+1)$
	I	0	$\sigma(1 + \frac{2}{\log \sigma})$	$\sigma(1 + \frac{m+1}{\log \sigma})$
t_C	B	2σ	$2\lambda + 3$	$2m\lambda' + 2m + 1$
	I	$\sigma(1 + \frac{1}{\log \sigma})$	$\lambda(1 + \frac{1}{\log \lambda}) + 3$	$\lambda'(1 + \frac{m}{\log \lambda'}) + 2m + 1$
t_S	B	0	4σ	$2\sigma(m+1)$
	I	0	$\sigma(1 + \frac{2}{\log \sigma})$	$\sigma(1 + \frac{m+1}{\log \sigma})$
ϵ_p	B & I	0	0	0
ϵ_s	B & I	0	$2^{-\lambda}$	$2^{-\lambda}$

The main takeaway from the table is that by comparing non-delegated evaluation of exponentiation with our protocols, we have the following analysis:

- our protocol in Section 4.1 reduces t_C by a multiplicative factor of about σ/λ with respect to non-delegated computation when using both basic and improved settings;
- the protocol in Section 4.2 reduces t_C by a multiplicative factor of about σ/λ' when using improved settings (where $\lambda' = \lceil \lambda/m \rceil$ by definition);

Table 4.2: Efficiency parameters of protocols using square and multiply algorithm and Yao's algorithm

m	Improved t_P and t_S	Basic t_C	Improved t_C
1	$2 \cdot t_{exp}(\sigma) = 2,420$	**259**	149
2	$3 \cdot t_{exp}(\sigma) = 2,607$	261	90
3	$4 \cdot t_{exp}(\sigma) = 2,793$	263	73
4	$5 \cdot t_{exp}(\sigma) = 2,979$	265	66
5	$6 \cdot t_{exp}(\sigma) = 3,165$	267	64
6	$7 \cdot t_{exp}(\sigma) = 3,351$	269	**63**
7	$8 \cdot t_{exp}(\sigma) = 3,537$	271	64
8	$9 \cdot t_{exp}(\sigma) = 3,724$	273	65
9	$10 \cdot t_{exp}(\sigma) = 3,910$	275	67
10	$11 \cdot t_{exp}(\sigma) = 4,069$	277	69

In Table 4.2, we show performance of our protocols from Sections 4.1 (when $m = 1$) and Section 4.2 (for any m). We set $\sigma = 2048$, $\lambda = 128$ and use both basic and improved settings in the client's multiplications. Notice that t_C has minimum points at $m = 5, 6, 7, 8$ in the improved settings. However, we can see that the client's number of multiplications in the offline phase (t_P), and the server's number of multiplications (t_S), monotonically increase as m increase. Note that $t_P = t_S$ by Theorem 4.2.1. Therefore, we are interested in the lowest number t_C, where m is the smallest. In this case, we consider $m = 5$ then $t_C = 65$ and $t_S = t_P = 6 \cdot t_{exp}(\sigma) = 3,165$ with improved settings, using the protocol in Section 4.2. This can be computed with verses when $m = 1$ then $t_C = 145$ and $t_S = t_P = 2,420$ by using the protocol in Section 4.1.

Chapter 5

RSA-type Group Exponentiation

In this chapter we present protocols for delegation of exponentiation in RSA-type groups. Our first protocol in Section 5.1 consists of an efficient atomic protocol using the idea of probabilistic checks (as in Chapter 4) with security probability equal to $2^{-\lambda}$. Our second protocol in Section 5.2 improves the number of C's multiplications in the online phase over the first protocol, with the following efficiency tradeoff: it reduces the number of group multiplications by the client's during the online phase, while slightly increasing the number of group exponentiations of random exponents during the offline phase and the number of group exponentiations by the server. Both protocols satisfy the privacy and security properties and do not rely on any additional complexity assumptions. We start the presentation of the two protocols with group notations that are relevant to both protocols.

Group notations. Let p_1, p_2, p'_1, p'_2 be large primes of the same-length where $p_1 = 2p'_1+1$, $p_2 = 2p'_2 + 1$, and $n = p_1 p_2$. Let $\mathbb{Z}_n^*$ denote the set of integer coprime with n, with order $\phi(n)$, where $\phi(n) = (p_1 - 1)(p_2 - 1) = 4p'_1 p'_2$. In the group $(\mathbb{Z}_n^*, \cdot)$ $\cdot$ denotes multiplication modulo n. Let the function $F_{n,exp,k} : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$ denote the function that maps every $x \in \mathbb{Z}_n^*$ to the exponentiation of x to the k-th power modulo n where $gcd(k, \phi(n)) = 1$, in other words k is relatively prime with 4, p'_1 and p'_2. Note that the order of elements in $\mathbb{Z}_n^*$ is one element of the set $\{1, 2, p'_1, p'_2, p'_1 p'_2, 2p'_1 p'_2\}$, proven in [25] Lemma 1.

5.1 Delegation of Exponentiation in RSA-type Group

In this section we present a client-server protocol for delegated computation of group exponentiation in RSA-type groups, in the model where the adversary corrupting the server can be malicious on RSA-type groups. First: we formally state the result of the section, then describe the delegation protocol, and finally prove its correctness, security, privacy and efficiency properties.

Formal theorem statement. We show the following

Theorem 5.1.1. Let $F_{n,exp,k} : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$ defined as $F_{n,exp,k}(x) = x^k$ where $n = p_1 * p_2$, $p_1 = 2p'_1 + 1$, $p_2 = 2p'_2 + 1$, and p_1, p_2, p'_1, p'_2 be large primes with the same-length. Let σ, λ be security parameters such that $2^\lambda < p'_1, p'_2$. There exists (constructively) a client-server protocol (C, S) for delegated computation of function $F_{n,exp,k}$ which satisfies

1. δ_c-correctness, for $\delta_c = 1$;

2. ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$;

3. ϵ_p-privacy, for $\epsilon_p = 0$;

4. efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where

   - $t_F = t_{exp}(\sigma)$ in $\mathbb{Z}_n^*$;
   - $t_S = 2 \cdot t_{exp}(\sigma)$ in $\mathbb{Z}_n^*$;
   - $t_P = 2 \cdot t_{exp}(\sigma)$ with random exponents in $\mathbb{Z}_n^*$;
   - $t_C = 2 \cdot t_{exp}(\lambda) + 9$ in $\mathbb{Z}_n^*$;
   - $cc = 4$ elements in $\mathbb{Z}_n^*$ and $mc = 2$.

Formal description of protocol (C, S)**.**

Input to S: $1^\sigma, 1^\lambda$, $desc(F_{n,exp,k})$, $k \in \mathbb{Z}_n$, $\gcd(n, k) = 1$

Input to C: $1^\sigma, 1^\lambda$, $desc(F_{n,exp,k})$, $x \in \mathbb{Z}_n^*$, $k \in \mathbb{Z}_n$

Offline phase instructions:

1. C randomly chooses $u_i \in \mathbb{Z}_n^*$, for $i = 0, 1$

   C sets $v_i = u_i^k$, for $i = 0, 1$

Online phase instructions:

1. If $x = 1$ or $x^2 = 1 \mod n$

   C randomly chooses z_0, z_1 from $\mathbb{Z}_n^*$ and sends z_0, z_1 to S

C randomly chooses $b \in \{1, \ldots, 2^\lambda\}$

C sets $z_0 := x * u_0 \mod n,\ z_1 := x^b * u_1 \mod n$

C sends z_0, z_1 to S

2. S computes $w_i := z_i^k \mod n$ for $i = 0, 1$

   S sends w_0, w_1, to C

3. if $x = 1$

   C **returns** $y = 1$ and the protocol halts

   if $x^2 = 1 \mod n$

   C **returns** $y = x$ and the protocol halts

   C computes $y := w_0 * v_0^{-1}$

   C checks that $y^2 \neq 1$, also called the 'distinctness test'

   C checks that $w_1 = y^b * v_1$, also called the 'probabilistic test'

   C checks that $w_0, w_1 \in \mathbb{Z}_n^*$, also called the '$\mathbb{Z}_n^*$-membership test'

   if any one of these tests is not satisfied then

   C **returns** $\perp$ and the protocol halts

   C **returns** y

Properties of protocol (C, S)**:** *The efficiency properties* are verified by protocol inspection:

- *With respect to round complexity,* the protocol only requires one round, consisting of one message from C to S followed by one message from S to C.

- *With respect to communication complexity*, the protocol requires the transfer of 2 elements in $\mathbb{Z}_n^*$ from S to C and 2 elements in $\mathbb{Z}_n^*$ from C to S.

- *With respect to runtime complexity*, if $x = 1$ or $x^2 = 1 \mod n$, C randomly chooses two elements from $\mathbb{Z}_n^*$ and sends the elements to S. S performs 2 exponentiations. Otherwise, C performs 2 exponentiations with a random base during the offline phase, S performs 2 exponentiations and C performs 6 modular multiplications, 1 modular inversion and 2 exponentiations to a random exponent $\leq 2^\lambda$ during the online phase. In the typical setting if $\lambda = 128$, then C only performs at most 518 (or an average of 390) group multiplications in $\mathbb{Z}_n^*$ using the square and multiply algorithm in the online phase.

The correctness properties are demonstrated by observing that if C and S follow the protocol then the following holds:

- if $x = 1$ then C returns $y = 1$ which is $y = 1^k = F_{n,exp,k}(1)$

- if $x^2 = 1$ then C returns $y = x$. Note that

  - if the power of x is $2a$ (any even number in $\mathbb{Z}_n$) then $x^{2a} = (x^2)^a = 1^a = 1$

  - if the power of x is $2a + 1$ (any odd number in $\mathbb{Z}_n$) then $x^{2a+1} = x^{2a} * x = x$

  Since $gcd(k, \phi(n)) = 1$, we know that k is an odd positive integer which implies that $y = x = x^k = F_{n,exp,k}(x)$

- Otherwise, C performs 3 tests and outputs y such that $y = x^k$. All three tests always pass as follows:

  - $\mathbb{Z}_n^*$*-membership test*: since $w_i = z_i^k$, for $i = 0, 1$, and $z_i \in \mathbb{Z}_n^*$
  - *the probabilistic test:* since $w_1 = z_1^k = (x^b * u_1)^k = (x^k)^b * u_1^k = y^b * v_1$
  - *distinctness test:* since we assume if $x \neq 1$ and $x^2 \neq 1$ then $y^2 = (x^k)^2 = (x^2)^k \neq 1$ where $gcd(k, \phi(n)) = 1$.

  Thus, C never returns $\perp$, but does return y. To see that y is the correct output:

$$y = w_0 * v_0^{-1} = z_0^k * (u_0^k)^{-1} = (x * u_0)^k * u_0^{-k} = x^k.$$

The privacy against a malicious S follows similarly as in the previous protocols. We want to show that C's message to S does not leak any information about x. This message is a pair (z_0, z_1) where $z_0 = (x * u_0) \mod n$, $z_1 = x^b * u_1 \mod n$. u_0 and u_1 are uniformly and independently distributed in $\mathbb{Z}_n^*$, and thus so are z_0 and z_1. For the same reason, it satisfies the following two properties: (1) for any x, z_0 and z_1 are uniformly and independently distributed in $\mathbb{Z}_n^*$; and (2) any x, z_0 and z_1 do not leak any information about b. We will use both facts in the proof of the security property.

To prove *the security property* against a malicious S we need to compute an upper bound ϵ_s on the security probability that S convinces C to output y such that $y \neq F_{n,exp,k}(x)$. If $x = 1$ or $x^2 = 1$, C does not use the calculation of S; in other words C calculates

$F_{n,exp,k}(x) = x^k$ without using w_0, w_1 (send by S). Thus $\epsilon_s = 0$ in this case. Otherwise, we use the Observations which were considered in Chapter 4 to find an upper bound for ϵ_s. First we define the following events, similar to the previous section, with slight changes:

- $e_{1,b}$, defined as '$\exists$ exactly one b such that S's message (w_0, w_1) satisfies $w_1 = (w_0 * v_0^{-1})^b * v_1$'
- $e_{>1,b}$, defined as '$\exists$ more than one b such that S's message (w_0, w_1) satisfies $w_1 = (w_0 * v_0^{-1})^b * v_1$'.

By definition, events $e_{1,b}, e_{>1,b}$ are complementary.

In our proof of the privacy property of (C, S), we proved that for any x, C's message (z_0, z_1) does not leak any information about b. This implies that all values in $\{1, \ldots, 2^\lambda\}$ are still equally likely even when conditioning over message (z_0, z_1). Then, if the event $e_{1,b}$ is true, the probability that S's message (w_0, w_1) satisfies the probabilistic test is 1 divided by the number 2^λ of values of b that are still equally likely even when conditioning over message (z_0, z_1). Thus, the Observation 4.1.2 in precious chapter follows.

