Input Certification for Secure Computation

by

Myoungin Jeong

SEPTEMBER 2018

A dissertation submitted to the

Faculty of the Graduate School of

the University at Buffalo, State University of New York

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

Department of Mathematics

Contents

List of Notations

$\mathbb{N}$	Set of natural numbers.
$\mathbb{R}_{\geq 0}$	Set of positive real numbers.
$\mathbb{F}_p$	A field of order p.
κ	A security parameter.
$p(\cdot)$	A polynomial.
$\epsilon(\cdot), \mathsf{negl}$	A negligible function.
$e(\cdot, \cdot)$	A bilinear map.
$\mathbb{Z}_q$	Set of integers modulo p.
$G = \langle g \rangle$	A group G generated by g.
$x_m = com(m, r)$	A commitment to m using randomness r.
$\|\|$	A concatenation.
$\{0,1\}^n$	Set of all bit-strings of length n.
$\{0,1\}^*$	Set of all finite bit-strings.

H	A hash function.
h	A hash function with homomorphic XOR property.
PRG	A pseudorandom generator.
$X \stackrel{c}{\equiv} Y$	X and Y is computationally indistinguishable.
Pr[X]	A probability of event X.
(pk, sk)	A public/private key-pair.
$\wedge$	The AND operator.
$\oplus$	The exclusive-or (XOR) operator.
OT_l^m	m independent oblivious transfers of l-bits strings.
$\neg y$	The complement of a binary bit y
$\perp$	An empty response

List of Tables

List of Figures

Abstract

Secure multi-party computation (SMC) allows its participants to securely compute functionality without revealing their private input and has broad application across a diverse spectrum. In general, to prove the security of a cryptographic construction standard security model, one must consider participants as semi-honest or with malicious intent. However, we treat the problem outside of traditional security models even in the presence of malicious participants. We strengthening the security of SMC protocol with the ability to guarantee that the participants provide truthful inputs in the computation. In other words, we focus on enforcing input correctness.

To begin this examination, we combine SMC techniques based on secret-sharing with signatures in order to enforce input correctness in the form of certification. We modify two signature schemes, the Camenisch-Lysyanskaya scheme and ElGamal scheme, to achieve private verification and efficiency of batch verification. Consequently, this shows the potential for integration with two prominent SMC protocols.

Next, we utilize a certificate issued by a certification authority to verify the user's input correctness and consequently use it in the secure computation. In this work, we treat the enforcing correctness of evaluator's inputs to the two-party computation based

on a garbled circuit evaluation in the presence of malicious participants. For the purpose of discourse, we modify the oblivious transfer (OT)/OT extension and construct new protocol with the goal of achieving efficient computation.

Chapter 1

Introduction

1.1 Introduction

Secure multi-party computation (SMC) protects the confidentiality of private data during computation in distributed or outsourced settings while allowing two or more participants to jointly evaluate a function. This is a mature research field with a variety of applications for secure evaluation of arbitrary functions by two or more computational parties that does not allow access to all inputs in the clear. The rapid development of this field in recent years has greatly reduced SMC overhead and accordingly, the development of SMC solutions is on the rise [5, 6, 30].

A standard formulation of SMC allows a set of participants to jointly evaluate some function f on the private input $\mathsf{in}_1, \ldots, \mathsf{in}_k$ ($k \geq 1$) from different sources by m (≥ 2) computational parties and to produce s (≥ 1) outputs which get revealed to the designated parties. For the security reason, the input of each participant should not be exposed to other participants and be able to share the desired results as agreed in advance. Standard

security definitions model the participants as either semi-honest or malicious. In the semi-honest model, the participants correctly follow the prescribed computation even if he is corrupted. Conversely in the malicious model, the corrupted parties can arbitrarily deviate from the computation in the attempt to learn unauthorized information about other parties' inputs. Output correctness guarantees must also hold in these respective models. However, these definitions provide no guarantees with respect to what inputs are entered into the computation. Therefore, a malicious participant can modify its real input thereby attempting to harm security or correctness. For example, the participant can perturb his input in such a way that all output recipients receive incorrect information, but is able to compensate for the error and learn the correct result. Alternatively, the participant can modify his input in such a way as to learn the maximum amount of information concerning private data of others', beyond what would have been available if the computation was run on truthful inputs ([4] gives an example of this kind of attack in the context of computing with genomic data). These attacks are beyond the scope of standard SMC security models and cannot be mitigated.

