論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Constructions for Multi-Party Computation Based on Secret Sharing and Homomorphic Encryption
著者(和文)	盧儀
Author(English)	Yi Lu
出典(和文)	学位:博士(理学), 学位授与機関:東京工業大学, 報告番号:甲第12833号, 授与年月日:2024年9月20日, 学位の種別:課程博士, 審査員:田中 圭介,伊東 利哉,尾形 わかは,鹿島 亮,安永 憲司
Citation(English)	Degree:Doctor (Science), Conferring organization: Tokyo Institute of Technology, Report number:甲第12833号, Conferred date:2024/9/20, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Constructions for Multi-Party Computation Based on Secret Sharing and Homomorphic Encryption

Yi Lu
Supervisor: Keisuke Tanaka

Department of Mathematical and Computing Science
School of Computing
Tokyo Institute of Technology

Acknowledgement

First of all, I am deeply grateful to my supervisor Professor Keisuke Tanaka. He gave me a lot of advices and encouragements through the daily activities in his laboratory. Throughout these five years, his advice was great not only for my research but also for my life. Also, he gave me a lot of opportunities to experience important things in my life.

I would like to thank the referees of my doctoral thesis, Prof. Toshiya Itoh, Prof. Wakaha Ogata, Prof. Ryo Kashima, and Prof. Kenji Yasunaga for taking their time to reading this thesis and to listening to my presentations. Their feedbacks helped me to improve my work.

I also would like to thank current members and ex-members of Tanaka laboratory. I spent delightful time with them in daily life. Their kind and great help was crucial for completing my research.

I am also grateful to Goichiro Hanaoka who is a prime senior researcher at Cyber Physical Security Research Center (CPSEC) of National Institute of Advanced Industrial Science and Technology (AIST) and the members of "Shin-Akarui-Angou-Benkyou-Kai". They gives me grateful comments and suggestions for my research. I have a lot of experience in the discussion with them. Expressly, I would deeply like to thank Kazuma Ohara, Jacob Schuldt, and Keisuke Hara. Their comments and advice are indispensable for my thesis. Without their guidance and persistent help, this thesis would not be done.

Last but not least, I would also like to thank my father Lu Jinhai and my mother Li Chengqin for their grateful support and encouragements. Without their warm help, this thesis would not have been possible and I would not decide to go to the doctoral course after completing the master's course.

Contents

Chapter 1

Introduction

1.1 General Background on MPC

The advancement of computer science has led to a proliferation of data, fundamentally altering various aspects of our lives. For instance, individuals are now interconnected with vast quantities of personal information, encompassing details such as age, citizenship, and income. This flood of data has changed how businesses operate and make choices. Some companies even leverage this data abundance to offer personalized recommendations and optimize profits based on users' digital footprints. Even though there is a huge amount of data available online for these companies and organizations to use, that doesn't mean that we do not need to think about the privacy and security of data. Actually, protecting personal information is more challenging than ever before.

To solve such privacy issues in the real world, *multi-party computation* (MPC) protocols have gathered attentions recently. Roughly, MPC is a novel cryptographic protocol to allow participating parties to jointly compute a function over their inputs while keeping them private. In the real world, MPC has been used in many scenarios to achieve both utility and privacy, for example, privacy-preserving data mining, privacy-preserving machine learning, secure e-auctions, and private set intersection. Especially, machine learning is increasingly becoming one of the dominant research fields, with many real-world applications such as self-driving cars, healthcare, and medicine. To build a model with good utility and accuracy, machine learning needs data from various sources. In this process, MPC can provide privacy for different organizations to share datasets with others without worrying about data being revealed.

Since the idea of MPC was first introduced by Andrew Yao in the early 1980s [Yao82], there have been 30 years of research on MPC, progressing from pure theoretical research to real-world

applications. Yao's first talk introduced a toy application called "Millionaires' Problem". The problem discusses two millionaires, Alice and Bob, who are interested in knowing who is richer without revealing their actual wealth. Yao [Yao82] proposed a garbled circuit-based scheme for secure two-party computation. Here, circuit means all the computation can be constructed from AND and XOR gate. Garbled circuits are a cryptographic technique for parties compute some functions by evaluating the binary circuit of the function, without revealing the secret binary input strings. Since the Yao's protocol, there are many constructions of MPC have been proposed. In recent MPC research area, the main methods for constructing MPC are garbled circuit, secret sharing, and homomorphic encryption.

Secret Sharing Based MPC. A secret-sharing scheme is a method of distributing a secret among a group of participants, each of whom is assigned a share of the secret. Informally, any secret sharing protocol for a set of n parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ consists of two phases,a sharing phase by an algorithm **Share** and a reconstruction phase realized by **Reconstruct**. The goal of the sharing phase is to create shares based on secret input and distribute these shares to other parties without revealing anything other than the share itself. In the reconstruction phase, the secret input is reconstructed from the shares. The first two-party secret sharing based MPC protocol is introduced by [BGW88] using oblivious transfer. Then this protocol were proved can be used in more than two parties case. There are many interesting research using ideas from linear algebra to solve the problem of secret sharing. The linear secret sharing schemes include Shamir's secret sharing scheme, additive secret sharing scheme, threshold secret sharing scheme, etc. In this thesis, we will pay attention to MPC based on additive secret sharing scheme.

Homomorphic Encryption Based MPC. An alternative approach to construct MPC is based on homomorphic encryption (HE). An HE scheme is a novel and powerful cryptographic primitive having various applications. This primitive allows anyone to perform computations over encrypted data with the data kept secret from it. Initially, Rivest et al. [RAD78] formulated the idea of HE, then the first HE implementation is based on the breakthrough work of Gentry [Gen09]. HE has several applications, for example in cloud computation scenarios, a cloud provider could use an HE scheme to operate on encrypted data. This approach guarantees a cloud customer's data privacy in the presence of threats form malicious employees or intruders. To date, various kinds of homomorphic encryption schemes has been proposed depending on different types of operations. Some schemes, like RSA [RSA78] and ElGamal [ElG84], enable operations such as multiplication on encrypted data, making them

multiplicative encryption schemes. Another type is an additive encryption scheme, exemplified by Paillier's cryptosystem [Pai99], which allows for addition on encrypted data and multiplication by a public constant. Lastly, fully homomorphic encryption schemes support both additions and multiplications, enabling the evaluation of any circuit over encrypted data. The first such scheme was developed by Gentry in 2009 [Gen09].

In this thesis, we focus on two research areas on constructing specific MPC: One is *secret sharing based Two-party Exponentiation MPC* and the other is *homomorphic encryption based multi-client verifiable computation*.

1.2 Background on Two-Party Exponentiation MPC

As our first result, we deal with how to implement an exponentiation functionality in MPC based on secret sharing. Exponentiation is one of the most frequently used function in machine learning, and is also useful in such as protecting secret keys in distributed systems such as Blockchain. In the latter context, MPC is used to generate discrete logarithm-based digital signatures while distributing the secret key, which is referred to as a threshold signatures or distributed signatures [GGN16, Lin17, WWW+14].

Exponentiation MPC protocols for different settings have been proposed so far. The first one is *public base*: the base is public and the exponent is secret; the second one is *public exponent*: the base is private and the exponent is public; and last one is *private exponentiation*: both the base and the exponent are privately held. In this thesis, we focus on the public base variant. For example, when computing the function used in the deep learning setting mentioned above, it is frequently required that the value e^x can be computed, where e is Napier's constant. Thus, when considering real-world applications, the public base variant is a useful and needed variant of exponentiation protocols.

In the following, in order to clarify our goal, we explain the three properties of exponentiation MPC protocols: (based on) additive secret sharing / Shamir's secret sharing, honest-majority / dishonest-majority, and with / without bit-decomposition. Firstly, we note that our goal is to construct an efficient public base exponentiation protocol *without using bit-decomposition* based on *additive secret sharing* in the *dishonest-majority* setting.

Additive Secret Sharing vs. Shamir's Secret Sharing. When constructing MPC protocols based on secret sharing, we mainly have two types of secret sharing: additive secret sharing and Shamir's secret sharing.

Additive secret sharing [ISO] is defined over a finite additive group $(+, \mathbb{G})$. In additive secret sharing for n parties, a secret $x \in \mathbb{G}$ will be randomly divided into $[x]^1, \ldots, [x]^n$ such that $[x]^1 + \cdots + [x]^n = x$, where n is the number of parties. The defined group for additive secret sharing determines the element form of shares and type of circuit on which parties want to perform.

Compared with Shamir secret sharing, one of the advantages of MPC based on additive secret sharing is the compatibility with "dishonest-majority" MPC frameworks. In other words, MPC based on additive secret sharing can be easily integrated with existing dishonest-majority MPC frameworks [DPSZ12, DKL+13, KPR18, ALSZ15, KOS16], and thus can utilize the ecosystems of these frameworks such as other efficient MPC protocols or cheater detection functionality. This is explained in more detail below.

Dishonest-Majority vs. Honest-Majority. Honest/Dishonest-majority is a criteria of MPC security regarding the number of corrupted parties among all participants. We call a MPC protocol secure against honest(/dishonest)-majority if the number of corrupted parties are less than(/equal or more) than half of the total number of parties, respectively. What is important is that security against dishonest-majority is much harder to achieve than honest-majority. In the fully information-theoretic setting, there exists an impossibility result that we cannot construct MPC for arbitrary functions in dishonest-majority setting [BGW88]. To achieve security against a dishonest-majority, the previous studies often introduce some computational assumptions, or the "online/offline" paradigm described later.

A Bit-Decomposition-Based Approach. A common approach for realizing some MPC functionalities is to firstly compute a binary representation of the input in secret shared form, and then construct a MPC protocol for the evaluation of the functionality in question via a Boolean circuit or a "mixed" Boolean and arithmetic circuit. The first step is known as *bit-decomposition*, and the usefulness of this approach was illustrated by Damgård et al. in [DFK+06] who proposed MPC protocols for equality testing, comparison, and exponentiation. Concretely, in a bit-decomposition protocol, a secret shared input $[x]_p$ is converted to a bit-wise sharing $[x_0]_p, \cdots, [x_{\ell-1}]_p$, such that $x = \sum_{i=0}^{\ell-1} x_i 2^i$, where the input $x \in \mathbb{Z}_p$ for some prime p.

Making use of bit-decomposition in the public base setting allows the adaptation of the well-known "square-and-multiply" algorithm (also referred to as "exponentiation by squaring" or "binary exponentiation") to the MPC setting. More specifically, considering a public base a and a secret shared exponent $[x]_p$, using bit-decomposition the secret shared bit representation

$[x]_B = [x_0]_p \dots [x_{\ell-1}]_p$, where $\sum_{i=0}^{\ell-1} 2^i x_i, x_i \in \{0,1\}$ can be obtained. Then, using the square-and-multiply algorithm, the shares $[a^x]_p$ can be obtained from this equation:

$$a^x = a^{\sum_{i=0}^{i-1} 2^i x_i} = \prod_{i=0}^{\ell-1} a^{2^i x_i} = \prod_{i=0}^{\ell-1} (x_i a^{2^i} + 1 - x_i)$$

However, this protocol requires $\mathcal{O}(1)$ rounds and $\mathcal{O}(\ell \log \ell)$ invocations of the underlying multiplication MPC protocol[1] due to the cost of bit-decomposition. In particular, the cost of $\mathcal{O}(\ell \log \ell)$ invocations of multiplication cannot be dismissed.

Due to this limitation, previous works [NX11, AAN18] focused on how to construct public base exponentiation protocol without the bit-decomposition technique. Especially, Aly, Abidin, and Nikova [AAN18] proposed a highly efficient public base exponentiation protocol and it requires only 3 rounds and 6 invocations of multiplication. While their protocol is efficient, it depends on Shamir's secret sharing scheme. [2]

Online/Offline Paradigm and Additive Sharing. The online/offline paradigm using preprocessed random shares called "Beaver triple" or "multiplication triple" [Bea92] is a well-known and easy way to introduce multiplication in the dishonest-majority setting. These dishonest-majority protocols consist of a preprocessing phase for generating Beaver triples (called the "offline phase") and a MPC protocol for the function to be computed which consumes Beaver triples (called the "online phase"). To the best of our knowledge, all known efficient offline protocols generating Beaver triples are designed for additive secret sharing, such as protocols using homomorphic encryption [DPSZ12, DKL+13, KPR18], or oblivious transfer [ALSZ15, KOS16]. Therefore, MPC based on additive secret sharing is useful in that it is easily integrated with these protocols.

On the other hand, the BGW protocol [BGW88], which is a well-known MPC protocol based on the Shamir's secret sharing scheme, is limited in its scope to the honest majority setting (that is, the number of corrupted parties is bounded by $n/2$).

[1] Since the multiplication MPC protocol is dominant in the communication, the communication complexity of MPC is usually measured by the number of invocation of multiplication.

[2] To construct an exponentiation protocol over additive secret sharing, we could consider utilizing share conversion between Shamir and Additive secret sharing. However, since [AAN18] assumes the base and the exponent are shared by different moduli, an additional modulus conversion is needed, which makes this approach more expensive.

1.3 Background on Multi-Client Verifiable Computation

One of the most practically important applications of an HE scheme is to construct one type of MPC protocols called *private delegating computations* (over clouds). Consider a situation where there are a client who has only a relatively weak computational device and a server who has more powerful computational resources. In this situation, we assume that a client wants to securely outsource his data to the (possibly malicious) remote server (for getting some computation result over his data), while allowing it to reliably perform computations over the data. Utilizing an HE scheme, we can easily realize this demand (as we call private delegating computations). Concretely, a client generates a ciphertext of his secret data and sends it to a server. Then, the server get a ciphertext of the computation result by performing (possibly heavy) homomorphic evaluations on the client's ciphertext and return it to the client. Finally, the client decrypts the evaluated ciphertext and get the computation result.[3] Obviously, as an important requirement on efficiency, the computation cost for a client should be independent from the size of a delegated function.

While this cryptographic protocol is already attractive, it has a significant concern regarding the integrity of computation results by a server. More precisely, there may be an incentive for a server trying to cheat and return an incorrect computation result to the client. This incentive might be occurred due to the nature of the performed computation (such as, the case that the server wants to convince the client of a particular result when it will give the server some benefits) or just the server's laziness for heavy computations. This problem motivates us to consider a more desirable protocol called private and *verifiable* delegating computation enabling a client to verify the given computation result is correct as well as get the result.[4] As a natural core cryptographic primitive for obtaining private and verifiable delegating computation, Lai, Deng, Pang, and Weng [LDPW14] proposed a *verifiable homomorphic encryption* (VHE) scheme. As the name suggests, a VHE scheme allows a user to not only decrypt the computation result but also check it is the correct evaluated value. Obviously, if given an VHE scheme, we can obtain private and verifiable delegating computation immediately.

At first glance, one might think that the above private and verifiable delegating computation is sufficiently enough for practical applications. However, when taking into account of the

[3]Note that a more advanced situation where a client wants computations requiring multiple inputs (by this single user) also can be solved by using an HE scheme.

[4]This protocol is firstly treated formally by Gennaro, Gentry, and Parno [GGP10] and it is sometimes called just verifiable computation.

real-world setting, we can see that it is not true. Specifically, it is important for us to consider private and verifiable delegating computations in the *multi-client setting*, which enables secure delegating computations on data by multiple users who have their own respective secret keys.Actually, when considering some practically important applications (such as, big data analysis for medical cares), it is inherently required to support computations on data given multiple users and it is not reasonable to assume that all of the users share only one key (used both for encryption and decryption procedures). Unfortunately, almost of the previous private and verifiable delegating computations support only computations on data given by a single user. Especially, the existing Lai et al.'s VHE scheme [LDPW14] only supports homomorphic evaluation on the ciphertexts generated using the same secret key.

1.4 Our First Contribution: Efficient Two-Party Exponentiation from Quotient Transfer

We propose a new public base exponentiation protocol without bit-decomposition based on additive secret sharing with the following three contributions.

New Framework for Exponentiation Protocol. At first, we propose a new framework for exponentiation constructed via a *quotient transfer* (QT) functionality, which is formalized in this thesis.[5] In this framework, we realize a constant round public base exponentiation protocol based on an additive secret sharing scheme.

Efficient Exponentiation Protocol Based on Constrained QT Protocol. For obtaining an efficient exponentiation protocol in our framework, we propose a limited class of a QT protocol (which is called *constrained* QT protocol) without relying on bit-decomposition. Here, *constrained* means that our QT protocol only works for even integers as input. Since we bypass the use of a bit-decomposition protocol, we succeed in reducing the complexity of our QT protocol. Combining our framework and constrained QT protocol, we obtain an efficient public base exponentiation protocol based on additive secret sharing.

Note that our exponentiation protocol has a limitation that inputs are less than half of the underlying modulus. That is, compared to the existing exponentiation protocols, an additional condition $2x < p$ is required for our protocol, where x

[5]QT was implicitly defined by [KIM+18]. In addition, in [OWIO19], a part of their protocol can be seen as a QT protocol based on bit-decomposition, even though they did not mention it as a QT protocol directly.

is the input and p is the modulus of the underlying group. We believe that this limitation is not significant for many practical applications.

Modulus Conversion Protocol Based on Constrained QT Protocol. In order to utilize our constrained QT protocol effectively, the secret shared exponent must be multiplied by two to ensure it is even, which in turn leads to the requirement that the public base is a quadratic residue in the group over which the exponent is shared. To address this limitation, we also propose a new modulus conversion protocol that enables the efficient conversion of additive shares over a prime field to additive shares over a different prime field. Using this we can ensure that the public base is always a quadratic residue via an appropriate conversion before running our exponentiation protocol. The modulus conversion protocol is likewise based on our constrained QT protocol, and to the best of our knowledge, outperforms existing protocols. This might be of independent interest.

As the most important advantage, our resulting exponentiation protocol requires only 3 rounds and 4 invocations of multiplication even in the case that we need our modulus conversion protocol as subroutine. Moreover, if modulus conversion is not required, our exponentiation protocol only requires 2 rounds and 3 invocations of multiplication We furthermore note that our modulus conversion protocol requires only 1 round and 1 invocation of multiplication. In the following, we will outline the main ideas behind our constructions.

Local Exponentiation. The main idea behind our approach is to make the computing parties do most of the computation locally. In particular, the exponentiation itself is done locally based on the shares of the exponent x. Let us for a moment assume that the two parties hold shares $[x]_p^1, [x]_p^2 \in \mathbb{Z}_p$, respectively, and that $[x]_p^1 + [x]_p^2 = x$ *over the integers* i.e. no reduction modulo p is required to recover x. In this case, the parties can directly compute $y_i = a^{[x]_p^i} \bmod p$, which will satisfy $o = y_1 \cdot y_2 \bmod p = a^{[x]_p^1 + [x]_p^2} \bmod p = a^x \bmod p$. Shares of o can be obtained by letting each party compute a sharing of y_i, send one share to the other party[6] , and let both parties interact in a standard multiplication protocol to compute $[o]_i$.

However, a standard secure sharing of x requires a potential reduction modulo p when adding the shares i.e. $[x]_p^1 + [x]_p^2 = x \bmod p$ which implies $[x]_p^1 + [x]_p^2 = x + t \cdot p$ for $t \in \{0, 1\}$ over the integers. In other words, the above value o will in this case be of the form $o = a^x \cdot a^{tp}$. Here, $a^{tp} = a^t \bmod p$ due to Fermat's little theorem, and we observe that the term a^t can be

[6]In our actual exponentiation protocol given in Algorithm 1, each party *locally* sets the shares $[y_i]_p^i = y_i$ and $[y_i]_p^{1-i} = 0$ (and does not send their shares to each other) in order to optimize the round complexity.

eliminated from o assuming the parties can compute (shares of) t by multiplying o with the multiplicative inverse a^{-1} conditioned on the value of t.

Efficient Quotient Transfer. The above approach assumes that the parties can efficiently compute the value t. This was implicitly defined by Kikuchi et al. [KIM+18] as a QT protocol. Note, however, that the efficiency of this protocol is crucial in the above approach to exponentiation, and basing the QT protocol on e.g. bit-decomposition as done by [OWIO19], will defy the purpose of this approach.

We propose a simple but efficient approach to a QT protocol constrained to even inputs. Specifically, we observe that if the input x is even, the value of t can be determined by the least significant bits of the shares $[x]_p^1$ and $[x]_p^2$, as $t = 1$ implies that $[x]_p^1 + [x]_p^2$ over the integers must be odd as the prime p is likewise odd, whereas $t = 0$ implies that $[x]_p^1 + [x]_p^2$ over the integers must be even (as the input is likewise even). Hence, the QT protocol can in this case be implemented via the appropriate comparison of the least significant bits of the shares $[x]_p^1$ and $[x]_p^2$. To make use of this constrained QT protocol in the computation of an exponentiation, we simply multiply input x by 2, and compute $\sqrt{a}^{2x}$.

Modulus Conversion. The above assumes that an appropriate value $\sqrt{a}$ can be computed i.e. that a is a quadratic residue modulo the prime p. Note that since p is prime, half of all elements in $\mathbb{Z}_p$ are quadratic residues, and if this is the case for a, the above approach works. However, if a is a quadratic non-residue, a different approach is required. We address this case by simply converting the shares of x in $\mathbb{Z}_p$ to shares in $\mathbb{Z}_{p'}$ for a prime p' for which a *is* a quadratic residue.