The main proof of security is to show Observation 4.1.3, which states that that *any S cannot produce values w_0, w_1 in step 2 satisfying all three checks of C in step 3 for two values of $b \in \{1, \ldots, 2^\lambda\}$ where $2^\lambda < p'_1, p'_2$*

To prove this statement we assume, towards the contradiction assume, there exist two values b_1 and $b_2 \in \{1, 2, \ldots, 2^\lambda\}$. Without loss of generality assume that $b_1 > b_2$ then

$b_1 - b_2 \in \{0, 1, \ldots, 2^\lambda - 1\}$ which satisfies all three checks. Then

$$w_1 = y^{b_1} * v_1 \text{ and } w_1 = y^{b_2} * v_1$$

$$y^{b_1} * v_1 = y^{b_2} * v_1$$

$$y^{b_1 - b_2} = 1$$

$y = w_0 * v_0^{-1}$ and C checks that $w_0, w_1 \in \mathbb{Z}_n^*$, which implies that $y \in \mathbb{Z}_n^*$. Note that the element of $\mathbb{Z}_n^*$ can have order at most $2p_1'p_2'$ by [25], and $y \in \mathbb{Z}_n^*$ which implies that by Lagrange Theorem the order of y must divide $2p_1'p_2'$. Since C checks that $y^2 \neq 1$, the order of y can be greater than or equal to p_1' or p_2'. The above calculation shows that $y^{b_1 - b_2} = 1$, but $0 \leq b_1 - b_2 < p_1', p_2'$ (because $b_1 - b_2 \in \{0, 1, \ldots, 2^\lambda - 1\}$ and $2^\lambda < p_1', p_2'$). This implies that $b_1 - b_2 = 0 \Longleftrightarrow b_1 = b_2$. Therefore, b is unique. Thus the Observation 4.1.3 follows.

The rest of the proof consists of computing an upper bound ϵ_s on the probability of event $e_{y \neq F}$. We have the following

$$\begin{aligned}
\Pr(e_{y \neq F}) &\leq \Pr((e_\perp)^c) \\
&= \Pr(e_{1,b}) \cdot \Pr((e_\perp)^c | e_{1,b}) + \Pr(e_{>1,b}) \cdot \Pr((e_\perp)^c | e_{>1,b}) \\
&= \Pr(e_{1,b}) \cdot \Pr((e_\perp)^c | e_{1,b}) \\
&\leq \Pr(e_{1,b}) \cdot \frac{1}{2^\lambda}
\end{aligned}$$

$$\leq \frac{1}{2^{\lambda}},$$

where the first inequality follows from Observation 4.1.1, the first equality follows from the definition of events $e_{1,b}, e_{>1,b}$ and the conditioning rule, the second equality follows from Observation 4.1.3, and the second inequality follows from Observation 4.1.2. We then obtain that $\epsilon_s = 2^{-\lambda}$, which concludes the proof for the security property for a single execution of (C, S). □

A Protocol Extension. The above protocol works for the group of $\mathbb{Z}_n^*$ where $n = p_1 * p_2$, $p_1 = 2p_1' + 1$, $p_2 = 2p_2' + 1$, and p_1, p_2, p_1', p_2' are large primes such that $|\mathbb{Z}_n^*| = \phi(n) = 4p_1'p_2'$. Now, we can extend the protocol (C, S) with $n = p_1 * p_2$ sand the order of the group $\mathbb{Z}_n^*$ is equal to $\phi(n) = c_1 \cdots c_l \cdot (p_1')^{\alpha_1} \cdots (p_h')^{\alpha_h}$ where $c_1, \ldots, c_l \in \mathbb{Z}_n$ and each $c_i < \lambda$ for $i = 1, \ldots, l$ and $p_1', \ldots p_h'$ are prime numbers where each $p_i > 2^{\lambda}$ for $i = 1, \ldots, h$. Then, the protocol (C, S) is similar to the previous protocol. The only changes occur when C checks in the online phase if

- the order of x is smaller than λ (i.e. c_1 or ... or c_l) then it can evaluate by itself as we show above protocol for $x = 1$ or $x^2 = 1$.

- the order of x is greater than 2^{λ} (i.e. $(p_1')^{\alpha_1}$ or ... or $(p_h')^{\alpha_h}$) then it can evaluate by running the same protocol as above.

5.2 Delegation of Exponentiation in RSA-type Group Tradeoff Runtime Performance

By directly combining the techniques in Section 5.1 with the input mixing technique from Section 4.2, we obtain a protocol for delegating exponentiations RSA - type groups. The proof of the following theorem is a direct combination of proofs of Theorem 4.2.1 and Theorem 5.1.1

Theorem 5.2.1. Let $F_{n,exp,k} : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$ be defined as $F_{n,exp,k}(x) = x^k$ where $n = p_1 * p_2$, $p_1 = 2p_1' + 1$, $p_2 = 2p_2' + 1$, and p_1, p_2, p_1', p_2' are large primes with the same length. Let σ, λ be security parameters and m be a protocol parameter where $2^{\lceil \frac{\lambda}{m} \rceil} < p_1', p_2'$. There exists (constructively) a client-server protocol (C, S) for delegated computation of the function $F_{n,exp,k}$ which satisfies

1. δ_c-correctness, for $\delta_c = 1$;
2. ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$;
3. ϵ_p-privacy, for $\epsilon_p = 0$;
4. efficiency with parameters $(t_F, t_S, t_P, t_C, cc, mc)$, where
   - $t_F = t_{exp}(\sigma)$ in $\mathbb{Z}_n^*$;
   - $t_P = t_S = (m+1) \cdot t_{exp}(\sigma)$ in $\mathbb{Z}_n^*$;
   - $t_C = 2 \cdot t_{exp,m}(\lambda/m) + 2m + 7$ in $\mathbb{Z}_n^*$;

- $cc = 4m$ elements in $\mathbb{Z}_n^*$ and $mc = 2$.

Formal description of protocol (C, S)**.**

Input to S: $1^\sigma, 1^\lambda$, $desc(F_{n,exp,k})$, $k \in \mathbb{Z}_n$ parameter 1^m

Input to C: $1^\sigma, 1^\lambda$, $desc(F_{n,exp,k})$, $x \in \mathbb{Z}_n^*$, $k \in \mathbb{Z}_n$ parameter 1^m

Offline phase instructions:

1. For $j = 0, 1, \ldots, m$

   C randomly chooses $u_j \in \mathbb{Z}_n^*$,

   C sets $v_j = u_j^k$

Online phase instructions:

1. If $x = 1$ or $x^2 = 1 \mod n$

   C randomly chooses $z_0, z_1, \ldots, z_m$ from $\mathbb{Z}_n^*$ and sends them to S

   For $i = 1, 2, \ldots, m$

   C randomly chooses $b(i) \in \{1, \ldots, 2^{\lceil \frac{\lambda}{m} \rceil}\}$

   C sets $z_0 := x * u_0 \mod n$ and $z_i := x^{b(i)} * u_i \mod n$

   C sends $z_0, z_1, \ldots, z_m$ to S

2. For $j = 0, 1, \ldots, m$

   S computes $w_j = z^k \mod n$ and sends w_j to C

3. if $x = 1$

   C **returns** $y = 1$ and the protocol halts

   if $x^2 = 1 \mod n$

   C **returns** $y = x$ and the protocol halts

   C computes $y = w_0 * v_0^{-1}$

   C checks that $y^2 \neq 1$, also called the 'distinctness test'

   For $i = 1, 2, \ldots, m$

   C checks that $w_i = y^{b(i)} * v_i$, also called the 'probabilistic test'

   C checks that $w_0, w_i \in \mathbb{Z}_n^*$, also called the '$\mathbb{Z}_n^*$-membership test'

   if any one of these tests is not satisfied then

   C **returns** $\perp$ and the protocol halts

   C **returns** y

5.3 Performance Analysis

Similarly to the previous chapter the performance of our protocols has been expressed in terms of group multiplication and is parameterized by the functions t_{exp} and $t_{exp,m}$ where $t_{exp,m} := m \cdot t_{exp}$ by definition. We have two different settings to evaluate from group exponentiations to group multiplications in our protocols:

1. **Basic setting** uses the square and multiply algorithm to evaluate group exponentiation i.e.

- $t_{exp}(\ell) = 2\ell$
- $t_{exp,m}(\ell) = 2m\ell$

2. **Improved settings** uses the improved algorithms from the literature to evaluate group exponentiation. We use the closed-form estimates in [34], as well as these in [8, 19, 33, 35] for other algorithms, claiming improvements without closed-form evaluations. By studying these we obtain that:

   - $t_{exp,G}(\ell) \approx \ell(1 + \frac{1}{\log \ell})$ using Brauer's 1939 algorithm
   - $t_{exp,G,m}(\ell) \approx \ell(1 + \frac{m}{\log \ell})$ using Yao's 1976 algorithm

The Table 5.1 below compares the performance of our protocols in Section 5.1 and Section 5.2 with non-delegated computation the client under both basic (B) and improved (I) settings.

The main takeaway from the table is that by comparing non delegated protocols with our protocols we have the following:

- our protocol in Section 5.1 reduces t_C by a multiplicative factor of about σ/λ with respect to non-delegated computation when using both basic and improved settings;
- the protocol in Section 5.2 reduces t_C by a multiplicative factor of about σ/λ' when using the improved settings (where $\lambda' = \lceil \lambda/m \rceil$ by definition);

Table 5.1: Efficiency parameters of the protocol (C, S) when m and b increases $\lambda = 128$ in worst case.

Perf	**Basic Imp.**	$F_{G,exp,g}$ **with no Delegation**	$F_{G,exp,g}$ **with Delegation**	
			Section 5.1	*Section 5.2*
t_F	B	2σ	2σ	2σ
	I	$\sigma(1+\frac{1}{\log\sigma})$	$\sigma(1+\frac{1}{\log\sigma})$	$\sigma(1+\frac{1}{\log\sigma})$
t_P	B	0	4σ	$2\sigma(m+1)$
	I	0	$\sigma(1+\frac{2}{\log\sigma})$	$\sigma(1+\frac{m+1}{\log\sigma})$
t_C	B	2σ	$4\lambda+8$	$2m\lambda'+2m+6$
	I	$\sigma(1+\frac{1}{\log\sigma})$	$2\lambda(1+\frac{1}{\log\lambda})+9$	$2\lambda'(1+\frac{m}{\log\lambda'})+2m+7$
t_S	B	0	4σ	$2\sigma(m+1)$
	I	0	$\sigma(1+\frac{2}{\log\sigma})$	$\sigma(1+\frac{m+1}{\log\sigma})$
ϵ_p	B & I	0	0	0
ϵ_s	B & I	0	$2^{-\lambda}$	$2^{-\lambda}$

In Table 5.2 we show performance of our protocols in Sections 5.1 (when $m = 1$) and Section 5.2 (for any m). We set $\sigma = 2048$, $\lambda = 128$, and use both the basic and the improved settings in the client's multiplications. Notice that t_C has a minimum point at $m = 8$ in the improved settings. However, we can see that the client's number of multiplications in the offline phase (t_P), and the server's number of multiplications (t_S), monotonically increase as m increases. Note that $t_P = t_S$ by Theorem 4.2.1. Therefore, we are interested in the lowest number, t_C, where m is the smallest. In this case we consider $m = 8$ then $t_C = 119$ and $t_S = t_P = 9 \cdot t_{exp}(\sigma) = 3,724$ with improved settings by using the protocol in Section 4.2. This can be compared to when $m = 1$, then $t_C = 302$ and $t_S = t_P = 2,420$ by using the protocol in Section 4.1.

Table 5.2: Efficiency parameters of protocols using square and multiply algorithm and Yao's algorithm

m	t_P and t_S	Square and multiply t_C	Yao's algorithm t_C
1	$2 \cdot t_{exp}(\sigma)$	**521**	302
2	$3 \cdot t_{exp}(\sigma)$	523	182
3	$4 \cdot t_{exp}(\sigma)$	525	147
4	$5 \cdot t_{exp}(\sigma)$	527	130
5	$6 \cdot t_{exp}(\sigma)$	529	124
6	$7 \cdot t_{exp}(\sigma)$	531	122
7	$8 \cdot t_{exp}(\sigma)$	533	122
8	$9 \cdot t_{exp}(\sigma)$	535	**119**
9	$10 \cdot t_{exp}(\sigma)$	537	124
10	$11 \cdot t_{exp}(\sigma)$	539	124

Chapter 6

Multiple Exponentiations: Discrete Log

In this chapter we propose batch (multiple) verification tests and apply of these tests to the delegation of client - server multiple exponentiations over Discrete Log type groups. More specifically, multiple exponentiations in the q-order subgroup G_q of $\mathbb{Z}_p^*$, where $p = 2q + 1$ and p, q are large primes. A group G_q is often used in cryptosystems that base their security on the hardness of the discrete logarithm problem or related problems.