In this work, we study input enforcement which enforces the correct (i.e., truthful) input to be used in the SMC via input certification. At the time of computation initiation, a party supplying input accompanies it with a certificate and proves that the data input into the computation is identical to what has been certified. It goes without saying that the certificate and its verification must maintain data confidentiality. There are many types of data which can be generated or can be verified by an authority (such as the government, a medical facility, etc.) that issues private certification to the user at that time.

Past research has addressed the topic of enforcing input correctness via input certification for specific SMC applications (e.g., anonymous credentials [10] or set operations [12, 17]), and more recently for general functions [4, 27, 47]. In that kind of research, almost all of the efforts have been focused on the general case for secure two-party computation based on garbled circuits (GCs). This presents an interesting quandary to explore because GC evaluation does not naturally combine with signature or certification techniques. This also presents the issue that this dilemma deserves attention beyond GCs. With this in mind, we will treat the problem of input certification in the multi-party setting based on secret sharing. Because both secret sharing and signature schemes exhibit algebraic structure, the use of signatures appears to be a natural choice in enabling input certification in secret-sharing based SMC. Note that, unlike many other conventional uses of signatures, this problem setting requires more security. Signature verification needs to be performed privately without revealing any information about the signed message to the verifier. Another important consideration is that in many SMC applications, the size of the input is large (e.g., genomic data). Because signatures are built using rather expensive public-key techniques, which in the privacy-preserving setting often needs to be combined with zero-knowledge proofs, we are interested in improving signature verification time using batch verification of multiple signatures.

In Chapter 3, we study two types of signatures in the context of this problem, CL-signatures and ElGamal signatures.: (i) CL-signatures [8, 9] which were designed for anonymity applications and achieve both message privacy and unlinkability of multiples showings of the same signature and (ii) conventional ElGamal signatures [18]. After formulating the necessary security guarantees of private signature verification, we show that

the signature showing in [9] can be simplified to meet new definition of message privacy, and then construct a batch verifier for the resulting signature. In the case of ElGamal signatures, we first modify a provably secure ElGamal signature scheme from [40] to achieve private verification and consequently construct a batch verifier for the resulting algorithm. The batch verifiers use the Small Exponents Technique [3] to randomize multiple signatures to ensure that batch verification can succeed *only* when all individual signatures are valid.

The last part of Chapter 3 deals with combining the developed signature schemes with SMC techniques securely in the malicious model. We identify two prominent constructions of SMC based on the secret sharing: (i) Damgård-Nielsen solution [15] of low communication complexity where the number of corrupted parties is below $k/3$ and (ii) an improvement to SPDZ [14] with a very fast online phase that tolerates any number of corruptions. We next show how to modify the input phase to use new signatures with an additional optimization of utilizing a single commitment to multiple signatures instead of using individual commitments. Finally, we implement the new ElGamal-based signature scheme and SPDZ-based use of certified inputs for a varying number of messages (SMC inputs), and show the corresponding result of efficient performance. The techniques are general enough to be applicable to other signature algorithms (such as ElGamal-based DSA and others).