Similar to the above, for this to work, an efficient modulus conversion protocol is required. We obtain this by observing that our QT protocol allows the shares of q to be drawn from $\mathbb{Z}_{p'}$ as opposed to $\mathbb{Z}_p$ which the shares of the input x belongs to, which in turn, allows us to construct a very efficient modulus conversion protocol (the details are given in Section 2.2.2). Note that the restriction that the input x is even can easily be overcome by firstly multiplying x by 2, doing the conversion, and then multiply the result by 2^{-1}, which are both local operations. In comparison to the efficient conversion protocol by [KIM+18], which is based on bit-wise processing of the input, our protocol is simpler and more efficient. Concretely, while the modulus conversion protocol [KIM+18] requires $\mathcal{O}(\log p')$ rounds and $\mathcal{O}(\log p')$ invocations of multiplication, our protocol requires only 1 round and 1 invocation of multiplication.

Protocol	Tool[§]	DM frame. comp.[#]	BD[⋆]	Rounds	Multiplication[†]	
[DFK+06]	Linear	Yes	Yes	119	$\mathcal{O}(\ell \log \ell)$	50176
[NX11][*]	Linear	Yes	No	20	$\mathcal{O}(\ell)$	10508
[AAN18]	Shamir's	No	No	3	$\mathcal{O}(1)$	6
This work (with conversion)[$]	Additive	Yes	No	3	$\mathcal{O}(1)$	4
This work (w/o conversion)[$]	Additive	Yes	No	2	$\mathcal{O}(1)$	3

Table 1.1: Comparison between two-party (public base) exponentiation protocols.

§ In this column, "Linear" stands for (general) linear secret sharing, "Shamir's" stands for Shamir's secret sharing, and "Additive" stands for additive secret sharing. We note that a secret sharing scheme is called linear if each share has linearity and linear secret sharing includes Shamir's secret sharing and additive secret sharing.

In this column, we note whether each protocol is compatible with dishonest-majority (DM) frameworks.

⋆ In this column, we point out whether each protocol requires bit-decomposition (BD) or not.

* The proposed protocol is a private exponent type protocol, not a public base type protocol. As the former implies the later, in our comparison, we use their private exponent type protocol as a public base type.

† We consider the case $\ell = 64$ when estimating the number of multiplications.

$ Here, we consider two cases: whether we need modulus conversion or not. As mentioned above, in our protocol, if the public base does not have quadratic residue, we require an additional modulus conversion. In this case, when our modulus conversion is used, we need additional 1 round and 1 invocation of multiplication.

1.4.1 Related Works

Here, we compare the efficiency of the existing exponentiation protocols and summarize the comparison in Table 1.1. Up until now, as mentioned above, there have been a few works on exponentiation protocols not relying on a bit-decomposition protocol.

In 2011, Ning and Xu [NX11] introduced private exponentiation type protocols without a bit-decomposition technique. As a result, they obtain a protocol with 20 rounds and $164 \cdot \ell + 12$ invocations of multiplication for a public base, where ℓ is the number of message bits. Especially, when we consider $\ell = 64$, the number of invocations of multiplication is 10508.

Recently, Aly, Abidin, and Nikova [AAN18] simplified the Ning et al.'s protocol, and reduced the communication complexity and the number of invocations of multiplication based on the Shamir's secret sharing scheme. They also constructed a new public exponent exponentiation protocol. Regarding the public base exponentiation protocol, the number of rounds is 3 and the number of multiplication invocations is $3(1 + \lfloor \log(n) \rfloor)$, where n means the number of parties. Especially, in the two-party setting (that is, $n = 2$), the number of invocations of multiplication is 6. We note that their protocol needs to use different moduli in the groups of base and exponentiation in order to ensure correctness. This is a drawback when considering composition with other protocols (not only in theoretical sense but also in an implementa-

tion). Compared to their protocol, our protocol has same modulus in the groups of base and exponentiation.

In Table 1.1, we compare our work with the results by Ning et al. and Aly et al. in the two-party setting (all protocols without bit-decomposition), and the result by [DFK+06] which uses bit-decomposition. There, ℓ denotes the bit-length of the input. In the rows " [DFK+06]" and " [NX11]", we consider the case $\ell = 64$ when estimating the number of multiplications.

Although we obtain an efficient constant-round MPC protocol for an exponentiation functionality in the two-party setting, it is still an open problem to extend our protocol to the three or (more general) n-party setting. The main difficulty is to extend our QT protocol to the n-party setting efficiently. See Section 2.2.3 for the details.

1.5 Our Second Contribution: Multi-Key Verifiable Homomorphic Encryption

In this thesis, as a core cryptographic primitive for private and verifiable delegating computations in the multi-user setting, we propose *multi-key verifiable homomorphic encryption* (MVHE). Roughly, an MVHE scheme is a primitive has following features:[7]

- Each user U_i (having its secret key sk_i) to generate a ciphertext c_i of its message m_i.
- Receiving a function f (for a homomorphic evaluation) and users' ciphertexts $(c_i)_{i\in[n]}$, anyone can perform homomorphic evaluations and get an evaluated ciphertext c_f for the value $y = f(m_1, \cdots, m_n)$.
- By using all of the secret keys $(sk_i)_{i\in[n]}$, we can decrypt the evaluated ciphertext c_f and verify that the evaluated value y is correct as well as get the result.

In order to obtain an MVHE scheme, we also propose a new notion of *multi-key homomorphic encrypted authenticator* (MHEA), which is a multi-key variant of homomorphic encrypted authenticator (HEA) introduced by Lai et al. [LDPW14] as a building block.[8] We show that an MVHE scheme can be constructed by combining a multi-key homomorphic encryption (MHE) scheme [LTV12] and an MHEA scheme. As we will see below, MVHE plays an important role to realize private and verifiable delegating computations in the multi-client setting.

[7] Due to its nature, in a (M)VHE scheme, a user uses a secret key both for encryption and decryption procedures.

[8] Note that, when referring (multi-key) homomorphic (encrypted) authenticator, we only consider one requiring a secret key in the verification process, that is, the message authentication code setting.

We explain how to obtain a private and verifiable delegating computation in the multi-user setting by utilizing an MVHE scheme **MVHE** (with a multi-party computation (MPC) protocol with semi-honest security[9]). For clarity and concreteness, we assume the situation where there are many clients (with only weak computational devices) who want to get a (shared) prediction model, while keeping all their training data secret, and there is one (malicious) server with an enough computational power for generating the prediction model. More precisely, we consider a situation where there are multiple clients $(\mathsf{C}_i)_{i\in[n]}$ and a (computationally powerful) server S and each client C_i has its (secret) training data data_i and they hope to get an efficient prediction model $f(\mathsf{data}_1, \cdots, \mathsf{data}_n)$ computed from a (public) learning program f and their training data $(\mathsf{data}_i)_{i\in[n]}$. Here, we assume that all of the clients $(\mathsf{C}_i)_{i\in[n]}$ are semi-honest and the server S is malicious, where "semi-honest" means that the (possibly corrupted) clients follow the specified protocol and nothing is leaked in the transcript. As a preparation phase, each client generates their own secret key sk_i of **MVHE**, computes a ciphertext c_i of data_i using sk_i by itself, and sends the ciphertext c_i to S.[10] Then, given all users' ciphertexts $(c_i)_{i\in[n]}$, the server S performs the evaluation algorithm of **MVHE** on a (public) function f and ciphertexts $(c_i)_{i\in[n]}$ for obtaining a ciphertext c_f encrypting $f(\mathsf{data}_1, \cdots, \mathsf{data}_n)$ and returns to the users. Upon receiving an evaluated ciphertext c_f from S, the users run the decryption algorithm of **MVHE** (hardwired the evaluated ciphertext c_f) on a semi-honest MPC protocol given the secret keys $(sk_i)_{i\in[n]}$ as secret inputs. We note that this MPC procedure among clients does not depend on the size of f due to the succinctness of the evaluated ciphertext c_f. When this MPC procedure is done, all of the users can get the information of an evaluated prediction model $f(\mathsf{data}_1, \cdots, \mathsf{data}_n)$.[11]

1.5.1 Related Works

We briefly recall some previous related works on (fully) MHE, (fully) MHA, and private and verifiable delegating computation.

[9] We note that various MPC protocols with semi-honest security can be constructed based solely on secret sharing schemes [BGW88, GMW87].

[10] While in an actual MVHE scheme, in addition to secret keys, each user generates an (public) evaluation key ek_i used by a server S for executing homomorphic evaluations including a ciphertext generated on sk_i, we omit these evaluation keys for simplifying explanations.

[11] We note that since a (standard) MPC protocol does not have verifiability, we cannot realize private and verifiable delegating computations solely based on a MPC protocol.

Multi-Key Homomorphic Encryption. López-Alt, Tromer, and Vaikuntanathan [LTV12] proposed the notion of MHE and its concrete construction based on the NTRU lattice. Clear and McGoldrick [CM15] proposed a general transformation from (fully) HE into (fully) MHE based on the learning with errors (LWE) assumption over lattices. As a result, they obtain the first MHE scheme based on the LWE assumption. Mukherjee and Wichs [MW16] presented a new construction of MHE having a single round threshold decryption process based on the LWE assumption by simplifying the Clear et al.'s MHE scheme. Peikert and Shiehian [PS16] proposed a notion of multi-hop MHE, which enables that homomorphic evaluated ciphertexts can be reused in the following homomorphic evaluations involving additional users, and its construction based on the LWE assumption (with a restriction that the number of users are limited). Brakerski and Perlman [BP16] proposed a similar notion called fully dynamic MHE, which is different from multi-hop MHE in that the number of users are not a-priori bounded at the setup phase, and its construction based on the LWE assumption. Chen, Zhang, and Wang [CZW17] proposed the first multi-hop MHE scheme based on the ring LWE assumption. Recently, Ananth, Jain, Jin, and Malavolta [AJJM20] proposed the first MHE scheme in the plain model (which does not require any trusted setup phase) based on the LWE, ring LWE, and decisional small polynomial ratio assumption.

Multi-Key Homomorphic Authenticator. (M)HA is divided into two types depending on whether a verifier's is secret or not. In the former case, an MHA is also referred as multi-key homomorphic message authentication code (MHMAC), while in the latter case referred as multi-key homomorphic signature (MHS). Gennaro and Wichs [GW13] proposed the first fully HMAC scheme based on a (fully) HE scheme (and pseudorandom functions). Then, Gorbunov, Vaikuntanathan, Wichs [GVW15] proposed the first fully HS scheme based on the short integer solution (SIS) assumption. By extending the Gorbunov et al.'s HS scheme, Fiore, Mitrokotsa, Nizzardo, and Pagnin [FMNP16] proposed the first fully MHS scheme based on the SIS assumption. Lai, Tai, Wong, and Chow [LTWC18] introduced a new security notion called unforgeability under insider corruptions for MHS and showed that an MHS scheme satisfying such a strong security notion can be constructed from zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARK) [BCCT12, FN16]. Fiore and Pagnin [FP18] proposed a compiler called Matrioska which transforms any (single-key) HA scheme for polynomial-sized circuits into a fully MHA scheme.

Private and Verifiable Delegating Computation. Gennaro et al. [GGP10] firstly considered the formal notion of private and verifiable delegating computation (called verifiable

computation (VC) scheme for short) and proposed its construction based on a fully HE scheme and garbled circuits. Then, Parno, Raykova, and Vaikuntanathan [PRV12] (resp., Goldwasser, Kalai, Popa, Vaikuntanathan, and Zeldovich [GKP+13]) proposed the first VC scheme based on an attribute-based encryption scheme (resp., a succinct single-key functional encryption scheme). Lai et al. [LDPW14] proposed a VHE scheme as a building block for VC. Fiore, Gennaro, and Pastro [FGP14] proposed a generic construction of VC based on a fully HE scheme and a (non-private) VC scheme and its practically efficient instantiation. Recently, Fiore, Nitulescu, and Pointcheval [FNP20] extended the approach of [FGP14] to support public verifiability and the evaluation of more than quadratic functions. Moreover, Bois, Cascudo, Fiore, and Kim [BCFK21] proposed an improved protocol which solved the restriction on the modulus of the underlying HE scheme used in [FNP20].

Finally, as one of the most important related works, Choi et al. [CKKC13] proposed a cryptographic protocol called a multi-client VC scheme which allows multiple clients to outsource the computation of a function f of their inputs to a server. They construct a multi-client VC scheme by combining a Gennaro et al.'s (single-client) VC scheme [GGP10] and a proxy oblivious transfer protocol. Although their multi-client VC scheme is similar to our private and verifiable delegating computation in the multi-user setting, we have important differences between these. Firstly, before running the delegated computation, their multi-client VC scheme requires one client to execute a heavy computation depending on the size of a function f as setup. Compared to theirs, our protocol does not require any client to execute such a heavy setup phase. Secondly, while their multi-client VC scheme does not need any interaction among clients, our protocol needs to interact each other when running the decryption procedure since we execute a semi-honest MPC protocol. Compared to each other, both of them have (dis)advantages, we can say that these two notions are incomparable.[12]

1.6 Notations

In this thesis, we use the following notations. $x \leftarrow X$ denotes sampling an element x from a finite set X uniformly at random. $y \leftarrow \mathcal{A}(x;r)$ denotes that a probabilistic algorithm $\mathcal{A}$ outputs y for an input x using a randomness r, and we simply denote $y \leftarrow \mathcal{A}(x)$ when we need not write an internal randomness explicitly. Also, $x := y$ denotes that x is defined by y. λ denotes a security parameter. A function $f(\lambda)$ is a negligible function in λ, if $f(\lambda)$ tends to 0 faster than $\frac{1}{\lambda^c}$ for every constant $c > 0$. $\mathsf{negl}(\lambda)$ denotes an unspecified negligible function.

[12] One might see that these two primitives are same one except that they have different advantages.

PPT stands for probabilistic polynomial time.

Chapter 2

Efficient Two-Party Exponentiation from Quotient Transfer

In this chapter, we provide our two-party exponentiation protocol using quotient transfer. In Section 2.1, we review some notations and basic knowledge of secret-sharing based MPC. In Section 2.2, we provide the details of our exponentiation protocol.

2.1 Preliminaries

2.1.1 Notations

In this result, we use the following notations. p and p' denote safe prime numbers. n denotes the number of parties. Let $[x]_p$ denote a secretly shared input $x \in \mathbb{Z}_p$. Let P be the set of n parties and P_i the i-th party for $i = 1, \cdots, n$. The operation $+$ is a normal addition over integers. The congruence relation $a = kp + b$ is represented as $a \equiv b (\bmod p)$, where a, b, k, and p are integers.

2.1.2 Additive Secret Sharing

In this section, we introduce the definition of additive secret sharing. In general, an additive secret sharing scheme is defined over finite additive groups. Among them, we consider the case over $\mathbb{Z}_p$ (the set of integers modulo p). An additive secret sharing scheme over $\mathbb{Z}_p$ consists of the following two algorithms **Share** and **Reconstruct**.

- **Share** : Given a value $x \in \mathbb{Z}_p$ as input, this algorithm outputs shares $[x]_p = ([x]_p^1, \ldots, [x]_p^n)$ of x such that $[x]_p^1 + [x]_p^2 + \cdots + [x]_p^n \equiv x (\bmod p)$, where $[x]_p^i$ denotes P_i's share. All

shares are distributed uniformly at random in $\mathbb{Z}_p$ under the constraint that they sum to x. In the following, we use the notation $[x]_p \leftarrow \mathbf{Share}(x)$.

- **Reconstruct** : Given all n shares $[x]_p$ as input, this algorithm outputs a value $x = ([x]_p^1 + \cdots + [x]_p^n) \bmod p$.

Note that the requirement on the random distribution of shares implies that only given access to $n-1$ shares in $[x]_p$, the value x is information theoretically hidden.

An additive secret sharing scheme supports the following computations on shares.

- **Local operation** : Given shares $[a]_p, [b]_p$ and a scalar $\alpha \in \mathbb{Z}_p$, the parties can generate shares of $[a+b]_p$, $[\alpha a]_p$, and $[\alpha + a]_p$ using only local operations.
- **Multiplication** : Given shares $[a]_p$ and $[b]_p$, we assume the parties can generate $[ab]_p$ by invoking an ideal multiplication functionality $\mathcal{F}_{\mathrm{Mul}}([a]_p, [b]_p)$. This might be implemented using a multiplication protocol based on Beaver triples [Bea92]. In the following, we use the notation $[ab]_p \leftarrow [a]_p \cdot [b]_p$ to denote $[ab]_p \leftarrow \mathcal{F}_{\mathrm{Mul}}([a]_p, [b]_p)$.

2.1.3 A Model of Secure Two-Party Computation

In this section, we formally introduce two-party computation. A two-party computation is specified by a (possibly probabilistic) procedure referred to as a *functionality*. Denote $f : (\{0,1\}^*)^2 \to (\{0,1\}^*)^2$ as the two-ary functionality. Specifically, each party P_i can obtain distinct outputs $f_i(\mathbf{x})$ in general, where $f = (f_0, f_1)$ and $\mathbf{x} = (x_1, x_2)$ is a pair of inputs.

Definition of Security. The security of MPC is formalized by simulation-based security definitions. Roughly speaking, if there exist simulators who can generate the view of each party in the execution from given inputs and outputs, an MPC protocol is called *secure*. This formalization implies that each party learns nothing about other users' inputs from the execution of the protocol, except for the information that can be derived from outputs.

Definition 1 (Computational Indistinguishability)**.** *Two probability ensembles* $X = \{X(a,n)\}_{a\in\{0,1\}^*, n\in\mathbb{N}}$ *and* $Y = \{Y(a,n)\}_{a\in\{0,1\}^*, n\in\mathbb{N}}$ *are said to be computationally indistinguishable, denoted by* $X \overset{c}{\approx} Y$*, if for any non-uniform PPT algorithm* $\mathcal{D}$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for every* $a \in \{0,1\}^*$ *and every* $n \in \mathbb{N}$*,*

$$\left| \Pr_{X \leftarrow \mathcal{X}}[\mathcal{D}(X(a,n)) = 1] - \Pr_{Y \leftarrow \mathcal{Y}}[\mathcal{D}(Y(a,n)) = 1] \right| = \mathsf{negl}(\lambda).$$

Definition 2 (Security). *Let $f : (\{0,1\}^*)^2 \to (\{0,1\}^*)^2$ be a 2-ary functionality and let π be a two-party protocol for computing f. Let $\mathbf{x} = (x_0, x_1)$ be a pair of inputs. Let $\mathsf{View}_i^\pi(\mathbf{x}, \lambda)$ be the view of the party P_i during an execution of a protocol π on the input $\mathbf{x}$ and λ. Let $\mathsf{Output}^\pi(\mathbf{x})$ be the output of all parties from an execution of π. We say that a protocol π securely computes f in the presence of semi-honest adversaries, if there exist PPT algorithms $\mathcal{S}_1$ and $\mathcal{S}_2$ such that*

$$\{(\mathcal{S}_1(1^\lambda, x_1, f_1(\mathbf{x})), f(\mathbf{x}))\}_{\mathbf{x},\lambda} \overset{c}{\approx} \{(\mathsf{View}_1^\pi(\mathbf{x}, \lambda), \mathsf{Output}^\pi(\mathbf{x}, \lambda))\}_{\mathbf{x},\lambda}$$

and

$$\{(\mathcal{S}_2(1^\lambda, x_2, f_2(\mathbf{x})), f(\mathbf{x}))\}_{\mathbf{x},\lambda} \overset{c}{\approx} \{(\mathsf{View}_2^\pi(\mathbf{x}, \lambda), \mathsf{Output}^\pi(\mathbf{x}, \lambda))\}_{\mathbf{x},\lambda},$$

where $x_1, x_2 \in \{0,1\}^*$ and $|x_1| = |x_2|$.

In addition, we say that π securely computes f in the presence of semi-honest adversaries in the $\mathcal{F}$-hybrid model if π contains ideal calls to a trusted party computing a certain functionality $\mathcal{F}$.

Remark 1 (On local computations). *Note that the functionalities with only a local computation (i.e., a computation which needs no communication among parties) obviously satisfy the above Definition 2, since the view of such functionality is only the information that can be obtained from shares. Such a view leaks no information regarding inputs due to the security of the underlying secret sharing scheme.*

Universal Composability Framework. A stronger notion of security typically considered for MPC can be obtained via the Universal Composability (UC) framework, which is a general framework allowing arbitrary MPC protocols to be represented and analyzed. Protocols that are proven secure in the UC framework have the property that they maintain their security when run in parallel and concurrently with other secure and insecure protocols. In [KLR10], Kushilevitz, Lindell, and Rabin showed that under certain circumstances, stand-alone security as defined above (Definition 2), implies security in the UC framework:

Theorem 1 (Thm. 1.5 in [KLR10]). *Every protocol that is secure in the stand-alone model and has start synchronization and a straight-line black-box simulator is UC-secure under concurrent general composition (universal composition).*

In the above theorem, a "straight-line" simulator means that a non-rewinding simulator, and "start synchronization" means that the inputs of all parties are fixed before the execution begins (also called as "input availability"). All of the protocols considered in this thesis satisfies these conditions, and hence, Theorem 1 ensures that we obtain UC-security of our protocols.

2.1.4 Quotient Transfer Functionality

We now introduce the quotient transfer (QT) functionality $\mathcal{F}_{\mathrm{QT}}$ for a two-party protocol. This functionality plays a central role in our construction of an exponentiation protocol. Let $[a]_p = ([a]_p^1, [a]_p^2)$. Here, we have a value t satisfying $[a]_p^1 + [a]_p^2 = a + t \cdot p$ ($t \in \{0, 1\}$) over the integers. We define the QT functionality as

$$[t]_{p'} \leftarrow \mathcal{F}_{\mathrm{QT}}([a]_p, p').$$

Note that, if $[a]_p^1 + [a]_p^2 < p$, then $t = 0$, else $t = 1$. We emphasize that although the individual shares $[a]_p^i$ are elements in $\mathbb{Z}_p$, the addition considered above is over the integers.