We start by providing some useful number theory definitions and observations in Section 6.1. Then we show two batch verification of exponentiation tests in Section 6.2. Then we apply these two verification tests to two delegation of client server protocols in Section 6.3. Our first protocol, where the client does not hide its inputs from the server and the client does not have to precalculate in the offline phase is described in Section 6.3.1. Our second protocol, where the client completely hides the inputs from the server, and the client performs some calculations in the offline phase is described in Section 6.3.2. Lastly, we analyze performance in Section 6.4.

6.1 Number Theory Definitions and Properties

Let $p = 2q + 1$, for p, q primes. Recall that $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$ has a q-order subgroup of $\mathbb{Z}_p^*$, which we denote as G_q. As q is prime, G_q is cyclic. Let g be a generator. Consider the function $F_{p,q,g} : \mathbb{Z}_q \longrightarrow G_q$ defined as $F_{p,q,g}(x_i) = y_i$, for $y_i = g^{x_i} \mod p$ for all $i = 1, \ldots, m$. Here, p, q, g are assumed to be part of $desc(F_{p,q,g})$.

$t_{exp}(\ell)$ is a parameter denoting the number of group multiplications used to compute an exponentiation of a group value to an ℓ-bit exponent. By $t_{m,exp}(\ell)$ we denote a parameter for the number of group multiplications used to compute m exponentiations of the same group value (also called *fixed-base exponentiations*) to m arbitrary ℓ-bit exponents. By $t_{prod,m,exp}(\ell)$ we denote a parameter for the number of group multiplications used to compute a product of m exponentiations of (possibly different) group values to m arbitrary ℓ-bit exponents.

Definition. An integer $y \in \mathbb{Z}_p^*$ is a *quadratic residue modulo p* if there exists $r \in \mathbb{Z}_p^*$ such that $r^2 = y \mod p$. An integer $y \in \mathbb{Z}_p^*$ is a *quadratic non residue modulo p* if there does not exist an $r \in \mathbb{Z}_p^*$ such that $r^2 = y \mod p$.

Euler's theorem states that for any odd prime p and any $a \in \mathbb{Z}_p^*$, a is a quadratic residue modulo p if and only if $a^{(p-1)/2} = 1 \mod p$. For the considered type of integers $p = 2q+1$, with p, q primes, this implies the following

Observation 6.1.1. G_q is the set of quadratic residues in $\mathbb{Z}_p^*$.

The above observation also implies that $\mathbb{Z}_p^*$ can be partitioned into 2 equal-size sets: the set of quadratic residues modulo p (i.e., G_q) and the set of quadratic non residues modulo p (i.e., $\mathbb{Z}_p^* \setminus G_q$).

6.2 Batch Verification of Discrete Log Exponintiation

GIVEN: g a generator of G_q, batch instance: $(z_1, w_1), \ldots, (z_n, w_n)$ with $z_i \in \mathbb{Z}_q$ and values $t_1, \ldots, t_n$ which is calculated as $t_i = w_i^{\frac{q+1}{2}}$ (to check quadratic residue property) for each $i = 1, \ldots, n$. Note that $t_1, \ldots, t_n$ are needed for Small Exponent Test and are not necessary for Random Subset Test.

- **Random Subset (RS) Test:**

  CHECK: for all $i \in \{1, \ldots, n\}$: $w_i \in \mathbb{Z}_p^*$ and $w_i = g^{z_i}$

  1. *Membership Check:* For $i = 1, \ldots, n$,

     if $w_i \notin \mathbb{Z}_p^*$ then **return:** $\perp$ and halts

  2. Repeat the following atomic test, independently λ times and accept iff all subset tests accept:

     ATOMIC RANDOM SUBSET TEST

     (a) For each $i = 1, \ldots, n$ pick $b_i \in \{0, 1\}$ at random

     (b) Let $S = \{i : b_i = 1\}$

(c) Compute $z = \sum_{i \in S} z_i \mod q$ and $w = \prod_{i \in S} w_i$

(d) if $g^z \neq w$

return: $\perp$ and protocol halts

return: $w_1, \ldots, w_n$

- **Small Exponents Test:**

CHECK: for all $i \in \{1, \ldots, n\}$: $w_i \in G_q$ and $w_i = g^{z_i}$

1. *Membership Check:* For $i = 1, \ldots, n$,

if t_i or $w_i \notin \mathbb{Z}_p^*$ then **return:** $\perp$ and halts

if $t_i^2 \neq w_i \mod p$ then **return:** $\perp$ and halts

2. SE:

(a) Pick $s_1, \ldots, s_n \in \{0,1\}^\lambda$ at random

(b) Compute $z = \sum_{i=1}^{n} z_i s_i \mod q$ and $w = \prod_{i=1}^{n} w_i^{s_i}$

(c) if $g^z \neq w$

return: $\perp$ and protocol halts

return: $w_1, \ldots, w_n$

Note that the above batch verification test is an improvement of the batch verification test described in [6]. In [6] authors assume the existence of a membership test, however we are showing how efficiently to check the membership test. Next, we show security requirements for the above verification test.

Theorem 6.2.1. *Given a group G_q of prime order group q and generator g of G_q. Suppose $(z_1, w_1), \ldots, (z_n, w_n)$ is an incorrect batch instance of the batch verification function $F_{p,q,g}$. Then the*

1. *ATOMIC RANDOM SUBSET TEST accepts batch instance with probability* $1/2$
2. *SMALL EXPONENT TEST accepts batch instance with probability* $2^{-\lambda}$

Proof. ATOMIC RANDOM SUBSET TEST: Suppose $(z_1, w_1), \ldots, (z_n, w_n)$ is an incorrect batch instance of the batch verification problem. Meaning that there exist $i \in \{1, \ldots, n\}$ such that $w_i \neq g^{z_i}$. Let $\alpha_i = \frac{w_i}{g^{z_i}} \mod p$. Since S can be malicious it may send $w_i \notin G_q$. Let us look at three cases for α_i:

1. if $w_i \notin \mathbb{Z}_p^*$ then the Membership Check returns $\perp$ and protocol halts.
2. if $w_i \in \mathbb{Z}_p^* \backslash\ G_q$ then we prove that the protocol does not halt the error (i.e. $F_{p,q,g}(z_i) \neq w_i$) with probability at most $1/2$ similarly as [6]. By assumption there exist an i such that $\alpha_i \neq 1$, without loss of generality assume that $i = 1$. Now, suppose that the test accepts a particular subset M. Then it must be $\prod_{i \in M} g^{x_i} \mod p = \prod_{i \in M} w_i \mod p$ this implies that $\prod_{i \in M} \frac{w_i}{g^{x_i}} = 1 \mod p \iff \prod_{i \in M} \alpha_i \mod p = 1$. Now suppose that $T \subseteq \{2, \ldots, n\}$. Then note that

$$\prod_{i \in T} \alpha_i = 1 \Longrightarrow \prod_{i \in T \cup \{1\}} \alpha_i \neq 1$$

So if the test succeeds on $M = T$ then it must fail on $M = T \cup \{1\}$. This means the test must fail on at least half the tests M.

3. if $w_i \in G_q$ then the verification test follows from [6] and by assumption, $w_i \neq z_i^e$ then $\alpha_i \neq 1$.

SMALL EXPONENT TEST: Suppose $(z_1, w_1), \ldots, (z_n, w_n)$ is an incorrect batch instance of the batch verification problem. Meaning that there exists an $i \in \{1, \ldots, n\}$ such that $w_i \neq g^{z_i}$. Then, we have the following two cases:

1. if $w_i \notin G_q$ then the Membership Check returns $\perp$ and protocol halts because of Observation 6.1.1

2. if $w_i \in G_q$ then verification tests follows from [6]. □

The next theorem describes the number of operations required for each test:

Theorem 6.2.2. *Let p, q be large, same-length primes, such that $p = 2q+1$. Let σ be the computational security parameter associated with the subgroup G_q of the group $\mathbb{Z}_p^*$ where G_q is a prime order group with generator g. Let λ be a statistical security parameter. Then the cost of the computation of function $F_{p,q,g}$ on n inputs with*

1. *RANDOM SUBSET TEST is $\frac{n\lambda}{2} + t_{exp,\lambda}(\sigma)$ multiplications in $\mathbb{Z}_p^*$*

2. *SMALL EXPONENT TEST is $2n + t_{prod,n,exp}(\lambda) + t_{exp}(\sigma)$ multiplications in $\mathbb{Z}_p^*$*

Proof. These two batch verification tests are very similar to the test described in [6]. The only difference is that each test has *Membership Check* in first step. Thus, in this theorem, we need to see the cost of *Membership Check* and then add it to cost of verification test discussed in [6].

- **Random Subset Test:** The Atomic Random Subset requires about $n/2 + t_{exp}(\sigma)$ multiplications in $\mathbb{Z}_p^*$ as described in [6]. However, in order to lower the error to $2^{-\lambda}$, the atomic protocol has to be repeated λ times. Thus, the test performs $\frac{n\lambda}{2} + t_{exp,\lambda}(\sigma)$ multiplications in $\mathbb{Z}_p^*$. Note that to check whether $w_i \notin \mathbb{Z}_p^*$, the test only needs to check that these numbers are in $[1, p-1]$ (i.e. $w_i > 0$ and $w_i < p$), which requires time linear in σ. This is a lower-order operation when compared to multiplication in $\mathbb{Z}_p^*$, and we can ignore it in our calculations.

- **Small Exponent Test:** The test performs m exponentiations to λ-bit exponents, 1 exponentiation to a σ-bit exponent and $2m$ multiplications. Note that to check whether t_i or $w_i \notin \mathbb{Z}_p^*$, the test only needs to check that these numbers are in $[1, p-1]$, which we can ignore it in our calculations.

□

Note that if we are working with the group G_q which is $\mathbb{Z}_p^*$ and g is a generator of G_q then the Random Subset Test works as well as group G_q because G_q is cyclic group and it is shown in [6] that Random Subset Test works in cyclic group. The difference is that we do not assume that all $w_i \in G_q$ for all $i = 1, \ldots, n$, but we show in *Membership Check* that $w_i \in G_q$.

6.3 From Batch Verification to Batch Delegation

In this section, we propose two client-server protocols for secure delegated computation of batch exponentiations in a group G_q where q is prime and g is generator of G_q.We work in the model where the server can be malicious. Both protocols use two generic constructions of batch verification tests, described in the previous section. In the first protocol, the client does not hide the input from the server and the client does not have to precalculate in the offline phase. In the second protocol, the client hides the input from the server, but then has to do precalculation in the offline phase. In both protocols, we first describe the formal description of the protocol (C, S) and then prove the efficiency, correctness, privacy (if it is applicable), and security requirements.

6.3.1 Protocol 1: Batch Delegation with No Input Privacy or Offline Phase

Our Protocol 1 satisfies the following:

Theorem 6.3.1. Let p, q be large primes such that $p = 2q+1$, let σ be the computational security parameter associated with the q-order subgroup G_q of $\mathbb{Z}_p^*$, and let λ be a statistical security parameter. There exists (constructively) a client-server protocol (C, S) for batch delegation of the computation of $F_{p,q,g}$ on n inputs, which satisfies

1. δ_c-correctness, for $\delta_c = 1$;

2. ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$;

3. $(t_F, t_S, t_P, t_C, cc, mc)$-efficiency, where

   - t_F is $= t_{exp,n}(\sigma)$
   - t_S is $= t_{exp,2n}(\sigma)$
   - $cc = n$ elements in $\mathbb{Z}_q$ + $2n$ elements in $\mathbb{Z}_p^*$
   - $t_P = 0$ and $mc = 2$.
   - t_C depends on which verification test C uses:
     - Random Subset Test: t_C is $\frac{n\lambda}{2} + t_{exp,\lambda}(\sigma)$ multiplications in $\mathbb{Z}_p^*$
     - Small Exponent Test: t_C is $2n + t_{prod,n,exp}(\lambda) + t_{exp}(\sigma)$ multiplications in $\mathbb{Z}_p^*$

Formal description of protocol (C, S)**.**

Input to S and C: 1^σ, 1^λ $desc(F_{G,exp,g})$, $x_1, \ldots, x_n \in \mathbb{Z}_q$, $g \in G_q$

1. C sends $x_1, \ldots, x_n$ to S
2. For $i = 1, \ldots, n$

   S computes $y_i := g^{x_i}$, $t_i := y_i^{\frac{q+1}{2}} \mod p$

   S sends $y_1, \ldots, y_n, t_1, \ldots, t_n$ to C
3. C runs one of the following verification tests for batch instance $(x_1, y_1) \ldots, (x_n, y_n)$ and values $t_1, \ldots, t_n$.