We will also examine how to use the construction of certification with respect to the oblivious transfer protocol, which is the most fundamental primitive and widely used by people for SMC. It is GC-based protocol two-party protocol between a sender and a receive. The sender has a pair of strings and the receiver has selection bits. As a

result of the protocol, the receiver gets only one string depending on his choice, and the sender must not know which string the receiver has come to know. It is more interesting because it is difficult to integrate with other cryptographic schemes, unlike signature-based cryptosystem. The starting point of second topic is the work of Zhang et al. [47] for two-party setting. In the setting of two-party case, we also assume that there exists a certification authority that can issue the certification of user's data. The certification must not reveal any information about the certified data and after the certification step is completed, we use this certification in the secure two-party computation.

In Chapter 4 we discuss two possibilities: (i) The case where the evaluator's input is small, i.e., $m \leq \kappa$ for a computational security parameter κ and (ii) The case where the evaluator's input size is large, i.e., $m > \kappa$. For the first case, our starting point is Naor-Pinkas OT [38], and for the second case we use an OT extension that results in a more efficient solution. Our construction consists of two steps: input certification step and secure two-party computation. We modify Naor-Pinkas OT [38] adding the certification step and further advance Asharov et al.'s OT extension protocol [1]. After that, we construct an OT extension protocol that allows for certification to be reusable for multiple circuits.

1.2 Related work

Past publications on certified inputs concerning SMC that we examined were mentioned above, (i.e., techniques for specific applications [10, 12, 17] as well as techniques for GCs [4, 27, 47]). Additionally, work on using game theory to incentivize rational players to

enter their inputs truthfully (see, e.g., [25, 44] among others) was discussed as well. In this section, we will examine these publications in the scope of two sections: in terms of signature scheme; and GC-based construction.

1.2.1 Signature scheme with certified input

The first systematic treatment of batch signature verification appears in [3], although interest in batch verification of signatures and other cryptographic operations goes further back. Modern techniques for batch verification include: [7, 19] among others, although none of them target private verification (defined in section 2.1) which is a central pillar to this work.

Aggregate signatures benefit our work as it allows a number of different signatures to be compiled into a single short signature to save bandwidth in resource-constrained environments. We hope to build on the aggregate signature schemes that were developed for CL signatures previously [31]. However, there are two central differences to note in this work: (i) aggregate signatures have strictly weaker security guarantees than batch verification [7] because verification of an aggregate signature can succeed even if the individual signatures included in it do not verify, and (ii) message privacy was not considered. Case in point, Guo et al. [24] uses privacy features of CL signatures and constructs an aggregate CL signature, but the difference in the security guarantees still stands.

1.2.2 Oblivious transfer protocol with certified input

The idea and applications of GC were first presented by Yao in 1986 [46]. It is known that GC can safely evaluate any computable function in a two-party setting. The structure

introduced by Yao is safe from semi-honest participants, and includes solutions to deal with some malicious behavior. Initially published construction including [22, 23] used zero knowledge proofs of knowledge to cope with malicious participants and a cut-and-choose technique was used as an alternative to handle the malicious model in garbled circuit-based structures [33, 35, 42].

In our study, we focus on the input correctness in the malicious case. This dilemma has had several publications published to solve input consistency problems. For example, input consistency problems have arisen when multiple circuits in a Garbled circuit have had to evaluate same inputs, and solutions based on cut-and-choose techniques have been addressed in several publications [35, 37, 45]. In another study of this issue, Kolesnikov et al. [29] proposed a solution that allows a malicious participant to perform multiple secure two-party computations using a semi-honest server assistant at low cost. In this solution they used an efficient consistency check method across multiple two-party executions. Other related studies (see, e.g., [25, 43, 44] among others) apply game theory to function design to incentivize participants to enter correct input. These techniques however, are not widely applicable to general functionalities.

In a recent work on enforcing input correctness, Zhang et al. [47] utilized certification authority (CA) to certify user's input, which consequently is used in secure garbled circuit evaluation two-party computations. Zhang focused on enforcing correctness of the garbler's inputs via certification in the presence of malicious participants and showed how to integrate certificates into secure computations, guaranteeing correctness of the garbler's input. This paper is used as a starting point for this work. We study how to use the certificate produced by CA to force the correct evaluator's input in GC based two-party

computation and tie it with secure function evaluation.