2.1.5 Modulus Conversion Functionality

We introduce the modulus conversion functionality $\mathcal{F}_{\mathrm{Conv}}$. A modulus conversion functionality is a functionality that converts a share in $\mathbb{Z}_p$ into one in $\mathbb{Z}_{p'}$ (with $p \neq p'$). Let $x \in \mathbb{Z}_p$. We define $\mathcal{F}_{\mathrm{Conv}}$ as

$$[x]_{p'} \leftarrow \mathcal{F}_{\mathrm{Conv}}([x]_p, p').$$

2.1.6 Exponentiation Functionality

Here, we introduce a (public base) exponentiation functionality $\mathcal{F}_{\mathrm{EXP}}$. Let p be some prime. Let $a \in \mathbb{Z}_p$ and $x \in \mathbb{Z}_p$. We define $\mathcal{F}_{\mathrm{EXP}}$ as

$$[a^x \bmod p]_p \leftarrow \mathcal{F}_{\mathrm{EXP}}(a, [x]_p).$$

Note that, as we will consider additive secret sharing of x over $\mathbb{Z}_p$, we define the above exponentiation functionality for $x \in \mathbb{Z}_p$, whereas the exponent space typically considered for exponentiation in $\mathbb{Z}_p$ would be restricted to $\mathbb{Z}_{\phi(p)} = \mathbb{Z}_{p-1}$. However, the functionality remains well-defined for the extension $x \in \mathbb{Z}_p$.

2.2 Our Exponentiation Protocol

In this section, we propose our exponentiation protocol. We first introduce a new framework for an exponentiation protocol in Section 2.2.1. Then, we provide a modulus conversion protocol using a QT protocol in Section 2.2.2. Next, we provide a constrained QT protocol without bit-decomposition which only works on even numbers in Section 2.2.3. In the end, we introduce a concrete construction of our framework of an exponentiation protocol using

our constrained QT protocol and modulus conversion protocol (which is also obtained by our constrained QT protocol) in Section 2.2.4.

2.2.1 A New Framework for Exponentiation Protocol

In this section, we provide our new framework for an exponentiation protocol. Before describing our framework formally, we give its overview.

In the public base exponentiation setting, a naive idea for computing $a^x \bmod p$ is that each party P_i ($i \in \{0,1\}$) locally computes $a^{[x]_p^i} \bmod p$ and invokes a multiplication protocol to get a value $a^{[x]_p^0+[x]_p^1} \bmod p$. Here, a subtle point is that $[x]_p^0 + [x]_p^1$ is not always equal to x since there is a situation that $[x]_p^0 + [x]_p^1 = x + p$ holds. That is, we have the case $a^{[x]_p^0+[x]_p^1} \bmod p = a^{x+p} \bmod p = (a^x \cdot a^p) \bmod p$ and in this case, we should eliminate the term $a^p \bmod p$ to get a correct result $a^x \bmod p$. In order to solve this problem, we utilize a QT functionality $\mathcal{F}_{\mathrm{QT}}$. By using $\mathcal{F}_{\mathrm{QT}}$, we can know the secret shared values $[t]_p$ such that $[x]_p^0 + [x]_p^1 = x + t \cdot p$ and eliminate the term $a^p \bmod p$ correctly. Formally, our new framework $\mathbf{\Pi}_{\mathrm{EXP}}$ for $\mathcal{F}_{\mathrm{EXP}}$ is described in Algorithm 1.

Algorithm 1 Our framework for exponentiation protocol $\mathbf{\Pi}_{\mathrm{EXP}}$

Input: $a, [x]_p$

Output: $[o]_p$

1: Each $P_i (i \in \{0,1\})$ locally computes $y_i = a^{[x]_p^i} \bmod p$
2: Each $P_i (i \in \{0,1\})$ locally sets $[y_i]_p^i = y_i$ and $[y_i]_p^{1-i} = 0$
3: $[d]_p \leftarrow [y_0]_p \cdot [y_1]_p$
4: $[t]_p \leftarrow \mathcal{F}_{\mathrm{QT}}([x]_p, p)$
5: $[o_1]_p \leftarrow (1 - [t]_p)[d]_p$,
6: $[o_2]_p \leftarrow [t]_p[d]_p$
7: $[o]_p \leftarrow [o_1]_p + [o_2]_p(a)^{-p}$

Correctness. Here, we show the correctness of our framework of an exponentiation protocol $\mathbf{\Pi}_{\mathrm{EXP}}$.

Theorem 2. $\mathbf{\Pi}_{\mathrm{EXP}}$ *is correct in* $(\mathcal{F}_{\mathrm{Mul}}, \mathcal{F}_{\mathrm{QT}})$*-hybrid model.*

The protocol $\mathbf{\Pi}_{\mathrm{EXP}}$ is aimed at correctly computing $a^x \bmod p$, where $a \in \mathbb{Z}_p$ and $x \in \mathbb{Z}_p$. Each party P_i firstly computes $a^{[x]_p^i} \bmod p$ and sets $[y_i]_p^i = y_i$ and $[y_i]_p^{1-i} = 0$ locally, then utilizes a multiplication MPC protocol to compute $(a^{[x]_p^0}) \bmod p \cdot (a^{[x]_p^1}) \bmod p = a^{[x]_p^0+[x]_p^1} \bmod p$.

An obscure problem in this process is that we have two cases for the value x. Concretely, we have $[x]_p^0 + [x]_p^1 = x + t \cdot p$ ($t \in \{0, 1\}$). In the following proof, we have the correct value a^x in both of the cases $[x]_p^0 + [x]_p^1 \leq p$ and $[x]_p^0 + [x]_p^1 > p$.

Proof of Theorem 2. Let $a \in \mathbb{Z}_p$ and $x \in \mathbb{Z}_p$. Regarding the secret value x, we have the following two cases.

- In the case of $[x]_p^0 + [x]_p^1 \leq p$ (that is, we have $t = 0$ and $[x]_p^0 + [x]_p^1 = x$), each P_i locally computes $a^{[x]_p^i} \bmod p$, sets $[y_i]_p^i = y_i$ and $[y_i]_p^{1-i} = 0$, and uses $\mathcal{F}_{\text{Mul}}$ to compute d. From the correctness of $\mathcal{F}_{\text{Mul}}$, we have $d = a^{[x]_p^0 + [x]_p^1} \bmod p = a^x \bmod p$. Moreover, from the correctness of $\mathcal{F}_{\text{QT}}$, we get $t = 0$. Therefore, $o_1 = d, o_2 = 0$, and $o = a^x \bmod p$ hold.
- In the case of $[x]_p^0 + [x]_p^1 > p$ (that is, we have $t = 1$ and $[x]_p^0 + [x]_p^1 = x + p$), each P_i locally computes $a^{[x]_p^i} \bmod p$, sets $[y_i]_p^i = y_i$ and $[y_i]_p^{1-i} = 0$, and uses $\mathcal{F}_{\text{Mul}}$ to compute d. From the correctness of $\mathcal{F}_{\text{Mul}}$, we have $d = a^{[x]_p^0 + [x]_p^1} \bmod p = a^{x+p} \bmod p$. Moreover, from the correctness of $\mathcal{F}_{\text{QT}}$, we get $t = 1$. Therefore, $o_1 = 0, o_2 = d \cdot (a)^{-p} \bmod p = (a)^{x+p} \cdot (a)^{-p} \bmod p = a^x \bmod p$, and $o = a^x \bmod p$ hold. □ **(Theorem 2)**

Security. Then, we prove the security of our new framework.

Theorem 3. $\mathbf{\Pi}_{\text{EXP}}$ *can securely compute* $\mathcal{F}_{\text{EXP}}$ *in* $(\mathcal{F}_{\text{Mul}}, \mathcal{F}_{\text{QT}})$*-hybrid model.*

Proof of Theorem 3. We construct a separate simulator for each party (S_0 for the P_0's view and S_1 for the P_1's view, as in Definition 2). Consider the case that P_1 is corrupted. The view of P_1 can be written as:

$$\mathsf{View}_1^{\mathbf{\Pi}_{\text{EXP}}}(a, [x]_p) = ([d]_p^1, [t]_p^1, [o_2]_p^1, [o_1]_p^1, r),$$

where $[d]_p = [y_0]_p \cdot [y_1]_p$, $y_i = a^{[x]_p^i} \bmod p$ for $i \in \{0, 1\}$, $[y_i]_p^i = y_i$ and $[y_i]_p^{i-1} = 0$ for $i \in \{0, 1\}$, and r is a randomness used by P_1. We need to show that the simulator S_1 can generate the view of P_1. In the protocol, P_1 receives an input consisting of values a and $[x]_p^1$. Then, S_1 is given $(a, [x]_p^1, [o]_p^1)$ and works as follows:

1. S_1 chooses a uniform randomness r from $\mathbb{Z}_p$.
2. S_1 chooses a uniformly distributed random number $[d]_p^1$, $[t]_p^1$, and $[o_2]_p^1$ from $\mathbb{Z}_p$.
3. S_1 computes $[o_1]_p^1 = [o]_p^1 - [o_2]_p^1 \cdot a^{-1}$.

4. S_1 outputs $([d]_p^1, [t]_p^1, [o_2]_p^1, [o_1]_p^1, r)$.

Due to the security of the underlying additive secret sharing scheme, $[d]_p^1$, $[t]_p^1$, $[o_2]_p^1$, and r are uniformly at random in $\mathbb{Z}_p$. Moreover, since the shares of $[o]_p^1$ and $[o_2]_p^1$ are distributed randomly conditioned on $[o]_p^1 = [o_1]_p^1 + [o_2]_p^1 \cdot a^{-1}$, which is same as in the real execution. Thus, the distribution of $([d]_p^1, [t]_p^1, [o_2]_p^1, [o_1]_p^1)$ output by S_1 is equal to one which are given for P_1. Furthermore, due to the correctness of $\mathbf{\Pi}_{\mathrm{EXP}}$ (shown in Theorem 2), the output of $\mathbf{\Pi}_{\mathrm{EXP}}(a, [x]_p)$ is equal to the output of functionality $\mathcal{F}_{\mathrm{EXP}}(a, [x]_p)$. Hence, we have

$$\{(S_1(a, [x]_p^1, [o]_p^1), \mathcal{F}_{\mathrm{EXP}}(a, [x]_p)\} \quad \overset{\mathrm{c}}{\approx} \quad \{(\mathsf{View}_1^{\mathbf{\Pi}_{\mathrm{EXP}}}(a, [x]_p), \mathbf{\Pi}_{\mathrm{EXP}}(a, [x]_p)\}$$

Similar with above, in the case that P_0 is corrupted, we can also construct S_0 which can simulate the view of P_0. Therefore, $\mathbf{\Pi}_{\mathrm{EXP}}$ securely computes $\mathcal{F}_{\mathrm{EXP}}$ in $(\mathcal{F}_{\mathrm{Mul}}, \mathcal{F}_{\mathrm{QT}})$-hybrid model. □ **(Theorem 3)**

Remark 2 (Existing (inefficient) protocols over our framework)**.** *As mentioned in Section 1.4, we can realize* $\mathbf{\Pi}_{\mathrm{EXP}}$ *using existing primitives. Specifically, a part of the previous work [OWIO19] can be seen as a QT protocol, even though they did not explicitly define this as a QT protocol. However, since their protocol is based on bit-decomposition, the resulting* $\mathbf{\Pi}_{\mathrm{EXP}}$ *suffers from a large multiplication cost.*

Efficiency of Our Exponentiation Framework. Here, we give an analysis for the efficiency (round complexity and the number of invocations of multiplication) of our exponentiation framework. In the analysis for round complexity, an important point is that we can execute some procedures simultaneously in $\mathbf{\Pi}_{\mathrm{EXP}}$. Concretely, since the computation in the functionality $\mathcal{F}_{\mathrm{QT}}$ (Step. 4) does not depend on the previous steps, $\mathcal{F}_{\mathrm{QT}}$ can be executed with previous procedures in a parallel way. Taking into account this optimization, the round complexity of $\mathbf{\Pi}_{\mathrm{EXP}}$ is $\max(r_{\mathrm{QT}}, r_{\mathrm{Mul}}) + r_{\mathrm{Mul}}$, where r_{QT} and r_{Mul} represent the round complexity of $\mathcal{F}_{\mathrm{QT}}$ and $\mathcal{F}_{\mathrm{Mul}}$ respectively. Also, the number of invocations of multiplication is $i_{\mathrm{QT}} + 2 \cdot i_{\mathrm{Mul}}$, where i_{QT} and i_{Mul} represent the number of invocations of multiplication of $\mathcal{F}_{\mathrm{QT}}$ and $\mathcal{F}_{\mathrm{Mul}}$ respectively.

2.2.2 A Modulus Conversion Protocol Using Quotient Transfer Functionality

In this section, we provide a modulus conversion protocol which can change $x \in \mathbb{Z}_p$ to $x \in \mathbb{Z}_{p'}$. This modulus conversion protocol consists of the QT functionality and a transfer formula. By

using the QT functionality, we can know the secret shared values $[t]_{p'}$ such that $[x]_p^0 + [x]_p^1 = x + t \cdot p$. Then, the transfer formula change x into $x \in \mathbb{Z}_{p'}$ by eliminating the influence of overflow. Formally, our modulus conversion protocol is described in Algorithm 2.

Algorithm 2 Our modulus conversion protocol $\mathbf{\Pi}_{\mathrm{Conv}}$

Input: $[x]_p, p'$

Output: $[x]_{p'}$

1: $[t']_{p'} \leftarrow \mathcal{F}_{\mathrm{QT}}([x]_p, p')$

2: Each $P_i (i \in \{0,1\})$ sets $[x]_{p'}^i = [x]_p^i - [t']_{p'}^i \cdot p$

3: Output $[x]_{p'}$

Correctness. Firstly, we prove the correctness of $\mathbf{\Pi}_{\mathrm{Conv}}$.

Theorem 4. *$\mathbf{\Pi}_{\mathrm{Conv}}$ is correct in $\mathcal{F}_{\mathrm{QT}}$-hybrid model.*

Proof of Theorem 4. From the correctness of $\mathcal{F}_{\mathrm{QT}}$, we can obtain a correct $t' \in \{0,1\}$ satisfying $[x]_p^0 + [x]_p^1 = x + t' \cdot p$. Next, we obtain $[x]_{p'}^0 = [x]_p^0 - [t']_{p'}^0 \cdot p$ and $[x]_{p'}^1 = [x]_p^1 - [t']_{p'}^1 \cdot p$. Then, we have the following equation

$$\begin{aligned}
[x]_{p'}^0 + [x]_{p'}^1 &= [x]_p^0 - [t']_{p'}^0 \cdot p + [x]_p^1 - [t']_{p'}^1 \cdot p \bmod p' \\
&= [x]_p^0 + [x]_p^1 - ([t']_{p'}^0 + [t']_{p'}^1) \cdot p \bmod p' \\
&= x + t' \cdot p - ([t']_{p'}^0 + [t']_{p'}^1) \cdot p \bmod p'
\end{aligned}$$

Here, since we have $[t']_{p'}^0 + [t']_{p'}^1 = t'$ when $[t']_{p'}^0 + [t']_{p'}^1 \leq p'$ holds and $[t']_{p'}^0 + [t']_{p'}^1 = t' + p'$ when $[t']_{p'}^0 + [t']_{p'}^1 > p'$ holds, $[t']_{p'}^0 + [t']_{p'}^1 \bmod p'$ equals to t'. Then, $t' \cdot p - ([t']_{p'}^0 + [t']_{p'}^1) \cdot p$ is equal to 0 over modulus p'. Thus, we have $[x]_{p'}^0 + [x]_{p'}^1 = x \bmod p'$ and get shares $[x]_{p'}^0$ and $[x]_{p'}^0$ satisfying that the sum of them is equal to x over $\mathbb{Z}_{p'}$. Therefore, $\mathbf{\Pi}_{\mathrm{Conv}}$ is correct in $\mathcal{F}_{\mathrm{QT}}$-hybrid model.

□ **(Theorem 4)**

Security. Then, we prove the security of $\mathbf{\Pi}_{\mathrm{Conv}}$.

Theorem 5. *$\mathbf{\Pi}_{\mathrm{Conv}}$ is UC-secure in $\mathcal{F}_{\mathrm{QT}}$-hybrid model.*

Proof of Theorem 5. We construct a separate simulator for each party (S_0 for the P_0's view and S_1 for the P_1's view, as in Definition 2). Consider the case that P_1 is corrupted. The view of P_1 can be written as:

$$\mathsf{View}_1^{\mathbf{\Pi}_{\mathrm{Conv}}}([x]_p, p') = ([t]_{p'}^1, r),$$

where r is a randomness used by P_1. We need to show that the simulator S_1 can generate the view of P_1. In the protocol, P_1 receives an input consisting of values $[x]_p^1$ and p'. Then, S_1 is given $([x]_p^1, [x]_{p'}^1, p')$ and works as follows:

1. S_1 chooses a uniform randomness r from $\mathbb{Z}_p$.
2. S_1 chooses uniformly distributed random $[t]_{p'}^1$ from $\mathbb{Z}_{p'}$.
3. S_1 outputs $([t]_{p'}^1, r)$.

Due to the security of the underlying additive secret sharing scheme, r and $[t]_{p'}^1$ are uniformly at random in $\mathbb{Z}_{p'}$ in the real execution. Thus, the distribution of $[t]_{p'}^1$ output by S_1 is equal to one which are given for P_1. Furthermore, due to the correctness of $\mathbf{\Pi}_{\mathrm{EXP}}$ (shown in Theorem 4), the output of $\mathbf{\Pi}_{\mathrm{Conv}}([x]_p, p')$ is equal to the output of functionality $\mathcal{F}_{\mathrm{Conv}}([x]_p, p')$. Hence, we have

$$\{(S_1(p', [x]_p^1, [t]_{p'}^1), \mathcal{F}_{\mathrm{Conv}}([x]_p, p')\} \quad \overset{\mathrm{c}}{\approx} \quad \{(\mathsf{View}_1^{\mathbf{\Pi}_{\mathrm{Conv}}}([x]_p, p'), \mathbf{\Pi}_{\mathrm{Conv}}([x]_p, p')\}$$

Similar with above, in the case that P_0 is corrupted we can also construct S_0 which can simulate the view of P_0. Therefore, $\mathbf{\Pi}_{\mathrm{Conv}}$ securely computes $\mathcal{F}_{\mathrm{Conv}}$ in $(\mathcal{F}_{\mathrm{Mul}}, \mathcal{F}_{\mathrm{Conv}})$-hybrid model. Moreover, from Theorem 1, $\mathbf{\Pi}_{\mathrm{Conv}}$ is also UC-secure in $\mathcal{F}_{\mathrm{QT}}$-hybrid model, and thus Theorem 5 holds. □ **(Theorem 5)**

Efficiency of Our Modulus Conversion Protocol. Here, we give an analysis of the efficiency (round complexity and the number of invocations of multiplication) of our modulus conversion protocol $\mathbf{\Pi}_{\mathrm{Conv}}$. The round complexity is r_{QT}, where r_{QT} represents the round complexity of $\mathcal{F}_{\mathrm{QT}}$. Also, the number of invocations of multiplication is i_{QT}, where i_{QT} represents the number of invocations of multiplication of $\mathcal{F}_{\mathrm{QT}}$.

As mentioned in Section 2.2.3, both of the round complexity r_{QT} and the number of invocations of multiplication i_{QT} of our constrained QT protocol are 1. Thus, our modulus conversion protocol requires only 1 round and 1 invocation of multiplication. Compared to the most efficient modulus conversion protocol [KIM+18] which requires $\mathcal{O}(\log p')$ rounds and $\mathcal{O}(\log p')$ invocations of multiplication, we can see that our modulus conversion protocol is more efficient.

2.2.3 A Constrained Quotient Transfer Protocol without Bit-Decomposition

In this section, we provide our constrained QT protocol without the bit-decomposition protocol. Our core idea is that we can check whether $[x]_p^0 + [x]_p^1$ is bigger than p or not by using only the least significant bit (LSB) of the shares of even inputs. Namely, our QT protocol securely computes $\mathcal{F}_{\mathrm{QT}}$ if the input is an even number. Note that this restriction does not occur any problem when used in our exponentiation protocol. More specifically, the input of our exponentiation protocol is not restricted to even numbers. See Section 2.2.4 for the details.

We propose our constrained QT protocol as described in Algorithm 3. Let p and p' be odd primes. Regarding an input for Algorithm 3, let x be an even and b_i the LSB of $[x]_p^i$ for $i \in \{0, 1\}$.

Algorithm 3 Our constrained quotient transfer protocol $\mathbf{\Pi}_{\mathrm{QT}}$

Input: $[x]_p, p'$

Output: $[t]_{p'}$

1: Each $P_i (i \in \{0,1\})$ locally computes $b_i = \mathrm{LSB}([x]_p^i)$.
2: Each $P_i (i \in \{0,1\})$ locally sets $[b_i]_{p'}^i = b_i$ and $[b_i]_{p'}^{1-i} = 0$.
3: $[t]_{p'} = [b_0]_{p'} + [b_1]_{p'} - 2 \cdot [b_0]_{p'} \cdot [b_1]_{p'}$
4: Output $[t]_{p'}$

Remark 3 (On an extension to n-party setting)**.** *We can easily extend our two-party QT protocol into n-party protocol by executing the LSB checking (Step 3 in Algorithm 3) n times for an input x for judging how many times x exceeds the underlying modulus p. However, this naive approach requires n invocations of multiplication and the resulting protocol is not efficient. It is an interesting open question to extend our QT protocol into n-party setting efficiently (which derives an efficient constant-round n-party MPC protocol for an exponentiation functionality based on additive secret sharing).*

Correctness. Here, we prove the correctness of $\mathbf{\Pi}_{\mathrm{QT}}$.