(a) Random Subset (RS) Test (does not need the values $(t_1, \ldots, t_n)$)

(b) Small Exponent (SE) Test

if the chosen test from above is not satisfied then

C **returns** $\perp$ and the protocol halts

C **returns** $(y_1, \ldots, y_n)$

Properties of protocol (C, S)**:** *The efficiency properties* are verified by protocol inspection

- *Round complexity:* the protocol only requires one round, consisting of one message from C to S followed by one message from S to C.
- *Communication complexity:* the protocol requires the transfer of n elements from C to S in $\mathbb{Z}_q$ and $2n$ elements from S to C in $\mathbb{Z}_p^*$.
- *Runtime complexity:* S performs $2n$ exponentiations to σ-bit exponents in G_q C's performance depends on what test is used. The cost of the tests was described in Theorem 6.2.2.

The *correctness* property follows by showing that if C and S follow the protocol, C always outputs $y_i = g^{x_i}$ for all $i = 1, \ldots, n$. First, we show the Membership Checks performed by C are always passed. In The Membership Check in Random Subset Test, C checks if $y_i \in \mathbb{Z}_p^*$ by verifying y_i are > 0 and $< p$. This is true because y_i is computed by S as $g^{x_i} \mod p$, for $i = 1, \ldots, n$, and g is a generator of group G_q and $x_i \in \mathbb{Z}_q$. In The

Membership Check in the Small Exponent Test and Test, C checks if $y_i, t_i \in \mathbb{Z}_p^*$ ($y_i = t_i^2 \mod p$) is also passed, since

$$t_i^2 = \left(y_i^{\frac{q+1}{2}}\right)^2 = y_i^{q+1} = g^{x_i(q+1)} \equiv g^{x_i} = y_i \mod p$$

for y_i and t_i computed by S as $y_i := g^{x_i}$ and $t_i := y_i^{\frac{q+1}{2}} \mod p$. Lastly, y_i is the correct output since $y_i = g^{x_i}$ computed by S for all $i = 1, \ldots n$.

The *security* property against a malicious S we need to compute an upper bound ϵ_s on the probability that S convinces C to output a y_i such that $y_i \neq F_{G,exp,g}(x)$ for some $i = 1, \ldots, n$. By Theorem 6.2.1 in the previous section, we showed that for

- ATOMIC RANDOM SUBSET TEST satisfies $\epsilon_s = 1/2$, i.e $(x_1, y_1), \ldots, (x_n, y_n)$ is an incorrect batch instance and C does not return $\perp$. However for the **Random Subset Test** , C repeats ATOMIC RANDOM SUBSET TEST λ times. Thus $\epsilon_s = 2^{-\lambda}$.

- **Small Exponent Test** satisfies $\epsilon_s = 2^{-\lambda}$. □

6.3.2 Protocol 2: Batch Delegation with Input Privacy and Offline Phase

Our Protocol 2 satisfies the following:

Theorem 6.3.2. Let p, q be large primes such that $p = 2q+1$. Let σ be the computational security parameter associated with the q-order subgroup G_q of $\mathbb{Z}_p^*$, and let λ be a statistical

security parameter. There exists (constructively) a client-server protocol (C, S) for batch delegation of the computation of function $F_{p,q,g}$ on n inputs, which satisfies

1. δ_c-correctness, for $\delta_c = 1$;
2. ϵ_s-security, for $\epsilon_s = 2^{-\lambda}$;
3. ϵ_p-privacy, for $\epsilon_p = 0$;
4. $(t_F, t_S, t_P, t_C, cc, mc)$-efficiency, where
   - t_F is $= t_{exp,n}(\sigma)$
   - t_S is $= t_{exp,2n}(\sigma)$
   - $cc = n$ elements in $\mathbb{Z}_q$ + $2n$ elements in $\mathbb{Z}_p^*$
   - $t_P = 0$ and $mc = 2$.
   - t_C depends on which verification test C uses:
     - Random Subset Test: t_C is $n(1 + \frac{\lambda}{2}) + t_{exp,\lambda}(\sigma)$ multiplications in $\mathbb{Z}_p^*$
     - Small Exponent Test: t_C is $3n + t_{prod,n,exp}(\lambda) + t_{exp}(\sigma)$ multiplications in $\mathbb{Z}_p^*$

Formal description of protocol (C, S).

Input to S: 1^σ, 1^λ, $desc(F_{G,exp,g})$, $g \in G_q$

Input to C: 1^σ, 1^λ, $desc(F_{G,exp,g})$, $x_1, \ldots, x_n \in \mathbb{Z}_q$, $g \in G_q$

Offline phase instructions:

1. For $i = 1, \ldots, n$

   C randomly chooses $u_i \in \mathbb{Z}_q$ and sets $v_i := g^{u_i}$

Online phase instructions:

1. For $i = 1, \ldots, n$

   C sets $z_i := (x_i - u_i) \mod q$

   C sends $z_1, \ldots, z_n$ to S

2. For $i = 1, \ldots, n$

   S computes $w_i := g^{z_i}$, $t_i := w_i^{\frac{q+1}{2}} \mod p$

   S sends $w_1, \ldots, w_n, t_1, \ldots, t_n$ to C

3. C checks correctness of w_i for each $i = 1, \ldots, n$ for the given $t_1, \ldots, t_n$ and with batch instance $(z_1, w_1) \ldots, (z_n, w_n)$ to one of the following three tests:

   (a) Random Subset (RS) Test (it does not need $(t_1, \ldots, t_n)$)

   (b) Small Exponent (SE) Test

   if the chosen test is not satisfied then

   C **returns** $\perp$ and the protocol halts

   For $i = 1, \ldots, n$

   C sets $y_i := w_i \cdot v_i \mod q$

   C **returns** $(y_1, \ldots, y_n)$

Properties of protocol (C, S)**:** *The efficiency properties* are verified by protocol inspection.

- *Round complexity:* the protocol only requires one round, consisting of one message from C to S followed by one message from S to C.
- *Communication complexity:* the protocol requires the transfer of $2n$ elements from S to C and n elements from C to S in $\mathbb{Z}_q$.
- *Runtime complexity:* During the offline phase C performs n exponentiations in base g with random σ-bit exponents. Note that known-base exponentiations can be executed faster than unknown-base exponentiation using pre-computation techniques [8, 19]. During the online phase, S performs $2n$ exponentiations to σ-bit exponents in $\mathbb{Z}_p^*$. C's performance depends on what test is used. The cost of it described in Theorem 6.2.2. If the batch verification test is successfully passed, C performs n more group multiplications in order to get y_i in $\mathbb{Z}_p^*$.

The *correctness* property follows by showing that if C and S follow the protocol, C always outputs $y_i = g^{x_i}$ for all $i = 1, \ldots, n$. The Membership Check, performed by C are always passed, which follows from a similar argument as given Protocol 1. y_i is the correct output, since

$$y_i = w_i \cdot v_i = g^{z_i} \cdot g^{u_i} = g^{x_i - u_i} \cdot g^{u_i} = g^{x_i}$$

for w_i computed by S as $w_i := g^{z_i}$ which implies that $y_i = F_{p,q,g}(x_i)$ for all $x_i \in \mathbb{Z}_q$,

$i = 1, \ldots, n$.

The *privacy* property of the protocol against a malicious S follows by observing that C's only message to S does not leak any information about x_i for all $i = 1, \ldots, n$. This message is $(z_1, \ldots, z_n)$ where $z_i = (x_i - u_i) \mod q$. The values $u_1, \ldots, u_n$ are uniformly and independently distributed in $\mathbb{Z}_q$,

thus so are $z_1, \ldots, z_n$.

The *security* property against a malicious S requires an upper bound ϵ_s on the security probability that S convinces C to output a y_i such that $y_i \neq F_{p,q,g}(x)$ for some $i = 1, \ldots, n$. Note that the calculation of the correct y_i (i.e. $y_i = F_{p,q,g}(x_i)$) depends on w_i, sent by S. ϵ_s depends on the probability that C accepts an incorrect batch instance $(z_1, w_1), \ldots, (z_n, w_n)$. With similar reasoning to that given in the previous subsection, shows that the security probability is $\epsilon_s = 2^{-\lambda}$. □

6.4 Performance Analysis and Conclusions

The performance of our protocols has been expressed in terms of group multiplications, and parameterized by functions t_{exp}, $t_{m,exp}$, $t_{prod,m,exp}$. For a more explicit evaluation, one can set these 3 parameter functions using a textbook/definition algorithm or an improved algorithm from the literature (we use the closed-form estimates in [34]. See also [8, 19, 33, 35] for other algorithms claiming improvements without closed-form evaluations).

In Table 6.1 we listed all parameters of the two protocols described in Section 6.3 for secure batch delegation of exponentiations over the discrete log type groups to a single

Table 6.1: Efficiency parameter of Protocol 1 and 2 for the function $F_{p,q,g}(x_i) = g^{x_i}$ for all $i = 1, \ldots, n$

EP	Protocol 1	Protocol 2
t_F	$t_{n,exp}(\sigma)$	$t_{n,exp}(\sigma)$
t_P	0	$t_{n,exp}(\sigma)$
t_C	**RS:** $\frac{n\lambda}{2} + t_{exp,\lambda}(\sigma)$ **SE:** $2n + t_{prod,n,exp}(\lambda) + t_{exp}(\sigma)$	**RS:** $n(\frac{\lambda}{2} + 1) + t_{exp,\lambda}(\sigma)$ **SE:** $2n + t_{prod,n,exp}(\lambda) + t_{exp}(\sigma)$
t_S	**RS:** $t_{n,exp}(\sigma)$ **SE:** $t_{2n,exp}(\sigma)$	**RS:** $t_{n,exp}(\sigma)$ **SE:** $t_{2n,exp}(\sigma)$
ϵ_p	NA	0
ϵ_s	$2^{-\lambda}$	$2^{-\lambda}$

(and possibly malicious) server for the function $F_{p,q,g}(x_i) = g^{x_i}$, $i = 1, \ldots, n$.

Using the textbook square-and-multiply algorithm, we have that $t_{exp}(\ell) = 2\ell$ multiplications (and, on average, 1.5ℓ multiplications for a random exponent). An improved algorithm implies the setting $t_{exp}(\ell) = \ell \cdot (1 + 1/(\log \ell))$ for any $\alpha > 0$. $t_{n,exp}(\ell)$, by definition, can be set as $n \cdot t_{exp}(\ell)$ multiplications. An improved algorithm suggests $t_{n,exp}(\ell) = \ell \cdot (1 + n/(\log \ell))$. As for $t_{prod,n,exp}(\ell)$, by definition, it can be set as $t_{n,exp}(\ell) + 2n$ multiplications; an improved algorithm implies the value $\ell \cdot (1 + n/(\log n + \log \ell))$.

In Tables 6.2 and 6.3 is the concrete performance of protocol 1 and protocol 2 using batch verification tests Random Subset (RS) and Small Exponent (SE). In Table 6.2, we set $\sigma = 1024$ and $\lambda = 128$ (i.e. security probability will be $\epsilon_s = 2^{-128}$). In Table 6.3, we set $\sigma = 2048$ and $\lambda = 128$. We count the number of multiplications by the client and compare with the naive batch test. The test is not naively implemented because we used an improved algorithm as described above.

- In Table 6.2 observe that the RS test is actually worse than naive for $n \leq 487$, and

Table 6.2: C's online multiplication for increasing value n with $\epsilon_s = 2^{-128}, \lambda = 128, \sigma =$ 1024-bits

n	**Naive**	**Protocol 1**		**Protocol 2**	
		RS	**SE**	**RS**	**SE**
5	1,536	20,068	1,333	20,073	1,338
10	2,048	20,388	1,398	20,398	1,408
50	6,144	22,948	1,860	22,998	1,910
100	11,264	26,148	2,392	26,248	2,492
200	21,504	32,548	3,402	32,748	3,602
487	50,892	50,916	6,142	51,403	6,629
1,000	103,424	83,748	10,798	84,748	11,798
5,000	513,024	339,748	44,436	344,748	49,436
10,000	1,025,024	659,748	84,346	669,748	94,346

the SE test is the best, as expected. The factor of improvement increases with n: at $n = 10$ we do about 1.45 times better than naive (using SE); at $n = 487$, about 8 times better (using SE); at $n = 5000$, about 10.5 times better (using SE)and about 1.5 times better (using RS).

- In Table 6.3 observe that the RS test is actually worse than naive for $n \leq 306$, and the SE test is the best, as expected. The factor of improvement increases with n; at $n = 10$ we do about 1.55 times better than naive (using SE); at $n = 500$ we do about 12 times better (using SE), and at $n = 5000$ we do about 20 times better (using SE) and about 2.55 times better (using RS).