Lastly, the publication on oblivious transfer by Naor and Pinkas [38] examines efficient non-interactive oblivious transfer protocol without random oracle. We combine the certification produced by CA with this protocol to enforce evaluator's input correctness when the input size is small. For the case that the input size is greater than the security parameter, we improve the consistency check step of OT extension technique in [1], consequently tying the concept of certification with it to enforce the input correctness of evaluator in the malicious model.

Chapter 2

Preliminaries

In this chapter, we describe notations and definitions that are used throughout the dissertation. We also present the cryptographic background and primitives we use, and construct our protocol based on these.

2.1 Definitions

A function $\epsilon : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is *negligible* if for every positive polynomial $p(\cdot)$ there exists an integer N such that for all $\kappa > N$ $\epsilon(\kappa) < \frac{1}{p(\kappa)}$. We denote a negligible function by negl. The notation $G = \langle g \rangle$ means that g generates group G. We rely on groups with pairings, defined as follows.

Definition 2.1.1 (Bilinear map)**.** A one-way function $e : G \times G \to \mathbf{G}$ is a bilinear map if it satisfies following properties:

- *Efficient*: G and $\mathbf{G}$ are groups of the same prime order q and there exists an efficient algorithm for computing e.

- *Bilinear*: For all $g, h \in G$ and $a, b \in \mathbb{Z}_q$, $e(g^a, h^b) = e(g, h)^{ab}$.

- *Non-degenerate*: If g generates G, then $e(g, g)$ generates $\mathbf{G}$.

We assume that there is a trusted setup algorithm Setup that, on input 1^κ for a security parameter κ, outputs the setup for group $G = \langle g \rangle$ of prime order $q \in \Theta(2^\kappa)$ that has a bilinear map e, and $e(g, g)$ generates $\mathbf{G}$ of order q. That is, $(q, G, \mathbf{G}, g, e) \leftarrow \mathsf{Setup}(1^\kappa)$.

2.2 Signature Scheme

A signature scheme consists of three algorithms; which are Key generation, Signing and Verification. We first define a signature scheme and describe a security of a signature scheme in terms of negligible function.

Definition 2.2.1 (Signature scheme)**.** A *signature scheme* consists of following three algorithms:

- KeyGen is a probabilistic polynomial-time (PPT) algorithm that, on input a security parameter 1^κ, generates a public-private key pair (pk, sk).

- Sign is a PPT algorithm that, on input a secret key sk and message m from the message space, outputs signature σ.

- Verify is a deterministic polynomial-time algorithm that, on input a public key pk, a message m and a signature σ, outputs a bit.

Security of a signature scheme is defined as "difficulty of existential forgery under a chosen-message attack by any PPT adversary". Let $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme and consider the following experiment:

Experiment $\mathsf{ForgeSig}_{\mathcal{A},\Pi}(\kappa)$:

1. The challenger creates a key pair $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and gives pk to $\mathcal{A}$.

2. $\mathcal{A}$ has oracle access to $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$. For each message m that $\mathcal{A}$ queries the oracle, m is stored in list $\mathcal{Q}$ and $\mathcal{A}$ learns $\sigma = \mathsf{Sign}_{sk}(m)$. $\mathcal{A}$ eventually outputs a pair (m^*, σ^*).

3. The experiment outputs 1 if both $\mathsf{Verify}_{pk}(m^*, \sigma^*) = 1$ and $m^* \notin \mathcal{Q}$. Otherwise, it outputs 0.

Using this experiment, we can define the security of a signature scheme as follows.