Theorem 6. *If the input x is even, $\mathbf{\Pi}_{\mathrm{QT}}$ is correct in $\mathcal{F}_{\mathrm{Mul}}$-hybrid model.*

Proof of Theorem 6. Since x is even and p is prime, the last bit of x must be 0 and the last bit of p must be 1. Since $[x]_p^0 + [x]_p^1 = x + t \cdot p$ $(t \in \{0,1\})$ holds, $\mathrm{LSB}([x]_p^0) \oplus \mathrm{LSB}([x]_p^1)$

have two cases. If $[x]_p^0 + [x]_p^1 = x$ holds, then $\mathrm{LSB}([x]_p^0) \oplus \mathrm{LSB}([x]_p^1) = 0$ holds. Otherwise, $\mathrm{LSB}([x]_p^0) \oplus \mathrm{LSB}([x]_p^1) = 1$ holds. Thus, the value of $\mathrm{LSB}([x]_p^0) \oplus \mathrm{LSB}([x]_p^1)$ is equal to the value t. Moreover, due to the correctness of $\mathcal{F}_{\mathrm{Mul}}$, we can see that $[b_0]_{p'} + [b_1]_{p'} - 2 \cdot [b_0]_{p'} \cdot [b_1]_{p'}$ computes the shares of $t = \mathrm{LSB}([x]_p^0) \oplus \mathrm{LSB}([x]_p^1)$ over $\mathbb{Z}_{p'}$ in Step 3. Therefore, if the input x is even, $\mathbf{\Pi}_{\mathrm{QT}}$ is correct in $\mathcal{F}_{\mathrm{Mul}}$-hybrid model. □ **(Theorem 6)**

Security. Then, we prove the security of $\mathbf{\Pi}_{\mathrm{QT}}$.

Theorem 7. $\mathbf{\Pi}_{\mathrm{QT}}$ *is UC-secure in* $\mathcal{F}_{\mathrm{Mul}}$*-hybrid model.*

Proof of Theorem 7. We construct a separate simulator for each party (S_0 for the P_0's view and S_1 for the P_1's view, as in Definition 2). Consider the case that P_1 is corrupted. The view of P_1 can be written as:

$$\mathsf{View}_1^{\mathbf{\Pi}_{\mathrm{QT}}}([x]_p) = ([c]_{p'}^1, r),$$

where $[c]_{p'} = [b_1]_{p'} \cdot [b_0]_{p'}$, $b_i = \mathrm{LSB}([x]_p^i)$ for $i \in \{0, 1\}$, $[b_i]_{p'}^i = b_i$ and $[b_i]_{p'}^{1-i} = 0$ for $i \in \{0, 1\}$, and r is a randomness used by P_1.

We need to show that a simulator can generate the view of the P_1. In the protocol, P_1 receives an input consisting of values $([x]_p^1, p')$. Then, S_1 is given $([x]_p^1, [t]_{p'}^1, p')$ and works as follows:

1. S_1 chooses a uniform randomness r from $\mathbb{Z}_{p'}$.
2. S_1 generates $b_1 = \mathrm{LSB}([x]_p^1)$.
3. S_1 sets $[b_1]_{p'}^1 = b_1$ and $[b_0]_{p'}^1 = 0$.
4. S_1 computes $[c]_{p'}^1 = ([b_1]_{p'}^1 + [b_0]_{p'}^1 - [t]_{p'}^1) \cdot 2^{-1}$.
5. S_1 outputs $[c]_{p'}^1$ and r.

Since the randomness r is uniformly at random in $\mathbb{Z}_{p'}$ and the share $[c]_{p'}^1$ is distributed randomly conditioned on $[t]_{p'}^1 = [b_0]_{p'}^1 + [b_1]_{p'}^1 - 2 \cdot [c]_{p'}^1$, which are same as in the real execution. Thus, the distribution of $([c]_{p'}^1, r)$ output by S_1 is equal to one which are given for P_1. Furthermore, due to the correctness of $\mathbf{\Pi}_{\mathrm{QT}}$ (shown in Theorem 6), the output of $\mathbf{\Pi}_{\mathrm{QT}}([x]_p, p')$ is equal to the output of functionality $\mathcal{F}_{\mathrm{QT}}([x]_p, p')$. Hence, we have

$$\{(S_1([x]_p^1, [t]_{p'}^1, p'), \mathcal{F}_{\mathrm{QT}}([x]_p, p')\} \overset{\mathrm{c}}{\approx} \{(\mathsf{View}_1^{\mathbf{\Pi}_{\mathrm{QT}}}([x]_p, p'), \mathbf{\Pi}_{\mathrm{QT}}([x]_p, p')\}.$$

Similar with above, in the case that P_0 is corrupted, we can also construct S_0 which can simulate the view of P_0.

Therefore, $\mathbf{\Pi}_{\mathrm{QT}}$ securely computes $\mathcal{F}_{\mathrm{QT}}$ in $\mathcal{F}_{\mathrm{Mul}}$-hybrid model. Moreover, from Theorem 1, $\mathbf{\Pi}_{\mathrm{QT}}$ is also UC-secure in $\mathcal{F}_{\mathrm{Mul}}$-hybrid model, and thus Theorem 7 holds. □ **(Theorem 7)**

Efficiency of Our Constrained QT Protocol. Here, we give an analysis of the efficiency (round complexity and the number of invocations of multiplication) of our constrained QT protocol $\mathbf{\Pi}_{\mathrm{QT}}$. The round complexity is r_{Mul}, where r_{Mul} represents the round complexity of $\mathcal{F}_{\mathrm{Mul}}$. Also, the number of invocations of multiplication is i_{Mul}, where i_{Mul} represents the number of invocations of multiplication of $\mathcal{F}_{\mathrm{Mul}}$.

2.2.4 A Concrete Protocol in Our Framework

In this section, we propose our concrete protocol $\mathbf{\Pi}'_{\mathrm{EXP}}$ based on our framework $\mathbf{\Pi}_{\mathrm{EXP}}$ for the exponentiation functionality $\mathcal{F}_{\mathrm{EXP}}$. Here, $\mathbf{\Pi}_{\mathrm{EXP}}$ is realized by the concrete $\mathbf{\Pi}_{\mathrm{Conv}}$ in Section 2.2.2 using our constrained QT protocol $\mathbf{\Pi}_{\mathrm{QT}}$ in Section 2.2.3, and we denote this protocol $\mathbf{\Pi}'_{\mathrm{EXP}}$ which is described in Algorithm 4. In the following, let $a \in \mathbb{Z}_p$ and $x \in \mathbb{Z}_p$. As mentioned in Section 1.4, since we need to ensure the inputs are always even, our concrete exponentiation protocol requires a condition $2x < p$ for inputs x. We note that regarding the output $[o]_{p'}$ in Algorithm 4, if we want to convert it back to shares in $\mathbb{Z}_p$ (as opposed to in $\mathbb{Z}_{p'}$), we just need to apply modulus conversion to it again.

Algorithm 4 Our concrete exponentiation protocol $\mathbf{\Pi}'_{\mathrm{EXP}}$

Input: $a, [x]_p, p'$

Output: $[o]_{p'}$

1: $b := \sqrt{a}$, where $b \in \mathbb{Z}_{p'}$
2: $[2x]_p \leftarrow 2[x]_p$
3: **if** $p \neq p'$ **then**
4: $\quad [2x]_{p'} \leftarrow \mathbf{\Pi}_{\mathrm{Conv}}([2x]_p, p')$
5: $\quad v := [2x]_{p'}$
6: **else**
7: $\quad v := [2x]_p$
8: **end if**
9: Output $[o]_{p'} \leftarrow \mathbf{\Pi}_{\mathrm{EXP}}(b, v)$

Remark 4 (On the selection of modulus p')**.** *In Algorithm 4, we assume that a prime p' is known such that there exists an element b satisfying $b = \sqrt{a}$ in $\mathbb{Z}_{p'}$. Note that for a prime p, half of elements in $\mathbb{Z}_p$ have square roots in $\mathbb{Z}_p$, and if this is the case for a, the element $b \in \mathbb{Z}_p$ can be found via standard algorithms. In other words, for a randomly chosen a, the probability that setting $p' = p$ is sufficient, where p is the prime underlying the additive sharing of x, is $1/2$. However, if no square root for a exists in $\mathbb{Z}_p$, a different prime p' must be used. An appropriate p' might be found simply by trying a random prime p', test whether a has a square root in $\mathbb{Z}_{p'}$, and if not, try a new random prime p'. Under the assumption that an element $a \in \mathbb{Z}_{p'}$ has a square root in $\mathbb{Z}_{p'}$ with probability $1/2$ for a randomly chosen p', this approach will efficiently find an appropriate p' with overwhelming probability. We note that since a is assumed to be public, finding an appropriate p' can be done before the exponentiation protocol is executed, and might be based on publicly available information for commonly used values of a and p.*

Correctness. Here, we prove the correctness of our protocol $\mathbf{\Pi}'_{\mathrm{EXP}}$.

Theorem 8. *$\mathbf{\Pi}'_{\mathrm{EXP}}$ correctly computes $\mathcal{F}_{\mathrm{EXP}}$ in $(\mathcal{F}_{\mathrm{QT}}, \mathcal{F}_{\mathrm{Conv}}, \mathcal{F}_{\mathrm{Mul}})$-hybrid model if $2x < p'$ holds.*

Before showing our formal proof, we give some subtle points which happens in the proof. The difference with the protocol $\mathbf{\Pi}_{\mathrm{EXP}}$ is that, x is extended to $2x$ in $\mathbf{\Pi}'_{\mathrm{EXP}}$ since $\mathbf{\Pi}_{\mathrm{QT}}$ can only work over even numbers. (Here, since we need to compute $2x$ exactly without reducing in p', $2x < p'$ is required.) Thus, we compute $\sqrt{a}^{2x}$ instead of a^x directly, where $[2x]^0_{p'} + [2x]^1_{p'} = 2x + t \cdot p'$ $(t \in \{0, 1\})$. In the following, it is confirmed that both in the two cases, we can obtain correct result $a^x \bmod p$ in the end of $\mathbf{\Pi}'_{\mathrm{EXP}}$.

Proof of Theorem 8. Here, we consider the case that $p \neq p'$ holds. (In the case of $p = p'$, we just need to skip the process of $\mathbf{\Pi}_{\mathrm{Conv}}$.) First, $\mathbf{\Pi}_{\mathrm{Conv}}$ changes $2x \in \mathbb{Z}_p$ to $2x \in \mathbb{Z}_{p'}$. In $\mathbf{\Pi}_{\mathrm{Conv}}$ on input $2x$, from the correctness of $\mathbf{\Pi}_{\mathrm{QT}}$ and the fact that the input $2x$ is an even number, we can get a correct value $t \in \{0, 1\}$ satisfying $[2x]^0_{p'} + [2x]^1_{p'} = 2x + t \cdot p'$. From the correctness of $\mathbf{\Pi}_{\mathrm{Conv}}$, we obtain $[2x]^0_{p'}$ and $[2x]^1_{p'}$, where $[2x]^0_{p'} + [2x]^1_{p'} = 2x \bmod p'$. Moreover, each party P_i computes $y_i = b^{[2x]^i_{p'}} \bmod p$ and shares y_i to each other, where $b^2 \equiv a (\bmod p)$. Thus, from the correctness of $\mathbf{\Pi}_{\mathrm{EXP}}$ and the correctness of $\mathbf{\Pi}_{\mathrm{QT}}$, we can obtain a (correct) output $o = b^{[2x]^0_{p'} + [2x]^1_{p'}} \bmod p = a^x \bmod p$. □ **(Theorem 8)**

Security. Finally, we prove the security of $\mathbf{\Pi}'_{\mathrm{EXP}}$.

Theorem 9. $\mathbf{\Pi}'_{\mathrm{EXP}}$ *securely computes* $\mathcal{F}_{\mathrm{EXP}}$ *in* $(\mathcal{F}_{\mathrm{QT}}, \mathcal{F}_{\mathrm{Conv}}, \mathcal{F}_{\mathrm{Mul}})$*-hybrid model.*

Proof of Theorem 9. From the UC-security of the underlying $\mathbf{\Pi}_{\mathrm{Conv}}$ (Theorem 5), $\mathbf{\Pi}_{\mathrm{QT}}$ (Theorem 7), and the underlying multiplication protocol, we can easily see that $\mathbf{\Pi}'_{\mathrm{EXP}}$ securely computes $\mathcal{F}_{\mathrm{EXP}}$. □ **(Theorem 9)**

Efficiency of Our Concrete Exponentiation Protocol. Here, we give an analysis of the efficiency (round complexity and the number of invocations of multiplication) of our concrete exponentiation protocol $\mathbf{\Pi}'_{\mathrm{EXP}}$.

Firstly, we estimate the round complexity of $\mathbf{\Pi}'_{\mathrm{EXP}}$. Recall that the round complexity of our exponentiation framework, modulus conversion protocol, and constrained QT protocol is $\max(r_{\mathrm{QT}}, r_{\mathrm{Mul}}) + r_{\mathrm{Mul}}$, r_{QT}, and r_{Mul} respectively, where r_{QT} and r_{Mul} represent the round complexity of $\mathcal{F}_{\mathrm{QT}}$ and $\mathcal{F}_{\mathrm{Mul}}$ respectively. That is, instantiating our exponentiation framework and modulus conversion protocol by our constrained QT protocol, the round complexity is $2 \cdot r_{\mathrm{Mul}}$ and r_{Mul}, respectively. In the case that we need to convert the modulus p to another p', since $\mathbf{\Pi}'_{\mathrm{EXP}}$ calls one modulus conversion protocol and one exponentiation framework, the round complexity of our concrete exponentiation protocol is $3 \cdot r_{\mathrm{Mul}}$. In contrast, if we do not need to convert the modulus p to another p', since $\mathbf{\Pi}'_{\mathrm{EXP}}$ calls only one exponentiation framework, the round complexity of our concrete exponentiation protocol is $2 \cdot r_{\mathrm{Mul}}$. By using a standard multiplication protocol, the round complexity of the multiplication protocol r_{Mul} is 1. Thus, the round complexity of of our concrete exponentiation protocol is 3 in the former cast and 2 in the latter case.

Secondly, we estimate the number of invocations of multiplications of $\mathbf{\Pi}'_{\mathrm{EXP}}$. Recall that the the number of invocations of multiplications of our exponentiation framework, modulus conversion protocol, and constrained QT protocol is $i_{\mathrm{QT}} + 2 \cdot i_{\mathrm{Mul}}$, i_{QT}, and i_{Mul}, respectively. That is, instantiating our exponentiation framework and modulus conversion protocol by our constrained QT protocol, the number of invocations of multiplications is 3 and 1, respectively.

Chapter 3

Multi-Key Verifiable Homomorphic Encryption

In this chapter, we propose multi-key verifiable homomorphic encryption (MVHE). In Section 3.1, we provide preliminaries used in this chapter. In Section 3.2, as a core primitive for MVHE, we propose a new cryptographic primitive called multi-key homomorphic encrypted authenticator (MHEA). In Section 3.3, we provide the formalization of MVHE and its generic construction based on multi-key homomorphic encryption and MHEA.

3.1 Preliminaries

In this section, we provide some notations and the definition of multi-key homomorphic encryption (MHE).

3.1.1 Notation

In the following, we use the following notations. If n is a natural number, $[n]$ denotes the set of integers $\{1, \cdots, n\}$. Also, if a and b are integers such that $a \leq b$, $[a, b]$ denotes the set of integers $\{a, \cdots, b\}$. If $\mathcal{O}$ is a function or an algorithm and $\mathcal{A}$ is an algorithm, $\mathcal{A}^{\mathcal{O}}$ denotes that $\mathcal{A}$ has oracle access to $\mathcal{O}$.

3.1.2 Multi-Key Homomorphic Encryption

In this section, we review the definition of multi-key homomorphic encryption (MHE) and its IND-CPA security. In the following, for a polynomial $t := t(\lambda)$, sets $\mathcal{M}$ and $\mathcal{ID}$, and a

function family $\mathcal{F} := \{f : \mathcal{M}^t \to \mathcal{M}\}$, we define a program $\mathcal{P} = (f, id_1, \cdots, id_t)$ as a tuple of a function f and t identities $id_1, \cdots, id_t \in \mathcal{ID}$.

Definition 3 (Multi-Key Homomorphic Encryption)**.** *A MHE scheme* **MHE** *with a plaintext space $\mathcal{M}$ and an identity space $\mathcal{ID}$ consists of the following PPT algorithms.*

MHE.Setup: *The setup algorithm, given a security parameter 1^λ, outputs a public parameter pp. This parameter includes the description of an identity label space $\mathcal{ID}$, a message space $\mathcal{M}$, and a set of admissible functions $\mathcal{F}$. (The public parameter pp is an input to all of the following algorithms, even when not specified.)*

MHE.KG: *The key generation algorithm, given a public parameter pp and an identity id, outputs an evaluation key ek_{id} and a secret key sk_{id}.*[1]

MHE.Enc: *The encryption algorithm, given a secret key sk and a message $m \in \mathcal{M}$, outputs a ciphertext c.*

MHE.Dec: *The decryption algorithm, given a program $\mathcal{P}$, a set of secret keys $(sk_{id})_{id \in \mathcal{P}}$, and a ciphertext c, outputs a message $\tilde{m} \in \mathcal{M} \cup \{\perp\}$.*

MHE.Eval: *The homomorphic evaluation algorithm, given a function $f : \mathcal{M}^t \to \mathcal{M}$ and a set $\{(c_i, \mathbf{EKS}_i)\}_{i \in [t]}$, outputs an evaluated ciphertext c_f, where each $\mathbf{EKS}_i$ is a set of evaluations keys.*

As the basic properties for MHE, we require that **MHE** *satisfies ordinary correctness, evaluation correctness, and succinctness.*

Ordinary Correctness. *We say that an MHE scheme* **MHE** *satisfies ordinary correctness if*

$$\Pr[\mathbf{MHE.Dec}(\mathcal{I}_l, sk_{id}, \mathbf{MHE.Enc}(sk_{id}, m)) \neq m] = \mathsf{negl}(\lambda)$$

holds, where $\lambda \in \mathbb{N}$, $pp \leftarrow \mathbf{MHE.Setup}(1^\lambda)$, $m \in \mathcal{M}$, $id \in \mathcal{ID}$, $(ek_{id}, sk_{id}) \leftarrow \mathbf{MHE.KG}(pp, id)$, and $\mathcal{I}_l = (f_{id}, id)$ is an identity program.

[1] Without loss of generality, when we do not need to write an identity id explicitly, we simply denote $(pk, sk) \leftarrow \mathbf{MHE.KG}(pp)$.

Evaluation Correctness. *Let $\lambda \in \mathbb{N}$, $t := t(\lambda)$ be some polynomial, $pp \leftarrow \mathbf{MHE.Setup}(1^\lambda)$, a set of tuples of (honest) evaluation keys and secret keys $\{(ek_{id}, sk_{id})\}_{id\in\widetilde{\mathcal{ID}}}$ for some $\widetilde{\mathcal{ID}} \subseteq \mathcal{ID}$, and any set of triples of programs, plaintexts, and ciphertexts $\{(\mathcal{P}_i, m_i, c_i)\}_{i\in[t]}$ such that $m_i = \mathbf{MHE.Dec}(\mathcal{P}_i, (sk_{id})_{id\in\mathcal{P}_i}, c_i)$ holds for all $i \in [t]$. Let $f : \mathcal{M}^t \to \mathcal{M}$ be a function, $m^* = f(m_1, \cdots, m_t)$, $\mathcal{P}^* = f(\mathcal{P}_1, \cdots, \mathcal{P}_t)$, and $c^* \leftarrow \mathbf{MHE.Eval}(f, \{(c_i, \mathbf{EKS}_i)\}_{i\in[t]})$ where $\mathbf{EKS}_i = (ek_{id})_{id\in\mathcal{P}_i}$. We say that an MHE scheme* **MHE** *satisfies evaluation correctness if we have*

$$\Pr[\mathbf{MHE.Dec}(\mathcal{P}^*, (sk_{id})_{id\in\mathcal{P}^*}, c^*) \neq m^*] = \mathsf{negl}(\lambda).$$

Next, we will recall IND-CPA security for **MHE**.