Table 6.3: C's online multiplication for increasing value n with $\epsilon_s = 2^{-128}, \lambda = 128, \sigma =$ 2048-bits

n	**Naive**	**Protocol 1**		**Protocol 2**	
		RS	**SE**	**RS**	**SE**
5	2,978	39,817	2,440	39,822	2,445
10	3,909	40,137	2,506	40,147	2,516
50	11,357	42,697	2,968	42,747	3,018
100	20,666	45,897	3,500	45,997	3,600
200	39,284	52,297	4,510	52,497	4,710
306	59,019	59,081	5,541	59,387	5,847
500	95,138	71,497	7,370	71,997	7,870
1,000	188,229	103,497	11,906	104,497	12,906
5,000	932,957	359,497	45,543	364,497	50,543
10,000	1,863,866	679,497	85,454	689,497	95,454

Chapter 7

Multiple Exponentiations: RSA

In this chapter we propose batch verification tests and the application of these tests to the delegation of client - server exponentiations over RSA type groups. More specifically in the group $\mathbb{Z}_N^*$, where $N = pq$ with p, q large primes of the form $p = 2p_1 + 1, q = 2q_1 + 1$, for primes p_1, q_1. We proceed similarly to the previous chapter for secure batch verification test and then application of those verification tests to delegation of client-server protocols where there is only one server in the protocol and it can be any malicious.

We start by providing useful number theory definitions and observations in Section 7.1. Then, we show two batch verification of exponentiation tests in Section 7.2 and apply these two verification tests to two delegation protocols in Section 7.3. Our first protocol, where the client does not hide inputs from the server and the client does not have to precalculate in the offline phase, is described in Section 7.3.1. Our second protocol, where the client completely hides inputs from the server and the client performs some calculations in the offline phase, is described in Section 7.3.2. Lastly, we show the performance analysis in Section 7.4.

7.1 Number Theory Definitions and Properties

Definition. For any prime p and value $a \in \mathbb{Z}_p^*$, the *Legendre symbol* $(a|p)$ of $a \mod p$ is defined as $+1$ if a is a quadratic residue modulo p and -1 otherwise.

Using Euler's theorem the Legendre symbol can be computed by one exponentiation modulo p. Exponentiation is expensive.

Let $N = pq$, for primes p, q. Let see some facts which we use them later:

- For any integer $a \in \mathbb{Z}_N^*$, we know that v is a quadratic residue modulo c if and only if $(v|b) = 1$ for any prime b dividing n (see, e.g., [32]).
- For any $a \in \mathbb{Z}_N^*$, the *Jacobi symbol* $(a|N)$ of $a \mod N$ is defined as equal to $(a|p) \cdot (a|q)$. Thus,
  - if $(a|N) = -1$ then a is a quadratic non residue modulo N, as it is a quadratic non residue modulo at least one of the primes dividing N.
  - if $(a|N) = 1$ then no efficient algorithm is known to compute whether a is a quadratic residue or non residue modulo c.
- For any $a_1, a_2 \in \mathbb{Z}_N^*$, let $\sim_N$ be the relation defined as

  $a_1 \sim_N a_2 =$ *the quadratic residuosity of* $a_1 a_2 \mod N$. We note that $\sim_N$ is an *equivalence relation* and the set of quadratic residues modulo m is an equivalence class for this relation;

- for any $a \in \mathbb{Z}_N^*$, the set of integers $\{ar \,|\, r \text{ is a quadratic residue } \mod N\}$ is an *equivalence class for this relation.* Then $\mathbb{Z}_N^*$ is divided by relation $\sim_N$ into 4 equal-size equivalence classes:
  - one is the class of *quadratic residues modulo N,*
  - one is the class of quadratic non-residues modulo N with Jacobi symbol +1,
  - the remaining 2 classes contain quadratic non-residues modulo N with Jacobi symbol -1.

Now let p, q, p_1, q_1 be large, same-length, primes, such that $p = 2p_1 + 1$, $q = 2q_1 + 1$, and let $N = pq$. Let $N = pq$, and let $\mathbb{Z}_N^*$ denote the set of integers coprime with N. Note that the order of $\mathbb{Z}_N^*$ is $\phi(N) = (p-1)(q-1) = 4p_1q_1$. For any e such that $\gcd(e, \phi(N)) = 1$, define the function $F_{N,e} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ such that as $F_{N,e}(x) = x^e \mod N$.

Next, we prove two observations that are critical in proving the security of our batch verification tests for the function $F_{N,e}$. The following observations follows from [25] Lemma 1:

Observation 7.1.1. Let p, q, p_1, q_1 be large, same-length, primes, such that $p = 2p_1 + 1$, $q = 2q_1 + 1$, and let $N = pq$. There are no integers in $\mathbb{Z}_N^*$ of order 4.

Consider the following 3 facts.

1. When N has this special form, $-1 \mod N$ is a quadratic non residue modulo N with Jacobi symbol $(-1|N) = (-1|p) \cdot (-1|q) = (-1) \cdot (-1) = +1$.

2. Note that by Lemma 1 of [31], every quadratic residue mod N has a square root with Jacobi symbol 1 and a square root with Jacobi symbol -1.

3. Note that if t is a square root of $w \mod N$ then $-t$ is also a square root of $w \mod N$.

By combining these 3 facts, we obtain that

- every quadratic residue mod N has *4 square roots modulo* N, which can be written as $t_{-1}, t_{+1}, -t_{-1} \mod N, -t_{+1} \mod N$. As seen by a case analysis, each of the roots is in a different $\sim_N$ equivalence class. Notation, $t_{\pm 1}$ depends what is Jacobi symbol is (e.g. if Jacobi symbol is $-1 \mod N$ then we have t_{-1})

  - For integer 1, its square roots modulo N can be further characterized as $1, -1, t, -t \mod N$, for some $t \in \mathbb{Z}_N^*$. Note that

    1. 1 is the root in the class of quadratic residues modulo N with Jacobi symbol 1 (because $1^2 = 1$)
    2. -1 is the root in the class of non quadratic residues modulo N with Jacobi symbol 1 (by above facts.)
    3. t and $-t$ are both quadratic non residue modulo N with Jacobi symbol -1 (by above facts) and each of these roots is in a different $\sim_N$ equivalence class.

Now, consider the values $x, y \in \mathbb{Z}_N^*$ and let $\alpha = y/x^e \mod N$ and $\alpha \neq 1$.

Assume α has order 2; that is, $\alpha^2 = 1 \mod N$. Recall the above characterization that square roots of $1 \mod N$ can be written as $1, -1, t, -t \mod N$. Since $\alpha \neq 1$ then α is a **quadratic non-residue** $\mod N$. Since $xy = \alpha x^{e+1} \mod N$ and e is odd, then xy is also **not a quadratic residue modulo** N, which proves the following.

Observation 7.1.2. Let p, q, p_1, q_1 be large, same-length, primes, such that $p = 2p_1 + 1$, $q = 2q_1 + 1$. Also, let $N = pq$, $\phi(N) = (p-1)(q-1)$, $x, y \in \mathbb{Z}_N^*$, and $e \in \mathbb{Z}_{\phi(N)}$ such that $\gcd(e, \phi(N))$=1. If $\alpha = y/x^e \mod N$ has order 2 and $\alpha \neq 1$ then $xy \mod N$ is a quadratic non residue modulo N.

7.2 Batch Verification Tests of Exponentiations Over RSA Type Groups

GIVEN: batch instance $(z_1, w_1), \ldots, (z_n, w_n)$ with $z_i \in \mathbb{Z}_N^*$ and values $t_1, \ldots, t_n$ are calculated as $t_i = z_i^{\frac{e+1}{2}}$ for each $i = 1, \ldots, n$. Note that $t_1, \ldots, t_n$ are needed for the Small Exponent Test and are not necessary for the Random Subset Test.

- **Random Subset Test(RST):**

  CHECK: for all $i \in \{1, \ldots, n\}$: $w_i \in \mathbb{Z}_N$ and $w_i = z_i^e$

  1. *Membership Check:* For $i = 1, \ldots, n$,

     if $w_i \notin \mathbb{Z}_N$ then **return** $\perp$ and halt

  2. Repeat the following atomic test, independently λ times and accept iff all subset tests accept:

ATOMIC RANDOM SUBSET TEST

(a) For each $i = 1, \ldots, n$ pick $b_i \in \{0, 1\}$ at random

(b) Let $S = \{i : b_i = 1\}$

(c) Compute $z = \prod_{i \in S} z_i \mod N$ and $w = \prod_{i \in S} w_i \mod N$

(d) if $z^e \neq w$

return: $\perp$ and protocol halts

return: $(w_1, \ldots, w_n)$

- **Small Exponents Test (SET):**

CHECK: for all $i \in \{1, \ldots, n\}$: $w_i = z_i^e$

1. *Membership Check:* For $i = 1, \ldots, n$,

   if t_i or $w_i \notin \mathbb{Z}_N$ then **return:** $\perp$ and halts

   if $t_i^2 \neq z_i w_i \mod N$ then **return:** $\perp$ and halts

2. SE:

   (a) Pick $s_1, \ldots, s_n \in \{0, 1\}^\lambda$ at random

   (b) Compute $z = \prod_{i=1}^n z_i^{s_i} \mod N$ and $w = \prod_{i=1}^n w_i^{s_i} \mod N$

   (c) if $z^e \neq w$

   return: $\perp$ and protocol halts

   return: $(w_1, \ldots, w_n)$

Note that above Batch verification Test for Small Exponent Test is an improvement of the batch verification test described in [21]. In [21] authors check that $\alpha z_i^e = w_i$ for some element $\alpha \in \mathbb{Z}_N^*$ of order ≤ 2, however we show that $z_i^e = w_i$ (i.e. $\alpha = 1$). In addition we are adding the Random Subset Test, as shown in the last chapter.

Theorem 7.2.1. *Let p, q, p_1, q_1 be large, same-length, primes, such that $p = 2p_1 + 1$, $q = 2q_1 + 1$, let $N = pq$, and let λ be a statistical security parameter. Suppose $(z_1, w_1), \ldots, (z_n, w_n)$ is an incorrect batch instance of the batch verification function $F_{N,e}$. Then the*

1. *ATOMIC RANDOM SUBSET TEST accepts the batch instance with probability* $1/2$
2. *SMALL EXPONENT TEST accepts the batch instance with probability* $2^{-\lambda}$

Proof. ATOMIC RANDOM SUBSET TEST: Suppose $(z_1, w_1), \ldots, (z_n, w_n)$ is an incorrect batch instance of the batch verification problem. Meaning that there exists $i \in \{1, \ldots, n\}$ such that $w_i \neq z_i^e$. Let $\alpha_i = \frac{w_i}{z_i^e} \mod N$. Since S can be malicious it may send $w_i \notin \mathbb{Z}_N^*$. Let us look at three cases for α_i:

1. if $w_i \notin \mathbb{Z}_N$ then the verification test returns $\perp$ and protocol halts (no need to calculate α_i)
2. if $w_i \in \mathbb{Z}_N \backslash\, \mathbb{Z}_N^*$ then w_i must be a multiple of p_1 or p_2, but since $z_i^e \in \mathbb{Z}_N^*$ then $gcd(z_i^e, N) = 1$. Thus z_i^e does not have multiple of p_1 or p_2. This implies that $\alpha_i \neq 1$.

3. if $w_i \in \mathbb{Z}_N^*$ and by assumption $w_i \neq z_i^e$ then $\alpha_i \neq 1$.

without loss of generality assume that $i = 1$, meaning $\alpha_1 \neq 1$ by the above explanation. Now, suppose the test accepts on a particular subset, M. Then, it must be $\prod_{i \in M} z_i^e \mod N = \prod_{i \in M} w_i \mod N$ which implies that $\prod_{i \in M} \frac{w_i}{z_i^e} = 1 \iff \prod_{i \in M} \alpha_i = 1$. Now suppose that $T \subseteq \{2, \ldots, n\}$. Then note that

$$\prod_{i \in T} \alpha_i = 1 \Longrightarrow \prod_{i \in T \cup \{1\}} \alpha_i \neq 1$$

So if the test succeeds on $M = T$ then it must fail on $M = T \cup \{1\}$. This means that the test must fail on at least half the tests M. □

SMALL EXPONENT TEST: Suppose $(z_1, w_1), \ldots, (z_n, w_n)$ is an incorrect batch instance of the batch verification problem. Meaning that there exists $i \in \{1, \ldots, n\}$ such that $w_i \neq z_i^e$. For all $i = 1, \ldots, n$, define $\alpha_i = w_i/(z_i)^e \mod N$. We obtain that $\epsilon_s = 2^{-\lambda}$ as a consequence of these 5 claims:

1. for all $i = 1, \ldots, n$, if $w_i \notin \mathbb{Z}_N$ then the Membership Test outputs $\perp$

2. for all $i = 1, \ldots, n$, if $w_i \in \mathbb{Z}_N \setminus \mathbb{Z}_N^*$ then N can be efficiently factored

3. if $w_1, \ldots, w_n \in \mathbb{Z}_N^*$, and for at least one value $j \in \{1, \ldots, n\}$, α_j has order > 4, the SE Test does not halt with probability at most $2^{-\lambda}$ because of the condition '$z^e \neq w \mod N$'

4. if $w_1, \ldots, w_n \in \mathbb{Z}_N^*$ and $\alpha_1, \ldots, \alpha_n$ have order ≤ 4, then either $\alpha_1 = \cdots = \alpha_n = 1$ or the Membership Test halts because of the condition '$t_i^2 = z_i w_i \mod N$';

5. if all of checks in the verification test are satisfied, then, except with probability $2^{-\lambda}$, it holds that $\alpha_i = 1$, and thus $w_i = (z_i)^e \mod N$, for all $i = 1, \ldots, n$.