Definition 2.2.2 (Security of a signature scheme [28])**.** A signature scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is *existentially unforgeable under an adaptive chosen-message attack* if for all PPT adversaries $\mathcal{A}$ there is a negligible function negl such that

$$\Pr[\mathsf{ForgeSig}_{\mathcal{A},\Pi}(\kappa) = 1] \leq \mathsf{negl}(\kappa).$$

2.2.1 CL signature Scheme A

Next we will build on Camenisch-Lysyanskaya signature Scheme A from [9] (CL Scheme A for short), defined as follows:

Key generation: On input 1^κ, execute $(q, G, \mathbf{G}, g, e) \leftarrow \mathsf{Setup}(1^\kappa)$, choose random $x, y \in \mathbb{Z}_q$ and compute $X = g^x, Y = g^y$. Set $sk = (x, y)$ and $pk = (q, G, \mathbf{G}, g, e, X, Y)$.

Signing: On input message $m \in \mathbb{Z}_q$, secret key $sk = (x, y)$ and public key $pk = (q, G, \mathbf{G}, g, e, X, Y)$, choose random $a \in G$ and output $\sigma = (a, b, c) = (a, a^y, a^{x+mxy})$.

Verification: On input message m, $pk = (q, G, \mathbf{G}, g, e, X, Y)$, and signature $\sigma = (a, b, c)$, check whether $e(a, Y) = e(g, b)$ and $e(X, a) \cdot e(X, b)^m = e(g, c)$. If both equalities hold, output 1; otherwise, output 0.

Proof of signature: The prover and verifier have $pk = (q, G, \mathbf{G}, g, e, X, Y)$. The prover also has $m \in \mathbb{Z}_q$ and the corresponding signature $\sigma = (a, b, c) = (a, a^y, a^{x+mxy})$.

1. The prover chooses random $r', r'' \in \mathbb{Z}_q$, computes blinded signature $\tilde{\sigma} = (a^{r''}, b^{r''}, c^{r''r'}) = (\tilde{a}, \tilde{a}^y, (\tilde{a}^{x+mxy})^{r'}) = (\tilde{a}, \tilde{b}, \hat{c})$, and sends it to the verifier.
2. Let $\mathbf{v}_x = e(X, \tilde{a})$, $\mathbf{v}_{xy} = e(X, \tilde{b})$, and $\mathbf{v}_s = e(g, \hat{c})$. The prover and verifier engage in the following ZKPK: $PK\{(\mu, \rho) : \mathbf{v}_x^{-1} = \mathbf{v}_{xy}^{\mu} \mathbf{v}_s^{\rho}\}$.
3. The verifier accepts if it accepts the proof above and $e(\tilde{a}, Y) = e(g, \tilde{b})$.

Unforgeability of CL Scheme A is shown under the LRSW assumption [36], demonstrating that the zero-knowledge property of proving possession of a signature uses no additional assumptions other than hardness of discrete logarithm.

2.2.2 ElGamal Signature Scheme

We also build on ElGamal signature scheme [18]. Because the original construction allows for existential forgeries, we will use a provably secure variant by Pointcheval and Stern [40, 41]. The setup assumes an α-hard prime number p for some fixed α, defined as having $p - 1 = qR$ where q is prime and $R \leq |p|^{\alpha}$. This is necessary when considering the difficulty of discrete logarithm and is more general than requiring the use of prime order q. This signature scheme uses a hash function H which Pointcheval and Stern prove to be unforgeable in the random oracle model (i.e., H is modeled as a random oracle).

Key generation: On input: a security parameter 1^κ, choose a large α-hard prime p and a generator g of $\mathbb{Z}_p^*$. Then choose random $x \in \mathbb{Z}_{p-1}$ and compute $y = g^x \bmod p$. Set $sk = x$ and $pk = (p, g, y)$.

Signing: On input: message m, secret key $sk = x$ and public key $pk = (p, g, y)$, choose random $k \in \mathbb{Z}_{p-1}^*$, compute $t = g^k \bmod p$ and $s \equiv (H(m||t) - xt)k^{-1} \pmod{p-1}$, where $||$ denotes concatenation, then output $\sigma = (t, s)$.