Definition 4 (IND-CPA Security)**.** *Let* **MHE** *be an MHE scheme. Let $n := n(\lambda)$ be a polynomial in λ. Consider the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.*

1. *$\mathcal{C}$ chooses a challenge bit $b \leftarrow \{0,1\}$ and generates a public parameter $pp \leftarrow \mathbf{MHE.Setup}(1^\lambda)$ and keys $(ek_i, sk_i) \leftarrow \mathbf{MHE.KG}(pp)$ for all $i \in [n]$. Then, $\mathcal{C}$ sends pp and $(ek_1, \cdots, ek_n)$ to $\mathcal{A}$ and prepares lists $L_{ch} := \emptyset$ and $L_{corr} := \emptyset$.*

2. *$\mathcal{A}$ can make polynomially many challenge queries and corruption queries as follows:*

   Challenge Queries. *When $\mathcal{C}$ receives a challenge query (i, m_0, m_1) ($|m_0| = |m_1|$), it checks whether $i \in L_{corr}$ holds. If this is the case, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ computes $c \leftarrow \mathbf{MHE.Enc}(sk_i, m_b)$, sends c to $\mathcal{A}$, and appends i to L_{ch}.*

   Corruption Queries. *When $\mathcal{C}$ receives a corruption query j, it checks whether $j \in L_{ch}$ holds. If this is the case, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ sends sk_j to $\mathcal{A}$ and appends j to L_{corr}.*

3. *$\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.*

In this game, we define the advantage of the adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathbf{MHE},\mathcal{A}}(\lambda) := 2 \cdot \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

We say that **MHE** *satisfies IND-CPA security if for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathbf{MHE},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$ holds.*

3.2 Multi-Key Homomorphic Encrypted Authenticator

In this section, as a building block for our MVHE scheme, we provide a new cryptographic primitive called *multi-key homomorphic encrypted authenticator* (MHEA). One can see that an MHEA scheme is an extension of a multi-key homomorphic authenticator (MHA) scheme given by Fiore et al. [FMNP16]. Roughly, a MHA scheme enables users $(U_i)_{i\in[n]}$ (having their own (distinct) secret keys $(sk_i)_{i\in[n]}$) to generate authenticators σ_{m_i} for (possibly different) messages m_i. Then, based on the original authenticators $(\sigma_{m_i})_{i\in[n]}$, anybody (who does not know secret keys) can homomorphically perform an arbitrary program $\mathcal{P}$ over the authenticated messages $(m_i)_{i\in[n]}$ to generate a new *short* authenticator $\sigma_{\mathcal{P}(m_1,\cdots,m_n)}$ for the message $\mathcal{P}(m_1,\cdots,m_n)$, which ensures the validity of $y = \mathcal{P}(m_1,\cdots,m_n)$ as the output of $\mathcal{P}$. In addition to this property of an MHA scheme, an MHEA scheme has a privacy requirement ensuring that the information of an authenticated message is not revealed from the corresponding authenticator.

In the following, in Section 3.2.1, we firstly provide the syntax of MHEA. Then, in Section 3.2.2, we give the security notions for MHEA, unforgeability and privacy. Finally, in Section 3.2.3, we show a candidate of instantiation of MHEA. Although the syntax and unforgeability of MHEA is the same as one of MHA given by Fiore et al. [FMNP16], we provide their formal descriptions for completeness.

3.2.1 Syntax

In this section, we provide the syntax of MHEA. Firstly, we recall the definition of labeled programs given in [GW13].

Definition 5 (Labeled Programs)**.** *Let $f : \mathcal{M}^n \to \mathcal{M}$ be a function on n variables and $l_i \in \{0,1\}^*$ for $i \in [1,\cdots,n]$ the label of the i-th input of f. A labeled program $\mathcal{P}$ is defined as a tuple $(f, l_1, \cdots, l_n)$. A labeled program can be composed as follows. Given some labeled programs $\mathcal{P}_1,\cdots,\mathcal{P}_t$ and a function $g : \mathcal{M}^t \to \mathcal{M}$, the composed program $\mathcal{P}^*$ is obtained by evaluating g on the outputs of $\mathcal{P}_1,\cdots,\mathcal{P}_t$, and it is denoted as $\mathcal{P}^* = g(\mathcal{P}_1,\cdots,\mathcal{P}_t)$. (The labeled inputs with the same label are grouped together and considered as a unique input of $\mathcal{P}^*$.) Let $f_{id} : \mathcal{M} \to \mathcal{M}$ be the identity function and $l \in \{0,1\}^*$ be any label. We refer to $\mathcal{I}_l = (f_{id}, l)$ as an identity program with a label l. Note that a program $\mathcal{P} = (f, l_1, \cdots, l_n)$ can be expressed as the composition of n identity programs $f(\mathcal{I}_{l_1},\cdots,\mathcal{I}_{l_n})$.*

Depending on the definition of labeled programs, we can provide the syntax and its authentication correctness, evaluation correctness, and succinctness of MHEA as follows.

Definition 6 (Multi-Key Homomorphic Encrypted Authenticator)**.** *An MHEA scheme* **MHEA** *consists of the following five PPT algorithms.*

MHEA.Setup: *The setup algorithm, given the security parameter* 1^λ*, outputs a public parameter* pp*. This parameter consists of a description of a tag space* $\mathcal{T}$*, an identity space* $\mathcal{ID}$*, a message space* $\mathcal{M}$*, and a set of admissible functions* $\mathcal{F}$*. Given* $\mathcal{T}$ *and* $\mathcal{ID}$*, a label space is defined as* $\mathcal{L} := \mathcal{ID} \times \mathcal{T}$*. For a labeled program* $\mathcal{P} = (f, l_1, \cdots, l_t)$ *with labels* $l_i := (id_i, \tau_i) \in \mathcal{L}$ *for* $i \in [t]$*, we use* $id \in \mathcal{P}$ *as a compact notation for* $id \in \{id_1, \cdots, id_t\}$*. The public parameter* pp *is an input to all of the following algorithms, even when not specified.*

MHEA.KG: *The key generation algorithm, given a public parameter* pp *and an identity* id*, outputs a public evaluation key* ek_{id} *and a secret authentication key* sk_{id}*.*

MHEA.Auth: *The authentication algorithm, given an authentication key* sk_{id}*, a dataset identifier* Δ*, a message* m*, a label* $\ell = (id, \tau)$*, outputs an authenticator* σ_ℓ*.*

MHEA.Eval: *The evaluation algorithm, given a* t*-input function* $f : \mathcal{M}^t \to \mathcal{M}$ *and a set* $\{(\sigma_i, \mathbf{EKS}_i)\}_{i\in[t]}$ *where each* σ_i *is an authenticator and each* $\mathbf{EKS}_i$ *is a set of evaluation keys, outputs an evaluated authenticator* σ^**.*

MHEA.Ver: *The verification algorithm, given a labeled program* $\mathcal{P} = (f, l_1, \cdots, l_t)$*, a dataset identifier* Δ*, a set of authentication keys* $\{sk_{id}\}_{id\in\mathcal{P}}$ *corresponding to the identities involved in the program* $\mathcal{P}$*, a message* m *and an authenticator* σ*, outputs* 1 *(meaning "accept") or* 0 *(meaning "reject").*

Authentication Correctness. We say that an MHEA scheme **MHEA** satisfies authentication correctness if we have

$$\Pr[\mathbf{MHEA.Ver}(\mathcal{I}_l, \Delta, sk_{id}, m, \sigma) = 0] = \mathsf{negl}(\lambda),$$

where $pp \leftarrow \mathbf{MHEA.Setup}(1^\lambda)$, $id \in \mathcal{ID}$, $\tau \in \mathcal{T}$, $l = (id, \tau) \in \mathcal{L}$, $m \in \mathcal{M}$, $f_{id} \in \mathcal{F}$, $\mathcal{I}_l := (f_{id}, l)$, $(ek_{id}, sk_{id}) \leftarrow \mathbf{MHEA.KG}(pp, id)$, and $\sigma \leftarrow \mathbf{MHEA.Auth}(sk_{id}, \Delta, l, m)$.

Evaluation Correctness. Intuitively, we say that an MHEA scheme **MHEA** satisfies evaluation correctness when we run the evaluation algorithm on signatures $(\sigma_1, \cdots, \sigma_t)$ such that each σ_i verifies for m_i as the output of a labeled program $\mathcal{P}_i$ over a dataset Δ, the output

signature σ verifying for the message $f(m_1, \cdots, m_t)$ which is the output of the composed program $f(\mathcal{P}_1, \cdots, \mathcal{P}_t)$ over the dataset Δ.

More formally, let $t := t(\lambda)$ be some polynomial in λ, a public parameter pp from $\mathbf{MHEA.Setup}(1^\lambda)$, a set of key pairs $\{(ek_{id}, sk_{id})\}_{id \in \widetilde{\mathcal{ID}}}$ for some $\widetilde{\mathcal{ID}} \subseteq \mathcal{ID}$, a dataset Δ, a function $g : \mathcal{M}^t \to \mathcal{M} \in \mathcal{F}$, and any set $\{\mathcal{P}_i, m_i, \sigma_i)\}_{i \in [t]}$ such that $\mathbf{MHEA.Ver}(\mathcal{P}_i, \Delta, \{sk_{id}\}_{id \in \mathcal{P}_i}, m_i, \sigma_i) = 1$ for all $i \in [t]$. Let $m^* = g(m_1, \cdots, m_t)$, $\mathcal{P}^* = g(\mathcal{P}_1, \cdots, \mathcal{P}_t)$, and $\sigma^* \leftarrow \mathbf{MHEA.Eval}(f, \{(\sigma_i, \mathbf{EKS}_i)\}_{i \in [t]})$ where $\mathbf{EKS}_i = \{ek_{id}\}_{id \in \mathcal{P}_i}$. We say that **MHEA** is evaluation correct if

$$\Pr[\mathbf{MHEA.Ver}(\mathcal{P}^*, \Delta, \{sk_{id}\}_{id \in \mathcal{P}^*}, m^*, \sigma^*) = 0] = \mathsf{negl}(\lambda)$$

holds.

Succinctness. Intuitively, an MHEA scheme **MHEA** is succinct if the size of every authenticator depends only logarithmically on the size of a dataset. Here, we allow authenticator to depend on the number of keys involved in the computation.

More formally, let $pp \leftarrow \mathbf{MHEA.Setup}(1^\lambda)$, $m_1, \cdots, m_t \in \mathcal{M}$, $id_1, \cdots, id_t \in \mathcal{ID}$, $\tau_1, \cdots, \tau_t \in \mathcal{T}$, $\mathcal{P} = (f, l_1, \cdots, l_t)$ with $l_i = (id_i, \tau_i)$ for all $i \in [t]$, $(ek_{id}, sk_{id}) \leftarrow \mathbf{MHEA.KG}(pp, id)$ for all $id \in \mathcal{P}$, and $\sigma_i \leftarrow \mathbf{MHEA.Auth}(sk_{id_i}, \Delta, l_i, m_i)$ for all $i \in [t]$. We say that **MHEA** is succient if there is a fixed polynomial p such that $|\sigma^*| = p(\lambda, n, \log t)$ where $\sigma^* \leftarrow \mathbf{MHEA.Eval}(g, \{(\sigma_i, \mathbf{EKS}_i)\}_{i \in [t]})$ and $n = |id \in \mathcal{P}|$.

3.2.2 Security Definitions

In this section, we provide the definitions of unforgeability and privacy for MHEA.

Definition 7 (Unforgeability for MHEA)**.** *Let* **MHEA** *be an MHEA scheme. We define the unforgeability game between a challenger* $\mathcal{C}$ *and an adversary* $\mathcal{A}$ *as follows.*

1. $\mathcal{C}$ *generates a public parameter* $pp \leftarrow \mathbf{MHEA.Setup}(1^\lambda)$*, sends* pp *to* $\mathcal{A}$*, and prepares lists* $L_{key} := \emptyset$ *and* $L_{corr} := \emptyset$*.*

2. $\mathcal{A}$ *may adaptively make polynomially many authentication queries, verification queries, and corruption queries.*

   Authentication Queries. *When* $\mathcal{C}$ *receives* (Δ, l, m)*, where* Δ *is a dataset identifier,* $l = (id, \tau)$ *is a label in* $\mathcal{ID} \times \mathcal{T}$ *and* $m \in \mathcal{M}$*, it answers as follows:*

(a) *If* (Δ, l, m) *is the first query for the dataset* Δ, $\mathcal{C}$ *initializes an empty list* $L_\Delta := \emptyset$ *and proceeds as follows.*

(b) *If* $(id, \cdot) \notin L_{key}$ *holds,* $\mathcal{C}$ *generates keys* $(ek_{id}, sk_{id}) \leftarrow \mathbf{MHEA.KG}(pp, id)$ *(that are implicitly assigned to identity* id*), gives* ek_{id} *to* $\mathcal{A}$*, and updates the list* $L_{key} := L_{key} \cup \{(id, sk_{id})\}$.

(c) *If* (Δ, l, m) *is such that* $(l, m) \notin L_\Delta$, $\mathcal{C}$ *computes* $\sigma_l \leftarrow \mathbf{MHEA.Auth}(sk_{id}, \Delta, l, m)$ *(note that* $\mathcal{C}$ *has already generated keys for the identity* id*), returns* σ_l *to* $\mathcal{A}$*, and updates the list* $L_\Delta \leftarrow L_\Delta \cup \{(l, m)\}$.

(d) *If* (Δ, l, m) *is such that* $(l, \cdot) \in L_\Delta$ *(which means that the adversary had already made a query* (Δ, l, m') *for some message* m'*),* $\mathcal{C}$ *ignores the query.*

Verification Queries. *When* $\mathcal{C}$ *receives* $(\mathcal{P}, \Delta, m, \sigma)$*, it answers as follows:*

(a) *If* $(id, \cdot) \notin L_{key}$ *holds for some* $id \in \mathcal{P}$*, then* $\mathcal{C}$ *generates* $(ek_{id}, sk_{id}) \leftarrow \mathbf{MHEA.KG}(pp, id)$ *and updates the list* $L_{key} := L_{key} \cup \{(id, sk_{id})\}$.

(b) $\mathcal{C}$ *returns* $v \leftarrow \mathbf{MHEA.Ver}(\mathcal{P}, \Delta, \{sk_{id}\}_{id \in \mathcal{P}}, m, \sigma)$ *to* $\mathcal{A}$ *(using the keys in* L_{key}*).*

Corruption Queries. *When* $\mathcal{C}$ *receives an identity* id*, it answers as follows:*

(a) *If* $(id, \cdot) \notin L_{key}$ *holds, then* $\mathcal{C}$ *generates* $(ek_{id}, sk_{id}) \leftarrow \mathbf{MHEA.KG}(pp, id)$ *and updates the list* $L_{key} := L_{key} \cup \{(id, sk_{id})\}$.

(b) $\mathcal{C}$ *returns* sk_{id} *to* $\mathcal{A}$ *(stored in* L_{key}*) and updates* $L_{corr} := L_{corr} \cup \{id\}$.

3. $\mathcal{A}$ *outputs a tuple* $(\mathcal{P}^*, \Delta^*, m^*, \sigma^*)$*, where* $\mathcal{P}^* = (f^*, l_1^*, \cdots, l_t^*)$. $\mathcal{C}$ *outputs* 1 *if the tuple returned by* $\mathcal{A}$ *satisfies the conditions of an event* **Forge** *(defined below), and* 0 *otherwise.*

In the above game, we say that an event **Forge** *occurs if* $\mathbf{MHEA.Ver}(\mathcal{P}^*, \Delta^*, \{sk_{id}\}_{id \in \mathcal{P}^*}, m^*, \sigma^*) = 1$ *for all* $id \in \mathcal{P}^*$, $id \notin L_{corr}$*, and either one of the following condition holds:*

Type 1: L_{Δ^*} *has not been initialized during the game (i.e., the dataset* Δ^* *was never queried).*

Type 2: *For all* $i \in [t]$, $\exists (l_i^*, m_i) \in L_{\Delta^*}$*, but* $m^* \neq f^*(m_1, \cdots, m_t)$*. (i.e.,* m^* *is not the correct output of* $\mathcal{P}^*$ *when executed over previously authenticated messages.)*

Type 3: *There exists a label* l_i^* *such that* $(l_i^*, \cdot) \notin L_{\Delta^*}$ *for some* $i \in [t]$*. (i.e.,* $\mathcal{A}$ *never made a query with the label* l_i^**.)*

We say that an MHEA scheme **MHEA** *satisfies unforgeability if for any PPT adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathsf{unf}}_{\mathbf{MHEA},\mathcal{A}}(\lambda) := \Pr[\mathbf{Forge}] = \mathsf{negl}(\lambda)$$

holds.

Definition 8 (Privacy for MHEA)**.** *Let* **MHEA** *be an MHEA scheme. We define the privacy game between a challenger* $\mathcal{C}$ *and an adversary* $\mathcal{A}$ *as follows.*

1. $\mathcal{C}$ *chooses a challenge bit* $b \leftarrow \{0,1\}$*, runs* $\mathbf{MHEA}.\mathbf{Setup}(1^\lambda)$ *to obtain a public parameter* pp*, sends* pp *to* $\mathcal{A}$*, and prepares lists* $L_{key} := \emptyset$*,* $L_{ch} := \emptyset$*, and* $L_{corr} := \emptyset$*.*

2. $\mathcal{A}$ *can polynomially many challenge queries and corruption queries as follows:*

   Challenge Queries. *When* $\mathcal{C}$ *receives a triple* (Δ, l, m_0, m_1)*, where* Δ *is a dataset identifier,* $l = (id, \tau)$ *is a label in* $\mathcal{ID} \times \mathcal{T}$*, and* $m \in \mathcal{M}$*, it proceeds as follows:*

   (a) *If* (Δ, l, m_0, m_1) *is the first query for the dataset* Δ*,* $\mathcal{C}$ *initializes an empty list* $L_\Delta = \emptyset$ *and proceeds as follows.*

   (b) *If* (Δ, l, m_0, m_1) *is the first query with identity* id *(that is,* $(id, \cdot, \cdot) \notin L_{key}$*),* $\mathcal{C}$ *generates* $(ek_{id}, sk_{id}) \leftarrow \mathbf{MHEA}.\mathbf{KG}(pp, id)$ *(that are implicitly assigned to the identity* id*), gives* ek_{id} *to* $\mathcal{A}$*, appends* (id, sk_{id}) *to* L_{key}*, and proceeds as follows.*

   (c) *If* (Δ, l, m_0, m_1) *is such that* $(l, m) \notin L_\Delta$*,* $\mathcal{C}$ *checks whether* $id \in L_{corr}$ *holds. If this is the case, then* $\mathcal{C}$ *returns* $\perp$ *to* $\mathcal{A}$*. Otherwise,* $\mathcal{C}$ *computes* $\sigma_l \leftarrow \mathbf{MHEA}.\mathbf{Auth}(sk_{id}, \Delta, l, m_b)$ *(note that* $\mathcal{C}$ *has already generated keys for the identity* id*), gives* σ_l *to* $\mathcal{A}$*, and updates the lists* $L_\Delta \leftarrow L_\Delta \cup \{(l, m)\}$ *and* $L_{ch} \leftarrow L_{ch} \cup \{id\}$*.*

   (d) *If* (Δ, l, m_0, m_1) *is such that* $(l, \cdot) \in L_\Delta$ *(which means that* $\mathcal{A}$ *had already made a query* (Δ, l, m') *for some plaintext* m'*),* $\mathcal{C}$ *ignores the query.*

   Corruption Queries. *When* $\mathcal{C}$ *receives a corruption query* id*, it answers as follows:*

   (a) $\mathcal{C}$ *checks whether* $id \in L_{ch}$ *holds. If this is the case, then* $\mathcal{C}$ *returns* $\perp$ *to* $\mathcal{A}$*.*

   (b) $\mathcal{C}$ *generates keys* $(ek_{id}, sk_{id}) \leftarrow \mathbf{MHEA}.\mathbf{KG}(pp, id)$*, gives* sk_{id} *to* $\mathcal{A}$*, and appends* (id, sk_{id}) *to* L_{key} *and* id *to* L_{corr}*.*

3. $\mathcal{A}$ *outputs a bit* $b' \in \{0,1\}$*.*

In the above game, we define the advantage of the adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MHEA},\mathcal{A}}(\lambda) := 2 \cdot \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

We say that **MHEA** *satisfies privacy if for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MHEA},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$ holds.*

3.2.3 Instantiation

In this section, we show a candidate of instantiation of MHEA. An instantiation of MHEA can be obtained by applying the compiler "Matrioska" [FP18], which turns any (sufficiently expressive) single-key homomorphic authenticator (HA) into a multi-key one, to the existing single-key fully HA scheme proposed by Gennaro and Wichs [GW13]. In the following, we explain about this instantiation.

As shown by Lai et al. [LDPW14], Gennaro et al.'s HA scheme satisfies privacy in the above sense, that is, their HA scheme is already a HEA scheme due to its construction. More precisely, their scheme is constructed based on a (single-key) fully homomorphic encryption (FHE) scheme and a pseudorandom function (PRF) F. Roughly, an authenticator σ on a message m and a label τ generated in their scheme consists of $(c_1, \cdots, c_\lambda, \nu)$, where c_i is a FHE ciphertext of the message m for $i \in [\lambda]$ and $\nu = F(\tau)$ is a value independent of m. Obviously, if the underlying FHE scheme satisfies standard IND-CPA security, then each c_i does not leak any information of m, that is, their scheme satisfies privacy.

We note that, as mentioned in [GW13], the complexity of the verification algorithm of this HEA scheme requires at least that of executing the evaluated program $\mathcal{P}$. That is, if we apply this HEA scheme to verifiable delegating computation without any efficiency improvement on the verification complexity, clients are required to perform some heavy computation which is larger than the size of $\mathcal{P}$. This is not preferable to the setting of verifiable delegating computation. Gennaro et al. showed as a solution that their construction can be extended to an HEA scheme whose verification complexity is independent of the size of $\mathcal{P}$ by utilizing succinct non-interactive arguments for polynomial-time deterministic computations (SNARGs for P). Roughly, using the fact that the heavy computation (included in the verification step) is independent of the (authenticated) message, their trick is to delegate even the above heavy computation to the server as a homomorphic evaluation based on SNARGs for P. From the recent work [CJJ21], SNARGs for P can be constructed based on the learning with errors (LWE) assumption over lattices. Thus, this problem on efficiency is solved.