Claim 1 directly follows by the Membership Test for that $w_i \in \mathbb{Z}_N$.

Claim 2 follows by definition of $\mathbb{Z}_N^*$; specifically, assume for sake of contradiction, that $w_i \notin \mathbb{Z}_N \backslash \mathbb{Z}_N^*$; this implies that $\gcd(w_i, N) > 1$. Since $w_i < N$, it holds that $\gcd(w_i, N) > 1$ is a factor of N.

Claim 3 follows by the soundness of the small-exponent batch verification test over $\mathbb{Z}_N^*$, as follows. First, note that the condition $z^e = w \mod N$ can be rewritten as $\Pi_{i=1}^n \alpha_i^{s_i} = 1 \mod N$, using the definition of α_i and the following two facts:

1. $z^e = (\Pi_{i=1}^n (z_i)^{s_i})^e = \Pi_{i=1}^n ((z_i)^e)^{s_i} \mod N$,

2. $w = \Pi_{i=1}^n (w_i)^{s_i} \mod N$.

Now, assume without loss of generality that the $j \in \{1, \ldots, n\}$ such that α_j has order > 4 is $j = 1$. Recall that by Lagrange's theorem in group theory, the order of any element in $\mathbb{Z}_N^*$ is a divisor of $\phi(N) = (p-1)(q-1) = 4p_1q_1$, for primes p_1, q_1. Thus, the order of α_1 is at least $\min(p_1, q_1)$. Now, assume that the condition $\Pi_{i=1}^n \alpha_i^{s_i} = 1 \mod N$ is satisfied for a particular tuple $(s_1, \ldots, s_n)$. This also means that

$$\alpha_1^{s_1} = \Pi_{i=2}^n \alpha_i^{-s_i} \mod N. \tag{7.1}$$

We observe that for any $s_2, \ldots, s_n$, since α_1 has order at least $\min(p_1, q_1)$, which is much larger than 2^λ, there is at most one $s_1 \in \{0,1\}^\lambda$ such that Equation 7.1 holds. Thus,

for fixed $s_2, \ldots, s_n$, the probability that this equation is true is at most $2^{-\lambda}$, since s_1 is uniformly chosen from $\{0,1\}^\lambda$, when s_1 is drawn at random. Hence, the probability that Equation 7.1 holds is at most $2^{-\lambda}$ even when all of $s_2, \ldots, s_n$ are chosen independently and uniformly from $\{0,1\}^\lambda$, from which the claim follows.

Claim 4 follows by combining Observation 7.1.1 and Observation 7.1.2 from Section 7.1. First, Observation 7.1.1 proves that when N has the special form we are considering, there are no integers of order 4 in $\mathbb{Z}_N^*$. Second, Observation 7.1.2 proves that if α_i has order 2, then $z_i w_i \mod N$ is not a quadratic residue mod N, which makes C halt because the check $(t_i^2 = z_i w_i \mod N)$ cannot be satisfied. Thus, we obtain that either $\alpha_1 = \cdots = \alpha_n = 1$ or C halts because of the condition $(t_i^2 = z_i w_i \mod N)$, from which the claim follows.

Claim 5 follows by directly combining claims 1-4. □

The next theorem describes the number of operation requires for each test:

Theorem 7.2.2. *Let p, q, p_1, q_1 be large, same-length, primes, such that $p = 2p_1 + 1$, $q = 2q_1 + 1$, and let $N = pq$. Let σ be the computational security parameter associated with the group $\mathbb{Z}_N^*$ and let λ be a statistical security parameter. Then the cost of the computation of function $F_{N,e}$ on n inputs with*

1. *RANDOM SUBSET TEST is $n\lambda + t_{exp,\lambda}(\sigma)$ multiplications in $\mathbb{Z}_N^*$*
2. *SMALL EXPONENT TEST is $2n + 2t_{prod,n,exp}(\lambda) + t_{exp}(\sigma)$ multiplications in $\mathbb{Z}_N^*$*

Proof. We prove it by showing each test:

- **Random Subset Test:** The Atomic Random Subset requires about $n + t_{exp}(\sigma)$ multiplications in $\mathbb{Z}_N^*$ by inspection of the Test. However, in order to lower the error $2^{-\lambda}$ the atomic protocol has to be repeated λ times. Thus the test performs $n\lambda + t_{exp,\lambda}(\sigma)$ multiplications in $\mathbb{Z}_N^*$. Note that to check whether $w_i \notin \mathbb{Z}_N^*$, the test only needs to check that these numbers are in $[1, p-1]$ (i.e. $w_i > 0$ and $w_i < p$), which only requires time linear in σ. Thus, the operation is a lower-order operation compared to multiplication in $\mathbb{Z}_N^*$, and we can ignore it in our calculations.

- **Small Exponent Test:** The test performs $2n$ exponentiations to λ-bit exponents, 1 exponentiation to a σ-bit exponent and $2n$ multiplications. Note that to check whether t_i or $w_i \notin \mathbb{Z}_N^*$, the test only needs to check that these numbers are in $[1, p-1]$, which we can ignore in our calculations. □

7.3 From Batch Verification to Batch Delegation

In this section, we propose two client-server protocols for secure delegated computation of batch exponentiations in group $\mathbb{Z}_N^*$. Both protocols use both generic constructions of batch verification tests which were described in the previous section. In the first protocol, the client does not hide the input from the server, and the client does not have to precalculate in the offline phase. In the second protocol, the client hides the input from the server, but then has to do precalculation in the offline phase. In both protocols, first we describe the formal description of the protocol (C, S) and then prove efficiency,

correctness, privacy (if it is applicable), and security requirements.

7.3.1 Protocol 1: Batch Delegation with No Input Privacy or Offline Phase

Protocol 1 satisfies the following:

Theorem 7.3.1. Let p, q, p_1, q_1 be large, same-length, primes, such that $p = 2p_1 + 1$, $q = 2q_1 + 1$, and let $N = pq$. Let σ be the computational security parameter associated with the group $\mathbb{Z}_N^*$, and let λ be a statistical security parameter. There exists (constructively) a client-server protocol (C, S) for batch delegation of computation of the function $F_{N,e}$ on n inputs, which satisfies

1. δ_c-correctness, for $\delta_c = 1$;

2. ϵ_s-security, unless S can factor N, for $\epsilon_s = 2^{-\lambda}$;

3. $(t_F, t_S, t_P, t_C, cc, mc)$-efficiency, where

   - t_F is $= t_{exp,n}(\sigma)$
   - t_S is $= t_{exp,n}(\sigma)$ for RST and t_S is $= t_{exp,2n}(\sigma)$ for SET
   - $cc = 2n$ elements in $\mathbb{Z}_N^*$ RST and $cc = 3n$ elements in $\mathbb{Z}_N^*$ for SET
   - $t_P = 0$ and $mc = 2$.
   - t_C depends on which verification test C uses:
     - RST: t_C is $n\lambda + t_{exp,\lambda}(\sigma)$ multiplications in $\mathbb{Z}_N^*$

– SET: t_C is $2n + 2t_{prod,n,exp}(\lambda) + t_{exp}(\sigma)$ multiplications in $\mathbb{Z}_N^*$

Formal description of protocol (C, S)**.**

Input to S and C: 1^σ, 1^λ , $desc(F_{N,e})$, $x_1, \ldots, x_n \in \mathbb{Z}_N^*, e \in \mathbb{Z}_{\phi(N)}$ such that $\gcd(e, \phi(N)) = 1$.

1. C sends $x_1, \ldots, x_n$ to S

2. For $i = 1, \ldots, n$

   S computes $y_i := x_i^e$, $t_i := x_i^{\frac{e+1}{2}} \mod N$

   S sends $y_1, \ldots, y_n, t_1, \ldots, t_n$ to C

3. C runs one of the following verification tests for batch instance $(x_1, y_1) \ldots, (x_n, y_n)$ and values $t_1, \ldots, t_n$.

   (a) RST (it does not need values $(t_1, \ldots, t_n)$)

   (b) SET

   if the chosen test from above is not satisfied then

   C **returns** $\perp$ and the protocol halts

   C **returns** $(y_1, \ldots, y_n)$

Properties of protocol (C, S)**:** *The efficiency properties* are verified by protocol inspection

- *Round complexity:* the protocol only requires one round, consisting of one message from C to S followed by one message from S to C.

- *Communication complexity:* the protocol requires to transfer n elements from C to S in $\mathbb{Z}_N^*$ and $2n$ elements from S to C in $\mathbb{Z}_N^*$ using SET and n elements from S to C in $\mathbb{Z}_N^*$ using RST (because RST does not need to have $t_1, \ldots, t_n$ elements).

- *Runtime complexity:* S performs n exponentiations to σ-bit exponents in $\mathbb{Z}_N^*$ in RST and n exponentiations to σ-bit exponents in $\mathbb{Z}_N^*$ in SET. C's performance also depends what test is used. The cost of the tests were described in Theorem 7.2.2.

The *correctness* property follows by showing that if C and S follow the protocol, C always outputs $y_i = x_i^e$ for all $i = 1, \ldots, n$. First, we show the Membership Checks performed by C are always passed. In the Membership Check in Random Subset Test, C checks if $y_i \in \mathbb{Z}_N$ by verifying that the y_i are ≥ 0 and $< N$. This is true because y_i is computed by S as $x_i^e \mod N$, for $i = 1, \ldots, N$ and $x_i \in \mathbb{Z}_N^*$. The Membership Check in Small Exponent Test and Test, C checks if $y_i, t_i \in \mathbb{Z}_N$ similarly as above explanations and the check $t_i^2 \neq x_i y_i \mod N$ is also passed since

$$t_i^2 = \left(x_i^{(e+1)/2}\right)^2 = x_i^{e+1} = x_i(x_i^{e_i}) = x_i y_i \mod N.$$

for y_i and t_i computed by S as $y_i := x_i^e$ and $t_i := x_i^{\frac{e+1}{2}} \mod N$. Lastly, y_i is the correct output since $y_i = x_i^e$ computed by S for all $i = 1, \ldots n$.

In order to show the *security* property against a malicious S, we need to compute an upper

bound ϵ_s on the probability that S convinces C to output a y_i such that $y_i \neq F_{N,e}(x_i)$ for some $i = 1, \ldots, n$. By Theorem 6.2.1, in the previous section, we showed that for

- ATOMIC RANDOM SUBSET TEST satisfies $\epsilon_s = 1/2$, i.e $(x_1, y_1), \ldots, (x_n, y_n)$ is incorrect batch instance and C does not return $\perp$. However for **Random Subset Test** C repeats ATOMIC RANDOM SUBSET TEST λ times. Thus $\epsilon_s = 2^{-\lambda}$.