By applying the compiler Matrioska to the above (single-key) HEA scheme, we obtain the first MHEA scheme for all polynomial-sized circuits. In fact, due to its mechanism, Matrioska inherits the privacy of a single-key HEA scheme to the (obtained) MHEA scheme. We note that, due to the restriction due to the underlying single-key HEA scheme and Matrioska, our MHEA scheme (i) is resilient to only fixed a-priori bounded number of verification queries and (ii) supports only constant number of users. These restrictions on our MHEA scheme are inherited to our MVHE scheme given in Section 3.3. We leave as an interesting open problem to examine how to construct a fully secure HEA scheme for all polynomial-sized circuits overcoming these restrictions.

3.3 Multi-Key Verifiable Homomorphic Encryption

In this section, we introduce our target cryptographic primitive called *multi-key verifiable homomorphic encryption* (MVHE). Roughly, an MVHE scheme is a multi-key variant of verifiable homomorphic encryption (VHE). An MVHE scheme allows each user U_i ($i \in [n]$) (having its secret key sk_i) to generate a ciphertext c_i of its message m_i. Then, receiving a polynomial-sized function f and their ciphertexts $(c_i)_{i\in[n]}$, anyone can perform homomorphic evaluations and get an evaluated ciphertext c_f encrypting the value $y = f(m_1, \cdots, m_n)$. By using all of the secret keys $(sk_i)_{i\in[n]}$, we can decrypt the evaluated ciphertext c_f and verify that the evaluated value y is correct as well as get the value y.

In the following, in Section 3.3.1, we provide a formal syntax and security definitions (privacy and unforgeability) of MVHE. Then, in Section 3.3.2, we give our construction of MVHE based on an MHE scheme and an MHEA scheme. Finally, in Section 3.3.3, we provide the security proofs for our MVHE scheme.

3.3.1 Formalization

In this section, we provide the formal definition of MVHE. Firstly, we give the syntax of MVHE.

Definition 9 (Multi-Key Verifiable Homomorphic Encryption)**.** *An MVHE scheme* **MVHE** *consists of the following PPT algorithms.*

MVHE.Setup: *The setup algorithm, given the security parameter* 1^λ*, outputs a public parameter* pp*. This parameter consists of a description of a tag space* $\mathcal{T}$*, an identity space* $\mathcal{ID}$*, a plaintext space* $\mathcal{M}$*, and a set of admissible functions* $\mathcal{F}$*.*

Given $\mathcal{T}$ and $\mathcal{ID}$, a label space is defined as $\mathcal{L} := \mathcal{ID} \times \mathcal{T}$. For a labeled program $\mathcal{P} = (f, l_1, \cdots, l_t)$ with labels $l_i := (id_i, \tau_i) \in \mathcal{L}$ for any $i \in [t]$, we use $id \in \mathcal{P}$ as compact notation for $id \in \{id_1, \cdots, id_t\}$. The public parameter pp is an input to all of the following algorithms, even when not specified.

MVHE.KG: *The key generation algorithm, given a public parameter pp and an identity id, outputs an evaluation key ek_{id} and a secret key sk_{id}.*[2]

MVHE.Enc: *The encryption algorithm, given a secret key sk, a dataset identifier Δ, a label $l = (id, \tau)$, and a message $m \in \mathcal{M}$, outputs a ciphertext c.*

MVHE.Dec: *The decryption algorithm, given a labeled program $\mathcal{P}$, a dataset identifier Δ, a set of secret keys $(sk_{id})_{id \in \mathcal{P}}$, and a ciphertext c, outputs a message $\tilde{m} \in \mathcal{M} \cup \{\perp\}$.*

MVHE.Eval: *The homomorphic evaluation algorithm, given a function $f : \mathcal{M}^t \to \mathcal{M}$ and a set $\{(c_i, \mathbf{EKS}_i)\}_{i \in [t]}$, outputs an evaluated ciphertext c_f, where each $\mathbf{EKS}_i$ is a set of evaluations keys.*

As the basic properties for MVHE, we require that **MVHE** *satisfies ordinary correctness, evaluation correctness, and succinctness.*

Ordinary Correctness. *We say that an MVHE scheme* **MVHE** *satisfies ordinary correctness if*

$$\Pr[\mathbf{MVHE.Dec}(\mathcal{I}_l, \Delta, sk_{id}, \mathbf{MVHE.Enc}(sk_{id}, \Delta, l, m)) \neq m] = \mathsf{negl}(\lambda)$$

holds, where $\lambda \in \mathbb{N}$, $pp \leftarrow \mathbf{MVHE.Setup}(1^\lambda)$, $m \in \mathcal{M}$, $id \in \mathcal{ID}$, $\tau \in \mathcal{T}$, $l = (id, \tau)$, Δ, $(ek_{id}, sk_{id}) \leftarrow \mathbf{MVHE.KG}(pp, id)$, and $\mathcal{I}_l = (f_{id}, id)$ is an identity program.

Evaluation Correctness. *Let $\lambda \in \mathbb{N}$, $t := t(\lambda)$ be some polynomial in λ, $pp \leftarrow \mathbf{MVHE.Setup}(1^\lambda)$, a set of tuples of (honest) evaluation keys and secret keys $\{(ek_{id}, sk_{id})\}_{id \in \widetilde{\mathcal{ID}}}$ for some $\widetilde{\mathcal{ID}} \subseteq \mathcal{ID}$, and any set of triples of programs, plaintexts, and ciphertexts $\{(\mathcal{P}_i, m_i, c_i)\}_{i \in [t]}$ such that $m_i = \mathbf{MVHE.Dec}(\mathcal{P}_i, \Delta, (sk_{id})_{id \in \mathcal{P}_i}, c_i)$ holds for all $i \in [t]$. Let $f : \mathcal{M}^t \to \mathcal{M} \in \mathcal{F}$ be a function, $m^* = f(m_1, \cdots, m_t)$, $\mathcal{P}^* = f(\mathcal{P}_1, \cdots, \mathcal{P}_t)$, and $c^* \leftarrow \mathbf{MVHE.Eval}(f,$*

[2]Without loss of generality, when we do not need to write an identity *id* explicitly, we simply denote $(ek, sk) \leftarrow \mathbf{MVHE.KG}(pp)$. Moreover, we assume that *ek* can be computed from *sk* efficiently.

$\{(c_i, \mathbf{EKS}_i)\}_{i\in[t]})$ *where each* $\mathbf{EKS}_i = (ek_{id})_{id\in\mathcal{P}_i}$. *We say that an MVHE scheme* **MVHE** *satisfies evaluation correctness if*

$$\Pr[\mathbf{MVHE.Dec}(\mathcal{P}^*, \Delta, (sk_{id})_{id\in\mathcal{P}^*}, c^*) \neq m^*] = \mathsf{negl}(\lambda)$$

holds.

Succinctness. *Let* $t := t(\lambda)$ *be some polynomial in* λ, $pp \leftarrow \mathbf{MVHE.Setup}(1^\lambda)$, $\mathcal{P} = (f, l_1, \cdots, l_t)$ *with* $l_i = (id_i, \tau_i) \in \mathcal{L}$, $(ek_{id}, sk_{id}) \leftarrow \mathbf{MVHE.KG}(pp, id)$ *for any* $id \in \mathcal{P}$, $m_i \in \mathcal{M}$, *and* $c_i \leftarrow \mathbf{MVHE.Enc}(sk_{id}, \Delta, l_i, m_i)$ *for all* $i \in [t]$. *We say that* **MVHE** *satisfies succinctness if there exists a fixed polynomial* poly *such that* $|c| = \mathsf{poly}(\lambda, n, \log t)$ *where* $n = |\{id \in \mathcal{P}\}|$ *and* $c \leftarrow \mathbf{MVHE.Eval}(f, \{(c_i, \mathbf{EKS}_{id_i})\}_{i\in[t]})$. *(We note that, similarly to the succinctness of MHEA, the size of all ciphertexts depend only logarithmically on the size of a dataset but linearly the number of keys involved in the evaluation.)*

Next, we provide the security definitions of MVHE: *privacy* and *unforgeability*.

Definition 10 (Privacy for MVHE)**.** *Let* **MVHE** *be an MVHE scheme. We define the privacy game between a challenger* $\mathcal{C}$ *and an adversary* $\mathcal{A}$.

1. $\mathcal{C}$ *chooses a challenge bit* $b \leftarrow \{0,1\}$, *generates a public parameter* $pp \leftarrow \mathbf{MVHE.Setup}(1^\lambda)$, *gives* pp *to* $\mathcal{A}$, *and prepares lists* $L_{key} := \emptyset$, $L_{ch} := \emptyset$, *and* $L_{corr} := \emptyset$.

2. $\mathcal{A}$ *can make polynomially many challenge queries and corruption queries as follows:*

   Challenge Queries. *When* $\mathcal{C}$ *receives a challenge query* $(\Delta, l = (id, \tau), m_0, m_1)$ *(*$|m_0| = |m_1|$*), it answers as follows:*

   (a) *If* (Δ, l, m_0, m_1) *is the first query for the dataset* Δ, $\mathcal{C}$ *initializes an empty list* $L_\Delta = \emptyset$ *and proceeds as follows.*

   (b) *If* (Δ, l, m) *is the first query with identity* id *(that is,* $(id, \cdot, \cdot) \notin L_{key}$*),* $\mathcal{C}$ *generates* $(ek_{id}, sk_{id}) \leftarrow \mathbf{MVHE.KG}(pp, id)$ *(that are implicitly assigned to the identity* id*), gives* ek_{id} *to* $\mathcal{A}$, *appends* (id, sk_{id}) *to* L_{key}, *and proceeds as follows.*

   (c) *If* (Δ, l, m_0, m_1) *is such that* $(l, \cdot) \notin L_\Delta$, $\mathcal{C}$ *checks whether* $id \in L_{corr}$ *holds. If this is the case, then* $\mathcal{C}$ *returns* $\perp$ *to* $\mathcal{A}$. *Otherwise,* $\mathcal{C}$ *computes* $c_l \leftarrow \mathbf{MVHE.Enc}(sk_{id}, \Delta, l, m_b)$ *(note that* $\mathcal{C}$ *has already generated keys for the identity* id*), gives* c_l *to* $\mathcal{A}$, *and updates the lists* $L_\Delta \leftarrow L_\Delta \cup \{(l, m)\}$ *and* $L_{ch} \leftarrow L_{ch} \cup \{id\}$.

(d) *If* (Δ, l, m_0, m_1) *is such that* $(l, \cdot) \in L_\Delta$ *(which means that* $\mathcal{A}$ *had already made a query* (Δ, l, m') *for some plaintext* m'*),* $\mathcal{C}$ *ignores the query.*

Corruption Queries. *When* $\mathcal{C}$ *receives a corruption query* id*, it answers as follows:*

(a) $\mathcal{C}$ *checks whether* $id \in L_{ch}$ *holds. If this is the case, then* $\mathcal{C}$ *returns* $\perp$ *to* $\mathcal{A}$*.*

(b) $\mathcal{C}$ *generates keys* $(ek_{id}, sk_{id}) \leftarrow \mathbf{MVHE.KG}(pp, id)$*, gives* sk_{id} *to* $\mathcal{A}$*, and appends* id *to* L_{corr}*.*

3. $\mathcal{A}$ *outputs a bit* $b' \in \{0, 1\}$*.*

In this game, we define the advantage of the adversary $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MVHE},\mathcal{A}}(\lambda) := 2 \cdot \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

We say that **MVHE** *satisfies privacy if for any PPT adversary* $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MVHE},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$ *holds.*

Definition 11 (Unforgeability for MVHE)**.** *Let* **MVHE** *be an MVHE scheme. We define the unforgeability game between a challenger* $\mathcal{C}$ *and an adversary* $\mathcal{A}$ *as follows.*

1. $\mathcal{C}$ *generates a public parameter* $pp \leftarrow \mathbf{MVHE.Setup}(1^\lambda)$*, gives* pp *to* $\mathcal{A}$*, and prepares lists* $L_{key} := \emptyset$ *and* $L_{corr} := \emptyset$*.*

2. $\mathcal{A}$ *can adaptively make polynomially many authentication queries, verification queries, and corruption queries as follows:*

Authentication Queries. *When* $\mathcal{C}$ *receives an authentication query* (Δ, l, m)*, where* Δ *is a dataset identifier,* $l = (id, \tau)$ *is a label in* $\mathcal{ID} \times \mathcal{T}$*, and* $m \in \mathcal{M}$*, it answers as follows:*

- *If* (Δ, l, m) *is the first query for the dataset* Δ*,* $\mathcal{C}$ *initializes an empty list* $L_\Delta := \emptyset$ *and proceeds as follows.*
- *If* $(id, \cdot) \notin L_{key}$ *holds, then* $\mathcal{C}$ *generates keys* $(ek_{id}, sk_{id}) \leftarrow \mathbf{MVHE.KG}(pp, id)$ *(that are implicity assigned to identity* id*), gives* ek_{id} *to* $\mathcal{A}$ *and updates the list* $L_{key} \leftarrow L_{key} \cup \{(id, sk_{id})\}$*.*
- *If* (Δ, l, m) *is such that* $(l, m) \notin L_\Delta$*,* $\mathcal{C}$ *computes* $c_l \leftarrow \mathbf{MVHE.Enc}(sk_{id}, \Delta, l, m)$ *(note that* $\mathcal{C}$ *has already generated keys for the identity* id*), returns* c_l *to* $\mathcal{A}$*, and updates the list* $L_\Delta \leftarrow L_\Delta \cup \{(l, m)\}$*.*

- *If* (Δ, l, m) *is such that* $(l, \cdot) \in L_\Delta$ *(which means that the adversary had already made a query* (Δ, l, m') *for some message* m'*),* $\mathcal{C}$ *ignores the query.*

Verification Queries. *When* $\mathcal{C}$ *receives a verification query* $(\mathcal{P}, \Delta, m, \sigma)$*, it answers as follows:*

(a) If $(id, \cdot) \notin L_{key}$ *holds for some* $id \in \mathcal{P}$*, then* $\mathcal{C}$ *generates* $(ek_{id}, sk_{id}) \leftarrow \mathbf{MVHE.KG}(pp, id)$ *and updates the list* $L_{key} \leftarrow L_{key} \cup \{(id, sk_{id})\}$*.*

(b) $\mathcal{C}$ *gives* 1 *to* $\mathcal{A}$ *if* $\mathbf{MVHE.Dec}(\mathcal{P}, \Delta, \{sk_{id}\}_{id\in\mathcal{P}}, m, \sigma) \neq \perp$ *holds. Otherwise,* $\mathcal{C}$ *gives* 0 *to* $\mathcal{A}$*.*

Corruption Queries. *When* $\mathcal{C}$ *receives a corruption query* id*, it answers as follows:*

(a) If $(id, \cdot) \notin L_{key}$ *holds, then* $\mathcal{C}$ *generates* $(ek_{id}, sk_{id}) \leftarrow \mathbf{MVHE.KG}(pp, id)$ *and updates the list* $L_{key} := L_{key} \cup \{(id, sk_{id})\}$*.*

(b) $\mathcal{C}$ *gives* sk_{id} *to* $\mathcal{A}$ *and appends* id *to* L_{corr}*.*

3. $\mathcal{A}$ *outputs a forgery* $(\mathcal{P}^* = (f^*, l_1^*, \cdots, l_t^*), \Delta^*, m^*, \sigma^*)$*.*

In the above game, we say that an event **Forge** *occurs if* $\mathbf{MVHE.Dec}(\mathcal{P}^*, \Delta^*, \{vk_{id}\}_{id\in\mathcal{P}^*}, m^*, \sigma^*) \neq \perp$ *for all* $id \in \mathcal{P}^*$*,* $id \notin L_{corr}$*, and either one of the following condition holds:*

Type 1: L_{Δ^*} *has not been initialized during the game.*

Type 2: *For all* $i \in [t]$*,* $\exists (l_i^*, m_i) \in L_{\Delta^*}$*, but* $m^* \neq f^*(m_1, \cdots, m_t)$*.*

Type 3: *There exists a label* l_i^* *such that* $(l_i^*, \cdot) \notin L_{\Delta^*}$ *for some* $i \in [t]$*.*

We say that an MVHE scheme **MVHE** *satisfies unforgeability if for any PPT adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathsf{unf}}_{\mathbf{MVHE},\mathcal{A}}(\lambda) := \Pr[\mathbf{Forge}] = \mathsf{negl}(\lambda).$$

3.3.2 Construction

In this section, we provide our construction of MVHE based on MHE and MHEA. Let $\mathbf{MHE} = (\mathbf{MHE.Setup}, \mathbf{MHE.KG}, \mathbf{MHE.Enc}, \mathbf{MHE.Dec}, \mathbf{MHE.Eval})$ be an MHE scheme and $\mathbf{MHEA} = (\mathbf{MHEA.Setup}, \mathbf{MHEA.KG}, \mathbf{MHEA.Auth}, \mathbf{MHEA.Eval}, \mathbf{MHEA.Ver})$ an MHEA scheme. We assume that **MHE.Setup** and **MHEA.Setup** support the same plaintext space $\mathcal{M}$, function family $\mathcal{F} : \mathcal{M}^t \to \mathcal{M}$, and identity space $\mathcal{ID}$. Using **MHE**

and **MHEA**, our MVHE scheme $\mathbf{MVHE} = (\mathbf{MVHE.Setup}, \mathbf{MVHE.KG}, \mathbf{MVHE.Enc}, \mathbf{MVHE.Dec}, \mathbf{MVHE.Eval})$ whose plaintext space $\mathcal{M}$ and identity space $\mathcal{ID}$ is described as follows.

$\mathbf{MVHE.Setup}(1^\lambda)$**:**

- Generate $pp^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Setup}(1^\lambda)$.
- Generate $pp^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Setup}(1^\lambda)$.
- Return $pp := (pp^{\mathbf{MHE}}, pp^{\mathbf{MHEA}})$.

$\mathbf{MVHE.KG}(pp, id)$**:**

- Parse $pp := (pp^{\mathbf{MHE}}, pp^{\mathbf{MHEA}})$.
- Generate $(ek_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHE}}) \leftarrow \mathbf{MHE.KG}(pp^{\mathbf{MHE}}, id)$.
- Generate $(ek_{id}^{\mathbf{MHEA}}, sk_{id}^{\mathbf{MHEA}}) \leftarrow \mathbf{MHEA.KG}(pp^{\mathbf{MHEA}}, id)$.
- Return $ek_{id} := (ek_{id}^{\mathbf{MHE}}, ek_{id}^{\mathbf{MHEA}})$ and $sk_{id} := (sk_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHEA}})$.

$\mathbf{MVHE.Enc}(sk_{id}, \Delta, l, m)$**:**

- Parse $sk_{id} := (sk_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHEA}})$ and $l := (id, \tau)$.
- Compute $c^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Enc}(sk_{id}^{\mathbf{MHE}}, m)$.
- Compute $\sigma^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Auth}(sk_{id}^{\mathbf{MHEA}}, \Delta, l, m)$.
- Return $c := (c^{\mathbf{MHE}}, \sigma^{\mathbf{MHEA}})$.

$\mathbf{MVHE.Dec}(\mathcal{P}, \Delta, (sk_{id})_{id\in\mathcal{P}}, c)$**:**

- Parse $c := (c^{\mathbf{MHE}}, \sigma^{\mathbf{MHEA}})$.
- For all $id \in \mathcal{P}$, parse $sk_{id} := (sk_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHEA}})$.
- Compute $m \leftarrow \mathbf{MHE.Dec}(\mathcal{P}, (sk_{id}^{\mathbf{MHE}})_{id\in\mathcal{P}}, c^{\mathbf{MHE}})$.
- If $\mathbf{MHEA.Ver}(\mathcal{P}, \Delta, (sk_{id}^{\mathbf{MHEA}})_{id\in\mathcal{P}}, m, \sigma^{\mathbf{MHEA}}) = 1$ then return m. Otherwise, return $\perp$.

$\mathbf{MVHE.Eval}(f, \{(c_i, \mathbf{EKS}_i)\}_{i\in[t]})$**:**

- For all $i \in [t]$, parse $c_i := (c_i^{\mathbf{MHE}}, \sigma_i^{\mathbf{MHEA}})$.

- For all $i \in [t]$, parse $\mathbf{EKS}_i := (ek_{id})_{id \in \mathcal{P}_i} = (ek_{id}^{\mathbf{MHE}}, ek_{id}^{\mathbf{MHEA}})_{id \in \mathcal{P}_i}$.
- For all $i \in [t]$, set $\mathbf{EKS}_i^{\mathbf{MHE}} := (ek_{id}^{\mathbf{MHE}})_{id \in \mathcal{P}_i}$ and $\mathbf{EKS}_i^{\mathbf{MHEA}} := (ek_{id}^{\mathbf{MHEA}})_{id \in \mathcal{P}_i}$.
- Compute $c_f^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Eval}(f, \{(c_i^{\mathbf{MHE}}, \mathbf{EKS}_i^{\mathbf{MHE}})\}_{i \in [t]})$.
- Compute $\sigma_f^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Eval}(f, \{(\sigma_i^{\mathbf{MHEA}}, \mathbf{EKS}_i^{\mathbf{MHEA}})\}_{i \in [t]})$.
- Return $c_f := (c_f^{\mathbf{MHE}}, \sigma_f^{\mathbf{MHEA}})$.

We can easily see that the ordinary correctness, evaluation correctness, and succinctness of **MVHE** are followed due to the ordinary correctness, evaluation correctness, and succinctness of **MHE** and **MHEA**.

3.3.3 Security Proof

In this section, we show that our MVHE scheme **MVHE** satisfies privacy (Theorem 10) and unforgeability (Theorem 11).

Theorem 10. *If* **MHE** *satisfies IND-CPA security and* **MHEA** *satisfies privacy, then* **MVHE** *satisfies privacy.*

Proof of Theorem 10. Let $Q_{key} := Q_{key}(\lambda)$ be an arbitrary polynomial that denotes the number of key pairs generated in the privacy game. Let $\mathcal{A}$ be any PPT adversary that attacks the privacy of **MVHE**. We proceed the proof via a sequence of games by introducing the following games: $\mathbf{Game}_i$ for $i \in [0, 2]$.

$\mathbf{Game}_0$: This is the original privacy game for **MVHE** conditioned on $b = 0$. The detailed description is as follows:

1. The challenger $\mathcal{C}$ proceeds as follows:
   (a) $\mathcal{C}$ generates $pp^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Setup}(1^\lambda)$ and $pp^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Setup}(1^\lambda)$ and sets $pp := (pp^{\mathbf{MHE}}, pp^{\mathbf{MHEA}})$.
   (b) $\mathcal{C}$ gives pp to $\mathcal{A}$ and prepares lists $L_{key} := \emptyset$, $L_{ch} := \emptyset$, and $L_{corr} := \emptyset$.
2. When $\mathcal{A}$ makes challenge queries and corruption queries, $\mathcal{C}$ answers as follows:

   Challenge Queries. When $\mathcal{C}$ receives a challenge query $(\Delta, l = (id, \tau), m_0, m_1)$, it proceeds as follows:

   (a) If (Δ, l, m_0, m_1) is the first query for the dataset Δ, $\mathcal{C}$ initializes an empty list $L_\Delta := \emptyset$.

(b) If $(id, \cdot) \notin L_{key}$ holds, $\mathcal{C}$ proceeds as follows:

i. $\mathcal{C}$ generates $(ek_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHE}}) \leftarrow \mathbf{MHE.KG}(pp^{\mathbf{MHE}}, id)$.

ii. $\mathcal{C}$ generates $(ek_{id}^{\mathbf{MHEA}}, sk_{id}^{\mathbf{MHEA}}) \leftarrow \mathbf{MHEA.KG}(pp^{\mathbf{MHEA}}, id)$.

iii. $\mathcal{C}$ sets $ek_{id} := (ek_{id}^{\mathbf{MHE}}, ek_{id}^{\mathbf{MHEA}})$ and $sk_{id} := (sk_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHEA}})$, gives ek_{id} to $\mathcal{A}$, and appends (id, sk_{id}) to L_{key}.

(c) If (Δ, l, m_0, m_1) is such that $(l, \cdot) \notin L_{\Delta}$, $\mathcal{C}$ proceeds as follows:

i. $\mathcal{C}$ checks whether $id \in L_{corr}$ holds. If this is the case, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

ii. $\mathcal{C}$ computes $c^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Enc}(sk_{id}^{\mathbf{MHE}}, m_0)$ (using $sk_{id}^{\mathbf{MHE}}$ included in L_{key}).

iii. $\mathcal{C}$ computes $\sigma^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Auth}(sk_{id}^{\mathbf{MHEA}}, \Delta, l, m_0)$ (using $sk_{id}^{\mathbf{MHEA}}$ included in L_{key}).

iv. $\mathcal{C}$ sets $c := (c^{\mathbf{MHE}}, \sigma^{\mathbf{MHEA}})$, gives c to $\mathcal{A}$, and appends (l, m) to L_{Δ} and id to L_{ch}.

(d) If (Δ, l, m_0, m_1) is such that $(l, \cdot) \in L_{\Delta}$, $\mathcal{C}$ ignores the query.

Corruption Queries. When $\mathcal{C}$ receives a corruption query id, it answers as follows:

(a) $\mathcal{C}$ checks whether $id \in L_{ch}$ holds. If this is the case, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.

(b) $\mathcal{C}$ generates $(ek_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHE}}) \leftarrow \mathbf{MHE.KG}(pp^{\mathbf{MHE}}, id)$.

(c) $\mathcal{C}$ generates $(ek_{id}^{\mathbf{MHEA}}, sk_{id}^{\mathbf{MHEA}}) \leftarrow \mathbf{MHEA.KG}(pp^{\mathbf{MHEA}}, id)$.

(d) $\mathcal{C}$ sets $sk_{id} := (sk_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHEA}})$, gives sk_{id} to $\mathcal{A}$, and appends id to L_{corr}.

3. $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$.

$\mathbf{Game}_1$: This game is identical to $\mathbf{Game}_0$ except that $\mathcal{C}$ computes $c^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Enc}(sk_{id}^{\mathbf{MHE}}, m_1)$ instead of $c^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Enc}(sk_{id}^{\mathbf{MHE}}, m_0)$ when responding to the challenge queries $(\Delta, l = (id, \tau), m_0, m_1)$.

$\mathbf{Game}_2$: This game is identical to $\mathbf{Game}_1$ except that $\mathcal{C}$ computes $\sigma^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Auth}(sk_{id}^{\mathbf{MHEA}}, \Delta, l, m_1)$ instead of $\sigma^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Auth}(sk_{id}^{\mathbf{MHEA}}, \Delta, l, m_0)$ when responding to the challenge queries $(\Delta, l = (id, \tau), m_0, m_1)$. Note that this game is the same as the original privacy game for **MVHE** conditioned on $b = 1$.

For $i \in [0,2]$, let $\mathbf{Win}_i$ be the event that $\mathcal{A}$ outputs $b' = 0$ in $\mathbf{Game}_i$. By using triangle inequality, we have

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MVHE},\mathcal{A}}(\lambda) &= 2 \cdot \left| \Pr[b = b'] - \frac{1}{2} \right| \\
&= |\Pr[b' = 0 | b = 0] - \Pr[b' = 0 | b = 1]| \\
&= |\Pr[\mathbf{Win}_0] - \Pr[\mathbf{Win}_2]| \\
&\leq \sum_{i=0}^{1} |\Pr[\mathbf{Win}_i] - \Pr[\mathbf{Win}_{i+1}]| \,.
\end{aligned}$$

It remains to show how each $|\Pr[\mathbf{Win}_i] - \Pr[\mathbf{Win}_{i+1}]|$ is upper-bounded. To this end, we will show the following lemmata.

- There exists an adversary $\mathcal{B}_1$ against the IND-CPA security of **MHE** such that $|\Pr[\mathbf{Win}_0] - \Pr[\mathbf{Win}_1]| = \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathbf{MHE},\mathcal{B}_1}(\lambda)$ (Lemma 1).
- There exists an adversary $\mathcal{B}_2$ against the privacy of **MHEA** such that $|\Pr[\mathbf{Win}_1] - \Pr[\mathbf{Win}_2]| = \mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MHEA},\mathcal{B}_2}(\lambda)$ (Lemma 2).

Lemma 1. *There exists an adversary $\mathcal{B}_1$ against the IND-CPA security of* **MHE** *such that* $|\Pr[\mathbf{Win}_0] - \Pr[\mathbf{Win}_1]| = \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathbf{MHE},\mathcal{B}_1}(\lambda)$.

Proof of Lemma 1. We construct an adversary $\mathcal{B}_1$ that attacks the IND-CPA security of **MHE** so that $|\Pr[\mathbf{Win}_0] - \Pr[\mathbf{Win}_1]| = \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathbf{MHE},\mathcal{B}_1}(\lambda)$, using the adversary $\mathcal{A}$ as follows.

1. Upon receiving a public parameter $pp^{\mathbf{MHE}}$ and a set of evaluation keys $(ek_i^{\mathbf{MHE}})_{i \in [Q_{key}]}$ from the challenger, $\mathcal{B}_1$ proceeds as follows:
   (a) $\mathcal{B}_1$ generates $pp^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Setup}(1^\lambda)$ and sets $pp := (pp^{\mathbf{MHE}}, pp^{\mathbf{MHEA}})$.
   (b) $\mathcal{B}_1$ gives pp to $\mathcal{A}$ and initializes a list $L_{key} := \emptyset$ and a counter $\mathsf{cnt} := 1$.
2. When $\mathcal{A}$ makes challenge queries and corruption queries, $\mathcal{B}_1$ answers as follows:

   Challenge Queries. When $\mathcal{B}_1$ receives a challenge query $(\Delta, l = (id, \tau), m_0, m_1)$, it proceeds as follows:

   (a) If (Δ, l, m_0, m_1) is the first query for the dataset Δ, $\mathcal{B}_1$ initializes an empty list $L_\Delta := \emptyset$.
   (b) If $(id, \cdot) \notin L_{key}$ holds, $\mathcal{B}_1$ proceeds as follows:

i. $\mathcal{B}_1$ sets $ek_{id}^{\mathbf{MHE}} := ek_{\mathsf{cnt}}^{\mathbf{MHE}}$ ($ek_{\mathsf{cnt}}^{\mathbf{MHE}}$ is associated with an identity id) and updates $\mathsf{cnt} := \mathsf{cnt} + 1$.

ii. $\mathcal{B}_1$ generates $(ek_{id}^{\mathbf{MHEA}}, sk_{id}^{\mathbf{MHEA}}) \leftarrow \mathbf{MHEA.KG}(pp^{\mathbf{MHEA}})$.

iii. $\mathcal{B}_1$ sets $ek_{id} := (ek_{id}^{\mathbf{MHE}}, ek_{id}^{\mathbf{MHEA}})$, gives ek_{id} to $\mathcal{A}$, and appends $(id, ek_{id}, sk_{id}^{\mathbf{MHEA}})$ to L_{key}.

(c) If (Δ, l, m_0, m_1) is such that $(l = (id, \tau), \cdot) \notin L_\Delta$, $\mathcal{B}_1$ proceeds as follows:

i. $\mathcal{B}_1$ checks whether $id \in L_{corr}$ holds. If this is the case, then $\mathcal{B}_1$ returns $\perp$ to $\mathcal{A}$.

ii. $\mathcal{B}_1$ makes a challenge query (i, m_0, m_1) to its challenger, where i is an index assigned to id in the previous step, and gets a ciphertext $c^{\mathbf{MHE}}$.

iii. $\mathcal{B}_1$ computes $\sigma^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Auth}(sk_{id}^{\mathbf{MHEA}}, \Delta, l, m_0)$.

iv. $\mathcal{B}_1$ sets $c := (c^{\mathbf{MHE}}, \sigma^{\mathbf{MHEA}})$, gives c to $\mathcal{A}$, and appends (l, m) to L_Δ and id to L_{ch}.

(d) If (Δ, l, m_0, m_1) is such that $(l, \cdot) \in L_\Delta$, $\mathcal{C}$ ignores the query.

Corruption Queries. When $\mathcal{B}_1$ receives a corruption query id, it proceeds as follows:

(a) $\mathcal{B}_1$ checks whether $id \in L_{ch}$ holds. If this is the case, then $\mathcal{B}_1$ returns $\perp$ to $\mathcal{A}$.

(b) $\mathcal{B}_1$ makes a corruption query cnt to its challenger, gets a secret key $sk_{\mathsf{cnt}}^{\mathbf{MHE}}$, sets $sk_{id}^{\mathbf{MHE}} := sk_{\mathsf{cnt}}^{\mathbf{MHE}}$ ($ek_{\mathsf{cnt}}^{\mathbf{MHE}}$ is associated with an identity id), and updates $\mathsf{cnt} := \mathsf{cnt} + 1$.

(c) $\mathcal{B}_1$ generates $(sk_{id}^{\mathbf{MHEA}}, sk_{id}^{\mathbf{MHEA}}) \leftarrow \mathbf{MHEA.KG}(pp^{\mathbf{MHEA}})$.

(d) $\mathcal{B}_1$ sets $sk_{id} := (sk_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHEA}})$, gives sk_{id} to $\mathcal{A}$, and appends id to L_{corr}.

3. When $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$ and terminates, $\mathcal{B}_1$ outputs $\beta' := 0$ to the challenger and terminates if $b' = 0$ holds. Otherwise, $\mathcal{B}_1$ outputs $\beta' := 1$ to the challenger and terminates.

In the following, we let β be the challenge bit for $\mathcal{B}_1$ in the IND-CPA game. We can see that $\mathcal{B}_1$ perfectly simulates $\mathbf{Game}_0$ for $\mathcal{A}$ if it receives the challenge ciphertext $c^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Enc}(sk_{id}^{\mathbf{MHE}}, m_0)$ from its challenger. This ensures that the probability that $\mathcal{B}_1$ outputs 0 when $\beta = 0$ is exactly the same as the probability that $\mathbf{Win}_0$ happens in $\mathbf{Game}_0$. That is, $\Pr[\beta' = 0 | \beta = 0] = \Pr[\mathbf{Win}_0]$ holds.

On the other hand, we can see that $\mathcal{B}_1$ perfectly simulates $\mathbf{Game}_1$ for $\mathcal{A}$ if it receives the challenge ciphertext $c^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Enc}(sk_{id}^{\mathbf{MHE}}, m_1)$ from its challenger. This ensures that

the probability that $\mathcal{B}_1$ outputs 0 when $\beta = 1$ is exactly the same as the probability that $\mathbf{Win}_1$ happens in $\mathbf{Game}_1$. That is, $\Pr[\beta' = 0|\beta = 1] = \Pr[\mathbf{Win}_1]$ holds. Therefore, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathbf{MHE},\mathcal{B}_1}(\lambda) &= |\Pr[\beta' = 0|\beta = 0] - \Pr[\beta' = 0|\beta = 1]| \\ &= |\Pr[\mathbf{Win}_0] - \Pr[\mathbf{Win}_1]|.\end{aligned}$$

□ (**Lemma 1**)

Lemma 2. *There exists an adversary* $\mathcal{B}_2$ *against the privacy of* **MHEA** *such that* $|\Pr[\mathbf{Win}_1] - \Pr[\mathbf{Win}_2]| = \mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MHEA},\mathcal{B}_2}(\lambda)$.

Proof of Lemma 2. We construct an adversary $\mathcal{B}_2$ that attacks the privacy of **MHEA** so that $|\Pr[\mathbf{Win}_1] - \Pr[\mathbf{Win}_2]| = \mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MHEA},\mathcal{B}_2}(\lambda)$, using the adversary $\mathcal{A}$ as follows.

1. Upon receiving a public parameter $pp^{\mathbf{MHEA}}$, $\mathcal{B}_2$ generates $pp^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Setup}(1^\lambda)$, gives $pp := (pp^{\mathbf{MHE}}, pp^{\mathbf{MHEA}})$ to $\mathcal{A}$, and initializes a list $L_{key} := \emptyset$.
2. When $\mathcal{A}$ makes challenge queries and corruption queries, $\mathcal{B}_2$ answers as follows:

   Challenge Queries. When $\mathcal{B}_2$ receives a challenge query $(\Delta, l = (id, \tau), m_0, m_1)$, it proceeds as follows:

   (a) If (Δ, l, m_0, m_1) is the first query for the dataset Δ, $\mathcal{B}_2$ initializes an empty list $L_\Delta := \emptyset$.

   (b) If (Δ, l, m_0, m_1) is such that $(l = (id, \tau), \cdot) \notin L_\Delta$, $\mathcal{B}_2$ proceeds as follows:

      i. $\mathcal{B}_2$ makes a challenge query $(\Delta, l = (id, \tau), m_0, m_1)$ to its challenger and gets an authenticator $\sigma^{\mathbf{MHEA}}$. If $\sigma^{\mathbf{MHEA}} = \perp$ holds, then $\mathcal{B}_2$ returns $\perp$ to $\mathcal{A}$.

      ii. If (Δ, l, m) is the first query with identity id (that is, $(id, \cdot) \notin L_{key}$), $\mathcal{B}_2$ gets $ek_{id}^{\mathbf{MHEA}}$ from its challenger, computes $(ek_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHE}}) \leftarrow \mathbf{MHE.KG}(pp^{\mathbf{MHE}}, id)$, gives $ek_{id} := (ek_{id}^{\mathbf{MHE}}, ek_{id}^{\mathbf{MHEA}})$ to $\mathcal{A}$, and appends $(id, sk_{id}^{\mathbf{MHE}})$ to L_{key}.

      iii. $\mathcal{B}_2$ computes $c^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Enc}(sk_{id}^{\mathbf{MHE}}, m_1)$ (using a secret key $sk_{id}^{\mathbf{MHE}} \in L_{key}$).

      iv. $\mathcal{B}_2$ sets $c := (c^{\mathbf{MHE}}, \sigma^{\mathbf{MHEA}})$, gives c to $\mathcal{A}$, and appends (l, m) to L_Δ.

   (c) If (Δ, l, m_0, m_1) is such that $(l, \cdot) \in L_\Delta$, $\mathcal{C}$ ignores the query.

Corruption Queries. When $\mathcal{B}_2$ receives a corruption query id, it proceeds as follows:

(a) $\mathcal{B}_2$ makes a corruption query id to its challenger and gets a secret key $sk_{id}^{\mathbf{MHEA}}$. If $sk_{id}^{\mathbf{MHEA}} = \bot$ holds, then it also gives $\bot$ to $\mathcal{A}$.

(b) $\mathcal{B}_2$ generates $(ek_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHE}}) \leftarrow \mathbf{MHE.KG}(pp^{\mathbf{MHE}}, id)$.

(c) $\mathcal{B}_2$ sets $sk_{id} := (sk_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHEA}})$ and gives sk_{id} to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ and terminates, $\mathcal{B}_2$ outputs $\beta' := 0$ to the challenger and terminates if $b' = 0$ holds. Otherwise, $\mathcal{B}_2$ outputs $\beta' := 1$ to the challenger and terminates.

In the following, we let β be the challenge bit for $\mathcal{B}_2$ in the privacy game (of **MHEA**). We can see that $\mathcal{B}_2$ perfectly simulates $\mathbf{Game}_1$ for $\mathcal{A}$ if it receives the challenge authenticator $\sigma^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Auth}(sk_{id}^{\mathbf{MHEA}}, \Delta, l, m_0)$ from its challenger. This ensures that the probability that $\mathcal{B}_2$ outputs 0 when $\beta = 0$ is exactly the same as the probability that $\mathbf{Win}_1$ happens in $\mathbf{Game}_1$. That is, $\Pr[\beta' = 0 | \beta = 0] = \Pr[\mathbf{Win}_1]$ holds.

On the other hand, we can see that $\mathcal{B}_2$ perfectly simulates $\mathbf{Game}_2$ for $\mathcal{A}$ if it receives the challenge authenticator $\sigma^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Auth}(sk_{id}^{\mathbf{MHEA}}, \Delta, l, m_1)$ from its challenger. This ensures that the probability that $\mathcal{B}_2$ outputs 0 when $\beta = 1$ is exactly the same as the probability that $\mathbf{Win}_2$ happens in $\mathbf{Game}_2$. That is, $\Pr[\beta' = 0 | \beta = 1] = \Pr[\mathbf{Win}_2]$ holds. Therefore, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MHEA},\mathcal{B}_2}(\lambda) &= |\Pr[\beta' = 0 | \beta = 0] - \Pr[\beta' = 0 | \beta = 1]| \\ &= |\Pr[\mathbf{Win}_1] - \Pr[\mathbf{Win}_2]|.\end{aligned}$$

□ (**Lemma 2**)

Putting everything together, we obtain

$$\mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MVHE},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathbf{MHE},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MHEA},\mathcal{B}_2}(\lambda).$$

Since **MHE** satisfies IND-CPA security and **MHEA** satisfies privacy, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{priv}}_{\mathbf{MVHE},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$ holds. Therefore, **MVHE** satisfies privacy.

□ (**Theorem 10**)

Theorem 11. *If* **MHEA** *satisfies unforgeability, then* **MVHE** *satisfies unforgeability.*

Proof of Theorem 11. Let $\mathcal{A}$ be any PPT adversary that attacks the unforgeability of **MVHE**. The original unforgeability game is described as follows.

1. The challenger $\mathcal{C}$ proceeds as follows:

   (a) $\mathcal{C}$ generates $pp^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Setup}(1^\lambda)$ and $pp^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Setup}(1^\lambda)$ and sets $pp := (pp^{\mathbf{MHE}}, pp^{\mathbf{MHEA}})$.

   (b) $\mathcal{C}$ gives pp to $\mathcal{A}$ and prepares lists $L_{key} := \emptyset$, $L_{ch} := \emptyset$, and $L_{corr} := \emptyset$.

2. When $\mathcal{A}$ makes authentication queries, verification queries, and corruption queries, $\mathcal{C}$ answers as follows:

   Authentication Queries. When $\mathcal{C}$ receives an authentication query $(\Delta, l = (id, \tau), m)$, it proceeds as follows:

   (a) If (Δ, l, m) is the first query for the dataset Δ, $\mathcal{C}$ initializes an empty list $L_\Delta := \emptyset$.

   (b) If $(id, \cdot, \cdot) \notin L_{key}$ holds, $\mathcal{C}$ proceeds as follows:

      i. $\mathcal{C}$ generates $(ek_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHE}}) \leftarrow \mathbf{MHE.KG}(pp^{\mathbf{MHE}}, id)$.

      ii. $\mathcal{C}$ generates $(ek_{id}^{\mathbf{MHEA}}, sk_{id}^{\mathbf{MHEA}}) \leftarrow \mathbf{MHEA.KG}(pp^{\mathbf{MHEA}}, id)$.

      iii. $\mathcal{C}$ sets $ek_{id} := (ek_{id}^{\mathbf{MHE}}, ek_{id}^{\mathbf{MHEA}})$ and $sk_{id} := (sk_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHEA}})$, gives ek_{id} to $\mathcal{A}$, and appends (id, sk_{id}) to L_{key}.