- **Small Exponent Test** satisfies $\epsilon_s = 2^{-\lambda}$. □

7.3.2 Protocol 2: Batch Delegation with Input Privacy and Offline Phase

Protocol 2 satisfies the following:

Theorem 7.3.2. Let p, q, p_1, q_1 be large, same-length, primes, such that $p = 2p_1 + 1$, $q = 2q_1 + 1$, and let $N = pq$. Let σ be the computational security parameter associated with the group $\mathbb{Z}_N^*$, and let λ be a statistical security parameter. There exists (constructively) a client-server protocol (C, S) for batch delegation of the computation of function $F_{N,e}$ on n inputs, which satisfies

1. δ_c-correctness, for $\delta_c = 1$;

2. ϵ_s-security, unless S can factor N, for $\epsilon_s = 2^{-\lambda}$;

3. ϵ_p-privacy, for $\epsilon_p = 0$;

4. $(t_F, t_S, t_P, t_C, cc, mc)$-efficiency, where

- t_F is $= t_{exp,n}(\sigma)$
- t_S is $= t_{exp,n}(\sigma)$ for RST and t_S is $= t_{exp,2n}(\sigma)$ for SET
- $cc = 2n$ elements in $\mathbb{Z}_N^*$ RST and $cc = 3n$ elements in $\mathbb{Z}_N^*$ for SET
- $t_P = 0$ and $mc = 2$.
- t_C depends on which verification test C uses:
  - RST: t_C is $n(\lambda + 2) + t_{exp,\lambda}(\sigma)$ multiplications in $\mathbb{Z}_N^*$
  - SET: t_C is $4n + 2t_{prod,n,exp}(\lambda) + t_{exp}(\sigma)$ multiplications in $\mathbb{Z}_N^*$

Formal description of protocol (C, S)**.**

Input to S: 1^σ, 1^λ, $desc(F_{N,e})$, $e \in \mathbb{Z}_{\phi(N)}$ such that $\gcd(e, \phi(N)) = 1$.

Input to C: 1^σ, 1^λ , $desc(F_{N,e})$, $x_1, \ldots, x_n \in \mathbb{Z}_N^*, e \in \mathbb{Z}_{\phi(N)}$

Offline phase instructions:

1. For $i = 1, \ldots, n$

   C randomly chooses $u_i \in \mathbb{Z}_N^*$ and sets $v_i := u_i^{-e}$

Online phase instructions:

1. For $i = 1, \ldots, n$

   C sets $z_i := (x_i * u_i) \mod N$

   C sends $z_1, \ldots, z_n$ to S

2. For $i = 1, \ldots, n$

   S computes $w_i := z_i^e$, $t_i := z_i^{\frac{e+1}{2}} \mod N$

   S sends $w_1, \ldots, w_n, t_1, \ldots, t_n$ to C

3. C checks correctness of w_i for each $i = 1, \ldots, n$ for the given $t_1, \ldots, t_n$ and with batch instance $(z_1, w_1) \ldots, (z_n, w_n)$ to one of the following two tests:

   (a) Random Subset (RS) Test (it does not need $(t_1, \ldots, t_n)$)

   (b) Small Exponent (SE) Test

   if the chosen test is not satisfied then

   C **returns** $\perp$ and the protocol halts

   For $i = 1, \ldots, n$

   C sets $y_i := w_i \cdot v_i \mod N$

   C **returns** $(y_1, \ldots, y_n)$

Properties of protocol (C, S) follows from Section 6.3.2 and Section 7.3.1.

7.4 Performance Analysis and Conclusions

In Table 7.1 we listed all parameters for the two protocols described in Section 7.3 for secure batch delegation of exponentiations to a single (and possibly malicious) server over discrete log type groups for the function $F_{N,e}(x_i) = x_i^e$, $i = 1, \ldots, n$.

Table 7.1: Efficiency parameter of Protocol 1 and 2 for the function $F_{N,e}(x_i) = x_i^e$ for all $i = 1, \ldots, n$

EP	Protocol 1	Protocol 2
t_F	$t_{n,exp}(\sigma)$	$t_{n,exp}(\sigma)$
t_P	0	$t_{n,exp}(\sigma)$
t_C	**RS:** $n\lambda + t_{exp,\lambda}(\sigma)$ **SE:** $2n + 2 \cdot t_{prod,n,exp}(\lambda) + t_{exp}(\sigma)$	**RS:** $n(\lambda + 2) + t_{exp,\lambda}(\sigma)$ **SE:** $4n + 2 \cdot t_{prod,n,exp}(\lambda) + t_{exp}(\sigma)$
t_S	**RS:** $t_{n,exp}(\sigma)$ **SE:** $t_{2n,exp}(\sigma)$	**RS:** $t_{n,exp}(\sigma)$ **SE:** $t_{2n,exp}(\sigma)$
ϵ_p	NA	0
ϵ_s	$2^{-\lambda}$	$2^{-\lambda}$

In Table 7.2 and 7.3 show the concrete performance of our protocols 1 and 2 using the batch verification test Random Subset (RS) and Small Exponent (SE). In Table 7.2, we set $\sigma = 1024$ and $\lambda = 128$ (i.e. security probability will be $\epsilon_s = 2^{-128}$), in Table 7.3, we set $\sigma = 2048$ and $\lambda = 128$. We count the number of multiplications of the client in the online phase and compare with the naive batch test. The test is not naively implemented because we use an improved algorithm as described in Section 7.4.

- In Table 7.2, observe that the RS test is actually worse than naive for $n \leq 487$, the SE test is the best as we expected. The factor of improvement increases with n: at $n = 10$ we can do about 1.45 times better than naive (using SE); at $n = 487$, about 8 times better (using SE); at $n = 5000$, about 10 times better (using SE)and about 1.5 times better (using RS).

- In Table 7.3, observe that the RS test is actually worse than naive for $n \leq 306$, the SE test is the best as we expected. The factor of improvement increases with n:

Table 7.2: C's online multiplication for increasing value n with $\epsilon_s = 2^{-128}, \lambda = 128, \sigma =$ 1024-bits

n	**Naive**	**Protocol 1**		**Protocol 2**	
		RS	**SE**	**RS**	**SE**
5	1,536	20,068	1,333	21,097	1,343
10	2,048	20,388	1,398	21,422	1,418
50	6,144	22,948	1,860	24,022	1,960
100	11,264	26,148	2,392	27,272	2,592
200	21,504	32,548	3,402	33,772	3,802
487	50,892	50,916	6,142	52,427	7,116
1,000	103,424	83,748	10,798	85,772	12,798
5,000	513,024	339,748	44,436	345,772	54,436
10,000	1,025,024	659,748	84,346	670,772	104,346

at $n = 10$ we can do about 1.55 times better than naive (using SE); at $n = 500$, about 12 times better (using SE); at $n = 5000$, about 19 times better (using SE) and about 2.6 times better (using RS).

Table 7.3: C's online multiplication for increasing value n with $\epsilon_s = 2^{-128}, \lambda = 128$, $\sigma =$ 2048-bits

n	**Naive**	**Protocol 1**		**Protocol 2**	
		RS	**SE**	**RS**	**SE**
5	2,978	39,817	2,440	41,870	2,450
10	3,909	40,137	2,506	42,195	2,526
50	11,357	42,697	2,968	44,795	3,068
100	20,666	45,897	3,500	48,045	3,700
200	39,284	52,297	4,510	54,545	4,910
306	59,019	59,081	5,541	61,435	6,153
500	95,138	71,497	7,370	74,045	8,370
1,000	188,229	103,497	11,906	106,545	13,906
5,000	932,957	359,497	45,543	366,545	55,543
10,000	1,863,866	679,497	85,454	691,545	105,454

Chapter 8

Conclusions

We studied the problem of a client correctly, efficiently, privately and securely to delegating the computation of a single and multiple group exponentiations to a more computationally powerful server (i.e., a cloud server). This problem was originally left open in [17]. We consider the following functions during client server delegations:

1. Fixed-base variable-exponent functions, $F_{G,exp,g} : \mathbb{Z}_n \rightarrow G$ defined as $F_{G,exp,g}(x) = g^x$ where G is a multiplicative group and the fixed base is $g \in G$. More specifically, in this modular exponentiation, we concentrated on the discrete log based scheme for the cyclic group function, $F_{G,exp,g} : \mathbb{Z}_n \rightarrow G$ defined as $F_{G,exp,g}(x) = g^x$ where $g \in \mathbb{Z}_q$.

2. Variable-base fixed-exponent function, $F_{G,exp,k} : G \rightarrow G$ defined as $F_{G,exp,k}(x) = x^k$ where G is a multiplicative group and the fixed exponent is $k \in \mathbb{Z}^+$. More specifically in this modular exponentiation, we concentrated on RSA – type group function, $F_{G,exp,k}(x) = x^k \mod n$ where $n = p_1 * p_2$ and $p_i := 2p'_i + 1$ for p_1, p_2, p'_1, and p'_2 are

large primes and $i = 1, 2$.

We presented practical and provable solutions to single and multiple exponentiations delegation problems for groups commonly used in cryptography, based on both the discrete logarithm and RSA hardness assumptions. Our results directly solve various algorithms in cryptosystems of major importance, including RSA encryption and Diffie-Hellman key agreement protocols.

Appendix A

Delegation of Inverses

This section in appendix A shows a client-server protocol for delegated computation of group inverses, as described in paper [10]. The protocol described below works for any group (i.e. commutative and non-commutative groups), and any computationally unrestricted adversary and will be used as a subprotocol in the protocols given in Chapters 3 and 4 for delegated computation of modular exponentiation of the function $F_{G,exp,k}$.

Notation: Let $(G, *)$ be a group, where the group operation is *multiplication*, denoted as $*$. The identity element of a group G is 1. For any $a \in G$, let $b = a^{-1}$ denote the *inverse* of a (i.e. the value b is inverse if $a * b = 1$). And let $F_{G,inv} : G \to G$ where $F_{G,inv}(a) = a^{-1}$. The following theorem is about the existence of such a function:

Theorem A.0.1. *There exists (constructively) a client-server protocol (C, S) for delegated computation of function $F_{G,inv}$ which satisfies*

1. *δ_c-correctness, for $\delta_c = 1$;*

2. *ϵ_s-security, for $\epsilon_s = 0$;*

3. *ϵ_p-privacy, for $\epsilon_p = 0$;*

4. *efficiency with parameters (t_F, t_C, t_S, cc, mc) where*

   - *t_F and t_S are $= 1$ inversion in G;*
   - *t_C is $= 3$ multiplications in G;*
   - *$cc = 2$ elements in G and $mc = 2$.*

Remark 1. *The Theorem A.0.2 satisfies very strong versions of the security and privacy requirements because the server S can be the adversary and may run the protocol as much as desired (i.e. not restricted to polynomially many times), and for efficiency requirements t_C requires a running time of 3 multiplications in G.*

Informal description of protocol (C, S). Informally, the description of the protocol using Theorem A.0.2 for $F_{G,inv}$ shows that on input $x \in G$, C uses the group operation to mask x with the random element of G then sends the masked element to S . S then inverts the masked element and sends it back to C. Lastly, C checks the output by using the group operation on the received value and the masked value and derives the inverse for the input x. The following is the formal description:

Formal description of protocol (C, S).

Input to S: 1^σ , $desc(F_{G,inv})$

Input to C: 1^σ, $desc(F_{G,inv})$, $x \in G$

Protocol instructions:

1. C randomly chooses $c \in G$, computes $d = x * c$ and sends d to S;

2. S computes $e = d^{-1}$ and sends e to C;

3. C checks whether $d * e = 1$;

   if no, C returns failure symbol $\perp$;

   if yes, C computes $y = c * e$ and returns y.

Illustration of the protocol (C, S)**:**

Client		**Server**
$c \in_R G$	$\xrightarrow{d=x*c}$	
Checks: $d * e = 1$	$\xleftarrow{e=d^{-1}}$	
If $d * e \neq 1, C$ **returns** $\perp$		
If $d * e = 1$ C computes $y = c * e$		
and **returns** y		

Now we will proof the Theorem A.0.2:

Proof of Theorem A.0.2:

Properties of protocol (C, S):

1. The *efficiency properties:* C performs at most 3 multiplications in G ($t_C = 3_{mult}$) and S performs the inversion operation once ($t_S = 1_{inv}$). With respect to round complexity, the protocol only requires one message from C to S, followed by one

message from S to C ($m_c = 2$). With respect to communication complexity, the protocol only requires 2 the communication of elements in total ($cc = 1$).

2. The *correctness properties*: If C and S follow the protocol, C's check $d * e = 1$ is satisfied and C's output y satisfies

$$y = c * e = c * d^{-1} = c * (x * c)^{-1} = c * c^{-1} * x^{-1} = x^{-1}.$$

3. The *privacy property* follows by combining the following two observations:

   (a) on a single execution of (C, S), the message d sent by C does not leak any information about x;

   (b) seeing multiple executions of (C, S) does not help the adversary. C's input x is chosen by adversary.

   Both observations are consequences of the fact that in each execution of (C, S), the value d is uniformly distributed in G and is independent from all previous executions.

4. The *security property* follows by combining the following two observations:

   (a) on a single execution (C, S), C's verification in step 3 forces the adversary to send an honestly computed value, e, in step 2;

   (b) seeing multiple executions of (C, S) does not help the adversary, even C's inputs in these execution are chosen by adversary.

Observation (a) follows from the fact that there exists a single value e such that $e * d = 1$. Equivalently, $e = d^{-1}$ where e is the value that the honest S sends. Observation (b) follows by the privacy property.

□

Appendix B

Exponentiation Algorithms Without Delegation

In this appendix, we would like to use the textbook along with paper [6] to describe exponentiation algorithms without delegation.