   (c) If (Δ, l, m) is such that $(l, m) \notin L_\Delta$, $\mathcal{C}$ proceeds as follows:

      i. $\mathcal{C}$ computes $c^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Enc}(sk_{id}^{\mathbf{MHE}}, m)$ (using $sk_{id}^{\mathbf{MHE}}$ included in L_{key}).

      ii. $\mathcal{C}$ computes $\sigma^{\mathbf{MHEA}} \leftarrow \mathbf{MHEA.Auth}(sk_{id}^{\mathbf{MHEA}}, \Delta, l, m)$ (using $sk_{id}^{\mathbf{MHEA}}$ included in L_{key}).

      iii. $\mathcal{C}$ sets $c := (c^{\mathbf{MHE}}, \sigma^{\mathbf{MHEA}})$, gives c to $\mathcal{A}$, and appends (l, m) to L_Δ.

   (d) If (Δ, l, m) is such that $(l, \cdot) \in L_\Delta$, $\mathcal{C}$ ignores the query.

   Verification Queries. When $\mathcal{C}$ receives a verification query $(\mathcal{P}, \Delta, m, c)$, it proceeds as follows:

   (a) $\mathcal{C}$ parses $c := (c^{\mathbf{MHE}}, \sigma^{\mathbf{MHEA}})$ and $\mathcal{P} := (f, id_1, \cdots, id_t)$.

   (b) If $(id, \cdot) \notin L_{key}$ holds for some $id \in \mathcal{P}$, then $\mathcal{C}$ proceeds as follows:

      i. $\mathcal{C}$ generates $(ek_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHE}}) \leftarrow \mathbf{MHE.KG}(pp^{\mathbf{MHE}}, id)$.

ii. $\mathcal{C}$ generates $(ek_{id}^{\mathbf{MHEA}}, sk_{id}^{\mathbf{MHEA}}) \leftarrow \mathbf{MHEA.KG}(pp^{\mathbf{MHEA}}, id)$.

iii. $\mathcal{C}$ sets $ek_{id} := (ek_{id}^{\mathbf{MHE}}, ek_{id}^{\mathbf{MHEA}})$ and $sk_{id} := (sk_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHEA}})$, gives ek_{id} to $\mathcal{A}$, and appends (id, sk_{id}) to L_{key}.

(c) $\mathcal{C}$ computes $m \leftarrow \mathbf{MHE.Dec}(\mathcal{P}, (sk_{id}^{\mathbf{MHE}})_{id\in\mathcal{P}}, c^{\mathbf{MHE}})$.

(d) $\mathcal{C}$ checks whether $\mathbf{MHEA.Ver}(\mathcal{P}, \Delta, (sk_{id}^{\mathbf{MHEA}})_{id\in\mathcal{P}}, m, \sigma^{\mathbf{MHEA}}) = 1$ holds. If this is the case, then $\mathcal{C}$ returns 1 to $\mathcal{A}$. Otherwise, $\mathcal{C}$ returns 0 to $\mathcal{A}$.

Corruption Queries. When $\mathcal{C}$ receives a corruption query id, it answers as follows:

(a) $\mathcal{C}$ checks whether $(id, \cdot) \notin L_{key}$ holds. If this is the case, then $\mathcal{C}$ proceeds as follows:

i. $\mathcal{C}$ generates $(ek_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHE}}) \leftarrow \mathbf{MHE.KG}(pp^{\mathbf{MHE}}, id)$.

ii. $\mathcal{C}$ generates $(ek_{id}^{\mathbf{MHEA}}, sk_{id}^{\mathbf{MHEA}}) \leftarrow \mathbf{MHEA.KG}(pp^{\mathbf{MHEA}}, id)$.

iii. $\mathcal{C}$ sets $ek_{id} := (ek_{id}^{\mathbf{MHE}}, ek_{id}^{\mathbf{MHEA}})$ and $sk_{id} := (sk_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHEA}})$, and appends (id, sk_{id}) to L_{key}.

(b) $\mathcal{C}$ gives sk_{id} to $\mathcal{A}$ and appends id to L_{corr}.

3. $\mathcal{A}$ outputs a forgery $(\mathcal{P}^* = (f^*, l_1^*, \cdots, l_t^*), \Delta^*, m^*, \sigma^*)$.

In the following, we show that there exists an adversary $\mathcal{B}$ against the unforgeability of **MHEA** such that $\mathsf{Adv}^{\mathsf{unf}}_{\mathbf{MVHE},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{unf}}_{\mathbf{MHEA},\mathcal{B}}(\lambda)$. To this end, we construct an adversary $\mathcal{B}$ that attacks the unforgeability of **MHEA** using the adversary $\mathcal{A}$ as follows.

1. Upon receiving a public parameter $pp^{\mathbf{MHEA}}$, $\mathcal{B}$ generates $pp^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Setup}(1^\lambda)$, gives $pp := (pp^{\mathbf{MHE}}, pp^{\mathbf{MHEA}})$ to $\mathcal{A}$, and initializes a list $L_{key} := \emptyset$.

2. When $\mathcal{A}$ makes authentication queries, verification queries, and corruption queries, $\mathcal{B}$ answers as follows:

Authentication Queries. When $\mathcal{B}$ receives an authentication query $(\Delta, l = (id, \tau), m)$, it proceeds as follows:

(a) If (Δ, l, m) is the first query for the dataset Δ, $\mathcal{B}$ initializes an empty list $L_\Delta := \emptyset$.

(b) If (Δ, l, m) is such that $(l = (id, \tau), m) \notin L_\Delta$, $\mathcal{B}$ proceeds as follows:

i. $\mathcal{B}$ makes an authentication query $(\Delta, l = (id, \tau), m)$ to its challenger and gets an authenticator $\sigma^{\mathbf{MHEA}}$.

ii. If $(id, \cdot) \notin L_{key}$) holds, $\mathcal{B}$ gets $ek_{id}^{\mathbf{MHEA}}$ from its challenger, generates $(ek_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHE}}) \leftarrow \mathbf{MHE.KG}(pp^{\mathbf{MHE}}, id)$, gives $ek_{id} := (ek_{id}^{\mathbf{MHE}}, ek_{id}^{\mathbf{MHEA}})$ to $\mathcal{A}$, and appends $(id, sk_{id}^{\mathbf{MHE}})$ to L_{key}.

iii. $\mathcal{B}$ computes $c^{\mathbf{MHE}} \leftarrow \mathbf{MHE.Enc}(sk_{id}^{\mathbf{MHE}}, m)$ (using a secret key $sk_{id}^{\mathbf{MHE}} \in L_{key}$).

iv. $\mathcal{B}$ sets $c := (c^{\mathbf{MHE}}, \sigma^{\mathbf{MHEA}})$, gives c to $\mathcal{A}$, and appends (l, m) to L_{Δ}.

(c) If (Δ, l, m) is such that $(l, \cdot) \in L_{\Delta}$, $\mathcal{C}$ ignores the query.

Verification Queries. When $\mathcal{B}$ receives a verification query $(\mathcal{P}, \Delta, m, c)$, it proceeds as follows:

(a) $\mathcal{B}$ parses $c := (c^{\mathbf{MHE}}, \sigma^{\mathbf{MHEA}})$ and $\mathcal{P} := (f, id_1, \cdots, id_t)$.

(b) If $(id, \cdot) \notin L_{key}$ holds for some $id \in \mathcal{P}$, then $\mathcal{B}$ proceeds as follows:

i. $\mathcal{B}$ generates $(ek_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHE}}) \leftarrow \mathbf{MHE.KG}(pp^{\mathbf{MHE}}, id)$.

ii. $\mathcal{B}$ sets $ek_{id} := (ek_{id}^{\mathbf{MHE}}, ek_{id}^{\mathbf{MHEA}})$, gives ek_{id} to $\mathcal{A}$, and appends $(id, sk_{id}^{\mathbf{MHE}})$ to L_{key}.

(c) $\mathcal{B}$ computes $m \leftarrow \mathbf{MHE.Dec}(\mathcal{P}, (sk_{id}^{\mathbf{MHE}})_{id \in \mathcal{P}}, c^{\mathbf{MHE}})$.

(d) $\mathcal{B}$ makes a verification query $(\mathcal{P}, \Delta, m, \sigma^{\mathbf{MHEA}})$ to its challenger, gets a result v, and returns v to $\mathcal{A}$.

Corruption Queries. When $\mathcal{B}$ receives a corruption query id, it proceeds as follows:

(a) $\mathcal{B}$ makes a corruption query id to its challenger and gets a secret key $sk_{id}^{\mathbf{MHEA}}$. If $sk_{id}^{\mathbf{MHEA}} = \perp$ holds, then it also gives $\perp$ to $\mathcal{A}$.

(b) $\mathcal{B}$ generates $(ek_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHE}}) \leftarrow \mathbf{MHE.KG}(pp^{\mathbf{MHE}}, id)$.

(c) $\mathcal{B}$ sets $sk_{id} := (sk_{id}^{\mathbf{MHE}}, sk_{id}^{\mathbf{MHEA}})$ and gives sk_{id} to $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a forgery $(\mathcal{P}^*, \Delta^*, c^*)$ and terminates, $\mathcal{B}$ proceeds as follows:

(a) $\mathcal{B}$ parses $c^* := (c^{\mathbf{MHE}^*}, \sigma^{\mathbf{MHEA}^*})$ and $\mathcal{P}^* := (f^*, id_1^*, \cdots, id_t^*)$.

(b) $\mathcal{B}$ computes $m^* \leftarrow \mathbf{MHE.Dec}(\mathcal{P}^*, (sk_{id}^{\mathbf{MHE}})_{id \in \mathcal{P}^*}, c^{\mathbf{MHE}^*})$.

(c) $\mathcal{B}$ outputs $(\mathcal{P}^*, \Delta^*, m^*, \sigma^{\mathbf{MHEA}^*})$ to its challenger and terminates.

We can see that $\mathcal{B}$ perfectly simulates the unforgeability game for $\mathcal{A}$. In the following, we show that $\mathcal{B}$ can output a valid forgery $(\mathcal{P}^*, \Delta^*, m^*, \sigma^{\mathbf{MHEA}^*})$ if $\mathcal{A}$ outputs a valid forgery $(\mathcal{P}^*, \Delta^*, c^*)$. Firstly, if $\mathcal{A}$ outputs a valid forgery $(\mathcal{P}^*, \Delta^*, c^*)$, due to the construction of

$\mathbf{MVHE}$, $\perp \neq m^* \leftarrow \mathbf{MHE.Dec}(\mathcal{P}^*, \Delta^*, (sk_{id})_{id\in\mathcal{P}^*}, c^{\mathbf{MHE}^*})$, $1 = \mathbf{MHEA.Ver}((sk_{id}^{\mathbf{MHEA}})_{id\in\mathcal{P}^*}, m^*, \mathcal{P}^*, \sigma^{\mathbf{MHEA}^*})$, and $id \notin L_{corr}$ hold for all $id \in \mathcal{P}^*$. Next, if $(\mathcal{P}^*, \Delta^*, c^*)$ is valid, then $(\mathcal{P}^*, \Delta^*, c^*)$ satisfies at least one of the condition of Type X ($X \in \{1, 2, 3\}$). For any $X \in \{1, 2, 3\}$, we can see that if $(\mathcal{P}^*, \Delta^*, c^*)$ satisfies the condition of Type X, then $(\mathcal{P}^*, \Delta^*, m^*, \sigma^{\mathbf{MHEA}^*})$ (output by $\mathcal{B}$) also satisfies the condition of Type X in the game of unforgeability for $\mathbf{MHEA}$. Thus, if $\mathcal{A}$ outputs a valid forgery, then $\mathcal{B}$ can output a valid forgery. That is, we have $\mathsf{Adv}^{\mathsf{unf}}_{\mathbf{MVHE},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\mathsf{unf}}_{\mathbf{MHEA},\mathcal{B}}(\lambda)$. Since $\mathbf{MHEA}$ satisfies unforgeability, for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{unf}}_{\mathbf{MVHE},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$ holds. Therefore, $\mathbf{MVHE}$ satisfies unforgeability.

□ **(Theorem 11)**

Chapter 4

Conclusion

In recent years, secret sharing-based MPC and homomorphic encryption-based MPC have become two mainstream construction methods in the field. In this thesis, we explored two distinct aspects of the MPC domain, each built upon these different methods.

In our first result, we give a new two-party exponentiation protocol based on an additive secret sharing scheme which is compatible with well-known dishonest-majority MPC frameworks. The efficiency of our protocol is characterized by two cases. If we need modulus conversion in our protocol, it requires 3 rounds and 4 invocations of MPC multiplication. In contrast, it requires only 2 rounds and 3 invocations of MPC multiplication if we do not need modulus conversion. The core techniques for obtaining our protocol are two-fold. One is an efficient constrained quotient transfer protocol which only works on even numbers without bit-decomposition. The other is an efficient modulus conversion protocol based on the above efficient quotient transfer protocol. We believe that these two primitives might be of independent interest and could have further applications. We leave it as an interesting open problem to investigate how to extend our protocol to support non-integer values such as fixed-point numbers.

Furthermore, we propose a new notion of multi-key verifiable homomorphic encryption (MVHE) as a core cryptographic primitive for realizing private and verifiable delegating computation in the multi-user setting as an application. Concretely, we provide a formal syntax and security definitions for MVHE and the generic construction based on an MHE scheme and an MHEA scheme. As mentioned in Section 3.2.3, due to the restriction for our MHEA scheme occurred by the previous frameworks, our MVHE scheme (i) is resilient to only fixed a-priori bounded number of verification queries and (ii) supports only constant number of users. Then, we leave as an interesting open problem to examine how to construct a fully

secure VHE scheme for all polynomial-sized circuits overcoming these restrictions.

Bibliography

[ISO] ISO/IEC 19592-2:2017(en) Information technology — Security techniques — Secret sharing — Part 2: Fundamental mechanisms.

[AA20] Patra Arpita, Suresh Ajith. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. Proceedings 2020 Network and Distributed System Security Symposium, pages 459-480, 2020.

[AAN18] Abdelrahaman Aly, Aysajan Abidin, and Svetla Nikova. Practically efficient secure distributed exponentiation without bit-decomposition. In *FC 2018*, LNCS, pages 291–309. Springer, Heidelberg, February / March 2018.

[ALSZ15] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 673–701. Springer, Heidelberg, April 2015.

[BCP+20] Megha Byali, Harsh Chaudhari*, Arpita Patra, and Ajith Suresh. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. Proceedings on Privacy Enhancing Technologies, pages 459-480, 2020.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[CCPS19] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, Ajith Suresh. ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction. Proceedings

of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, pages 81–92, 2019.

[CRS20] Harsh Chaudhari, Rahul Rachuri, Ajith Suresh. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. Proceedings on 27th Annual Network and Distributed System Security Symposium, 2020.

[CVA18] Juvekar Chiraag, Vaikuntanathan Vinod, Chandrakasan Anantha. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. Proceedings of the 27th USENIX Conference on Security Symposium, pages 1651–1668, 2018.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 285–304. Springer, Heidelberg, March 2006.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, September 2013.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 247–264. Springer, Heidelberg, August 2003.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[GGN16] Rosario Gennaro, Steven Goldfeder and Arvind Narayanan. Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. In *ACNS 2016*, LNCS, pages 156–174. Springer, Heidelberg, June 2016.

[KIM+18] Ryo Kikuchi, Dai Ikarashi, Takahiro Matsuda, Koki Hamada, and Koji Chida. Efficient bit-decomposition and modulus-conversion protocols with an honest ma-

jority. In Willy Susilo and Guomin Yang, editors, *ACISP 18*, volume 10946 of *LNCS*, pages 64–82. Springer, Heidelberg, July 2018.

[KLR10] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-Theoretically Secure Protocols and Security under Composition. Read More: https://epubs.siam.org/doi/10.1137/090755886 SIAM J.Comput., vol.39, no.5, pp.2090–2112, 2010.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842. ACM Press, October 2016.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Heidelberg, April / May 2018.

[KRC+20] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma. CrypTFlow: Secure TensorFlow Inference. Proceedings 2020 IEEE Symposium on Security and Privacy, pages 336–353, 2020.

[Lin17] Yehuda Lindell. Fast Secure Two-Party ECDSA Signing. In *CRYPTO 2017*, LNCS, pages 613–644. Springer, Heidelberg, August 2017.

[MLS+20] Pratyush Mishra and Ryan Lehmkuhl and Akshayaram Srinivasan and Wenting Zheng and Raluca Ada Popa. Delphi: A Cryptographic Inference Service for Neural Networks. Proceedings of the 29th USENIX Conference on Security Symposium, pages 2505–2522, 2020.

[NX11] Chao Ning and Qiuliang Xu. Constant-rounds, linear multi-party computation for exponentiation and modulo reduction with perfect security. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 572–589. Springer, Heidelberg, December 2011.

[OWIO19] Kazuma Ohara, Yohei Watanabe, Mitsugu Iwamoto, and Kazuo Ohta. Multi-Party Computation for Modular Exponentiation Based on Replicated Secret Sharing. IEICE Trans. Fundam. Electron. Commun. Comput. Sci., 102(9): 1079–1090.

[RSC+19] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter and Farinaz Koushanfar. XONN: XNOR-based Oblivious Deep Neural Network Inference. Proceedings of the 28th USENIX Conference on Security Symposium, pages 1501–1518, 2019.

[WWW+14] Yujue Wang, Duncan S. Wong, Qianhong Wu, Sherman S. M. Chow, Bo Qin and Jianwei Liu. Practical Distributed Signatures in the Standard Model. In *CT-RSA 2014*, LNCS, pages 307–326. Springer, Heidelberg, February 2014.

[AJJM20] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In *TCC 2020, Part I*, LNCS, pages 28–57. Springer, Heidelberg, March 2020. doi:10.1007/978-3-030-64375-1_2.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012. doi:10.1145/2090236.2090263.

[BCFK21] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. LNCS, pages 528–558. Springer, Heidelberg, 2021. doi:10.1007/978-3-030-75248-4_19.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 190–213. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53018-4_8.

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. Cryptology ePrint Archive, Report 2021/808, 2021. https://eprint.iacr.org/2021/808.

[CKKC13] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Carlos Cid. Multi-client non-interactive verifiable computation. In Amit Sahai, editor, *TCC 2013*,

volume 7785 of *LNCS*, pages 499–518. Springer, Heidelberg, March 2013. doi:10.1007/978-3-642-36594-2_28.

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 630–656. Springer, Heidelberg, August 2015. doi:10.1007/978-3-662-48000-7_31.

[CZW17] Long Chen, Zhenfeng Zhang, and Xueqing Wang. Batched multi-hop multi-key FHE from ring-LWE with compact ciphertext extension. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 597–627. Springer, Heidelberg, November 2017. doi:10.1007/978-3-319-70503-3_20.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.

[FGP14] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 844–855. ACM Press, November 2014. doi:10.1145/2660267.2660366.

[FMNP16] Dario Fiore, Aikaterini Mitrokotsa, Luca Nizzardo, and Elena Pagnin. Multi-key homomorphic authenticators. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 499–530. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53890-6_17.

[FN16] Dario Fiore and Anca Nitulescu. On the (in)security of SNARKs in the presence of oracles. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 108–138. Springer, Heidelberg, October / November 2016. doi:10.1007/978-3-662-53641-4_5.

[FNP20] Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In *PKC 2020, Part II*, LNCS, pages 124–154. Springer, Heidelberg, 2020. doi:10.1007/978-3-030-45388-6_5.

[FP18] Dario Fiore and Elena Pagnin. Matrioska: A compiler for multi-key homomorphic signatures. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume

11035 of *LNCS*, pages 43–62. Springer, Heidelberg, September 2018. doi:10.1007/978-3-319-98113-0_3.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Heidelberg, August 2010. doi:10.1007/978-3-642-14623-7_25.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009. doi:10.1145/1536414.1536440.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Heidelberg, August 2013. doi:10.1007/978-3-642-40084-1_30.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. doi:10.1145/28395.28420.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015. doi:10.1145/2746539.2746576.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 301–320. Springer, Heidelberg, December 2013. doi:10.1007/978-3-642-42045-0_16.

[LDPW14] Junzuo Lai, Robert H. Deng, HweeHwa Pang, and Jian Weng. Verifiable computation on outsourced encrypted data. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part I*, volume 8712 of *LNCS*, pages 273–291. Springer, Heidelberg, September 2014. doi:10.1007/978-3-319-11203-9_16.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012. doi:10.1145/2213977.2214086.

[LTWC18] Russell W. F. Lai, Raymond K. H. Tai, Harry W. H. Wong, and Sherman S. M. Chow. Multi-key homomorphic signatures unforgeable under insider corruption. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 465–492. Springer, Heidelberg, December 2018. doi:10.1007/978-3-030-03329-3_16.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_26.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999. doi:10.1007/3-540-48910-X_16.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 422–439. Springer, Heidelberg, March 2012. doi:10.1007/978-3-642-28914-9_24.

[PS16] Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 217–238. Springer, Heidelberg, October / November 2016. doi:10.1007/978-3-662-53644-5_9.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On Data Banks and Privacy Homomorphisms. Foundations of Secure Communication, pages 169–177, Academic Press.

[RSA78] Ronald L. Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of ACM, 21(2)(1978), pages 120–126.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982. doi:10.1109/SFCS.1982.38.