B.1 Naive Way of Calculating Modular Exponentiation

Let G be a group and let $a \in G$. Define the function $F_{G,exp,g} : \mathbb{Z}_q \to G$ as $F_{G,exp,g}(n) = g^n$. The naive way of exponentiation is performing$n - 1$ multiplications (i.e. $g * g * ... * g$). Algorithm 4 describes naive exponentiation:

Algorithm 4 Group Exponentiation $NExp_F$

1: $y_0 = 1$
2: **for** $i = 1, \ldots, n$ **do**
3: $\quad y_i = y_{i-1} * g$
4: **end for**
5: **return** y_n

With this algorithm, $n-1$ group multiplications are required in order to evaluate $y = g^n$, but n can be as large as order of the group. Usually we look at groups containing about 2^{512} elements. Exponentiation by this method is not feasible. $2^{512}-1$ number of group multiplication takes very long time. In other words Algorithm 4 is an exponential time algorithm.

B.2 Square-and-Multiply Method for Modular Exponentiation

In this section we introduce the square and multiply method, which is much better than the method given in the previous section. First, we write an example then we write the algorithm.

Example: Suppose the binary length of n is 5 (i.e. $|n| = 5$), meaning the binary representation of n has the form b_4, b_3, b_2, b_1, b_0 where $b_i \in \{0, 1\}$. Then

$$\begin{aligned} n =& 2^4 b_4 + 2^3 b_3 + 2^2 b_2 + 2^1 b_1 + 2^0 b_0 \\ =& 16 b_4 + 8 b_3 + 4 b_2 + 2 b_1 + b_0 \end{aligned}$$

The square and multiply algorithm will proceed to compute the values $y_5, y_4, y_3, y_2, y_1, y_0$ in turn, as follows:

$$\begin{aligned} y_5 &= 1 \\ y_4 &= y_5^2 \cdot g^{b_4} = g^{b_4} \\ y_3 &= y_4^2 \cdot g^{b_3} = g^{2b_4 + b_3} \end{aligned}$$

$$y_2 = y_3^2 \cdot g^{b_2} = g^{4b_4+2b_3+b_2}$$

$$y_1 = y_2^2 \cdot g^{b_1} = g^{8b_4+4b_3+2b_2+b_1}$$

$$y_0 = y_1^2 \cdot g^{b_0} = g^{16b_4+8b_3+4b_2+2b_1+b_0}$$

Two group multiplications are required to compute y_i from y_{i+1} and the number of steps is the binary length of n (i.e. $\log_2 n$) □

In general, if $|n| = \sigma$ then the binary representation of n is $b_{\sigma-1}, b_{\sigma-2}, \ldots, b_0$ such that $n = 2^{\sigma-1}b_{\sigma-1} + 2^{\sigma-2}b_{\sigma-2} + \ldots + 2^0 b_0$. The square and multiply algorithm, Algorithm 5, proceeds as the example to evaluate the function $F_{G,exp,g} : \mathbb{Z}_q \to G$ as $F_{G,exp,g}(n) = g^n$.

Algorithm 5 Square Multiply Algorithm $SMExp_F$

1: Let $b_{\sigma-1}, b_{\sigma-2}, \ldots, b_0$ be the binary representation of n
2: $y_\sigma = 1$
3: **for** $i = \sigma - 1$ down to 0 **do**
4: $\quad y_i = y_{i+1}^2 * g^{b_i}$
5: **end for**
6: **return** y_0

Algorithm 5 uses two group multiplications per iteration of the loop:

1. multiply y by itself (i.e. y_{i+1}^2)

2. multiply the result by g^{b_i}

Note that computation of g^{b_i} is without cost because $b_i \in \{0, 1\}$. The total cost is $2\sigma = 2\log_2(n) = |n|$ group multiplications.

In the average case we can do only alittle better than Algorithm 5 in average case. Since $b_i = 0$ or $b_i = 1$ then $g^{b_i} = 1$ or $g^{b_i} = g$. Thus if $b_i = 0$ in the square and multiply algorithm we do not need to multiply by g. Specificly, consider Algorithm 6:

Algorithm 6 Better Square Multiply Algorithm $BSMExp_F$

Let $b_{\sigma-1}, b_{\sigma-2}, \ldots, b_0$ be the binary representation of n
$y_\sigma = 1$
for $i = \sigma - 1$ down to 0 **do**
    if $b_i = 1$ **then**
        $y_i = y_{i+1}^2 * g$
    else
        $y_i = y_{i+1}^2$
    end if
end for
return y_0

The Algorithm 6 runs in the average case 1.5σ. This is because in average, if $b_i = 1$, the number of group multiplications is $\sigma/2$, and we always square the y_i, meaning that the total number of group multiplications is σ. But, in the worst case, Algorithm 6 runs 2σ group multiplications, which is the same as Algorithm 5. This happens when $b_i = 1$ for all $i \in \{0, 1 \ldots, \sigma - 1\}$. In Chapters 3 and 4, we write the protocol for two parties, client and server, which improves on Algorithm 6 by delegating to two parties, in order to decrease the number of client multiplications as well as all other performance metrics.

Bibliography

[1] G. Di Crescenzo, M.Khodjaeva, D. Kahrobaei, V. Shpilrain, *Practical and Secure Delegation of Exponentiations over Discrete-Log Groups to a Single Malicious Server*, submitted at the 9th ACM Cloud Computing Security Workshop (CCSW 2017)

[2] G. Di Crescenzo, M.Khodjaeva, D. Kahrobaei, V. Shpilrain, *Computing Multiple Exponentiations in Discrete Log and RSA Groups: From Batch Verification to Batch Delegation*,accepted at the IEEE Workshop on Security and Privacy in the Cloud (SPC 2017)

[3] A. Arbit and Y. Livne and Y. Oren and A. Wool, *Implementing public-key cryptography on passive RFID tags is practical.* In: Int. J. Inf. Sec. 14(1): 85-99 (2015)

[4] P. Barrett, *Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor*, in: Advances in Cryptology, CRYPTO 1986, LNCS **263** (1986), 311-323.

[5] L. Batina and J. Guajardo and T. Kerins and N. Mentens and P. Tuyls and I. Verbauwhede, *Public-Key Cryptography for RFID-Tags.* In: Fifth Annual IEEE International Conference on Pervasive Computing and Communications - Workshops (PerCom Workshops 2007), March 2007, White Plains, New York, USA, pp. 217-222, 2007.

[6] Bellare, Mihir and Garay, Juan A and Rabin, Tal, *Fast batch verification for modular exponentiation and digital signatures.* In: International Conference on the Theory and Applications of Cryptographic Techniques: 236-250, Springer (1998)

[7] V. Boyko and M. Peinado and R. Venkatesan, *Speeding up discrete log and factoring based schemes via precomputations.* In: Advances in Cryptology, EUROCRYPT'98, pp. 221-235, Springer, 1998.

[8] Brickell, Ernest F and Gordon, Daniel M and McCurley, Kevin S and Wilson, David B, *Fast exponentiation with precomputation: algorithms and lower bounds*, In: Preprint, Mar (1995)

[9] Cai, Jianxing and Ren, Yanli and Huang, Chunshui, *Verifiable Outsourcing Computation of Modular Exponentiations with Single Serve*

[10] Cavallo, Bren and Di Crescenzo, Giovanni and Kahrobaei, Delaram and Shpilrain, Vladimir, *Efficient and secure delegation of group exponentiation to a single server*, In: International Workshop on Radio Frequency Identification: Security and Privacy Issues: 156-173, Springer (2015)

[11] Chaum, David and Evertse, Jan-Hendrik and van de Graaf, Jeroen and Peralta, Renér, *Demonstrating possession of a discrete logarithm without revealing it*, In: Conference on the Theory and Application of Cryptographic Techniques: 200-212, Springer (1986)

[12] X. Chen and J. Li and J. Ma and Q. Tang and W. Lou, *New algorithms for secure outsourcing of modular exponentiations.* In: Computer Security–ESORICS 2012, pp. 541-556, 2012.

[13] M. Dijk, D. Clarke, B. Gassend, G. Suh, and S. Devadas, *Speeding Up Exponentiation using an Untrusted Computational Resource.* In: Designs, Codes and Cryptography, 39 (2), pp. 253-273, 2006.

[14] Chevalier, Céline and Laguillaumie, Fabien and Vergnaud, Damien, *Privately outsourcing exponentiation to a single server: cryptanalysis and optimal constructions*, In: European Symposium on Research in Computer Security: 261-278, Springer (2016)

[15] R. Gennaro, C. Gentry, and B. Parno, *Non-interactive verifiable computing: Outsourcing computation to untrusted workers,* in: Advances in Cryptology, CRYPTO 2010, Lecture Notes Comp. Sc. **6223** (2010), 465-482.

[16] Goldreich, Oded and Micali, Silvio and Wigderson, Avi, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, In: Journal of the ACM (JACM), 38 (3): 690-728, ACM (1991)

[17] S. Hohenberger and A. Lysyanskaya, *How to securely outsource cryptographic computations.* In: Theory of Cryptography, pp. 264-282, 2005.

[18] M. Jakobsson and S. Wetzel, *Secure server-aided signature generation.* In: Public Key Cryptography, pp. 383-401, Springer, 2001.

[19] Lim, Chae Hoon and Lee, Pil Joong *More flexible exponentiation with precomputation*, In: Annual International Cryptology Conference: 95-107, Springer (1994)

[20] X. Ma and J. Li and F. Zhang, *Outsourcing computation of modular exponentiations in cloud computing.* In: Cluster Computing (2013) 16:787-796 (also INCoS 2012).

[21] P. Q. Nguyen and I. E. Shparlinski and J. Stern, *Distribution of modular sums and the security of the server aided exponentiation.* In: Cryptography and Computational Number Theory, pp. 331-342, Springer, 2001.

[22] Y. Wang and Q. Wu and D. Wong and B. Qin and S. Chow and Z. Liu and X. Tao, *Securely outsourcing exponentiations with single untrusted program for cloud storage.* In: Computer Security-ESORICS 2014, pp.326-343, Springer, 2014.

[23] A. C. Yao, *Protocols for secure computations.* In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, pp. 160-168, IEEE Computer Society, 1982.

[24] A. C. Yao, *On the evaluation of powers.* In: SIAM Journal on Computing 5(1976), pp. 100-108, MR 16128.

[25] Gennaro, Rosario and Krawczyk, Hugo and Rabin, Tal, *RSA-based undeniable signatures.* In: Advances in CryptologyCRYPTO'97, pp. 132-149, Springer, 1997.

[26] Boyd, Colin and Pavlovski, Chris, *Attacking and repairing batch verification schemes.* In: ASIACRYPT, (1976) pp. 58-71, Springer, 2000.

[27] Fiat, Amos, *Batch RSA,* in: Conference on the Theory and Application of Cryptology, pp.175-185, Springer, (1989)

[28] Lim, Chae Hoon and Lee, Pil Joong, *Security of interactive DSA batch verification*, In: Electronics letters, 30 (19): 1592-1593, IET (1994)

[29] Naccache, David, *Secure and practical identity-based encryption*, In: IET Information Security, 1 (2): 59-64, IET (2007)

[30] Camenisch, Jan and Hohenberger, Susan and Pedersen, Michael Ostergaard, *Batch verification of short signatures*, In: Eurocrypt, 4515: 246-263, Springer (2007)

[31] Van De Graaf, Jeroen and Peralta, Renė, *A simple and secure way to show the validity of your public key*, In: Conference on the Theory and Application of Cryptographic Techniques, pp.128-134, Springer (1987)

[32] Niven, Ivan and Zuckerman, Herbert S and Montgomery, Hugh L, *An introduction to the theory of numbers*, In:John Wiley & Sons (2008)

[33] Cubaleska, Biljana and Rieke, Andreas and Hermann, Thomas, *Improving and extending the Lim/Lee exponentiation algorithm*, In: International Workshop on Selected Areas in Cryptography, pp.163-174, Springer (1999)

[34] Bernstein, Daniel J, *PIPPENGER'S EXPONENTIATION ALGORITHM*, In: Citeseer (2002)

[35] Möller, Bodo, *Improved techniques for fast exponentiation*, In: ICISC, (2587): pp.298-312, Springer (2002)

[36] Yen, S-M and Laih, C-S, *Improved digital signature suitable for batch verification*, In: IEEE Transactions on Computers, 44 (7) pp.957-959, IEEE (1995)

[37] Naccache, David and M'RaÏhi, David and Vaudenay, Serge and Raphaeli, Dan, *Can DSA be improved? Complexity trade-offs with the digital signature standard*, In: Workshop on the Theory and Application of of Cryptographic Techniques pp.77-85, Springer (1994)