Verification: On input: message m, public key $pk = (p, g, y)$, and signature $\sigma = (t, s)$, check whether $1 < t < p$ and $g^{H(m,t)} \equiv y^t t^s \pmod{p}$. If both conditions hold, output = 1; otherwise, output = 0.

2.3 Batch Verification

Batch verification [3] is a method for verifying a set of signatures on different messages signed by the same or different signers. This method is intended to be more efficient than verifying each signature independently. This work is primarily interested in batch verification of signatures produced by the same signer. Batch verification is defined as:

Definition 2.3.1 (Batch verification of signatures [7])**.** Let $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme and κ be a security parameter. Let (pk_1, sk_1), ..., (pk_n, sk_n) be key pairs of n signers $P_1, \ldots, P_n$ produced by $\mathsf{KeyGen}(1^\kappa)$ and $PK = \{pk_1, \ldots, pk_n\}$. Let Batch be a PPT algorithm that takes a set of tuples (pk_i, m_i, σ_i) and outputs a bit. Then Batch is a batch verification algorithm if the following holds:

- If $pk_i \in PK$ and $\mathsf{Verify}(pk_i, m_i, \sigma_i) = 1$ for all $i \in [1, n]$, then $\mathsf{Batch}((pk_1, m_1, \sigma_1),$

$\ldots, (pk_n, m_n, \sigma_n)) = 1$.

- If $pk_i \in PK$ for all $i \in [1, n]$ and $\mathsf{Verify}(pk_i, m_i, \sigma_i) = 0$ for at least one $i \in [1, n]$, then $\mathsf{Batch}((pk_1, m_1, \sigma_1), \ldots, (pk_n, m_n, \sigma_n)) = 1$ with probability at most $2^{-\kappa}$.

Note: Even if only one verification fails, the verifier should detect that with the probability at least $1 - 2^{-\kappa}$. Also this verification is faster than the independent verification.

2.4 Zero-knowledge Proof of Knowledge and Commitment Scheme

In our construction, we rely on *zero-knowledge proofs of knowledge* (ZKPKs) and commitment scheme for framework. A ZKPK is a two-party interactive protocol between a prover and a verifier, during which the prover convinces the verifier that a certain statement is true without revealing anything else about the values used in the statement. We use a notation $PK\{(\text{Variables}) : \text{statement}\}$ to present a ZKPK of given statements for the variables. Here, the variables are private to the prover and the statements are given to both the prover and the verifier. Informally, a ZKPK should satisfy the following properties:

- *Completeness*: If the statement is true, then an honest verifier will be convinced of the statement's validity after interacting with an honest prover.

- *Soundness*: If the statement is false, then no cheating prover can convince an honest verifier that the statement is true, except with a negligible probability (in the security parameter).

- *Zero-Knowledge*: If the statement is true, then no cheating verifier can learn anything other than the fact that the statement is true.

If this proof is successful, the verifier will accept that the given statement is true even if variables are unknown to the verifier. However, we are interested in simple statements over discrete logarithms such as those described, e.g., in [11, 13].

A *commitment scheme* carries out the following functions: it allows one to commit to message m in such a way that the commitment reveals no information about m and, given a commitment on m, it is not feasible to open it to a value other than m. In other words, once the value m has been committed to, it cannot be changed and kept private until the user reveals it. These properties are known as *hiding* and *binding*. A commitment scheme is defined by Commit and Open algorithms that are correspondingly known as commitment and opening algorithms. We note that Commit is a randomized algorithm and for that reason, use notation $com(m, r)$ to denote a commitment to m using randomness r.

Next, we utilize a well-known Pedersen commitment scheme [39] based on discrete logarithms. The setup consists of a group G of prime order q and two generators g and h. To commit to message $m \in \mathbb{Z}_q$, one chooses random $r \in \mathbb{Z}_q$ and set $com(m, r) = g^m h^r$. To open the commitment, the user reveals r. This commitment scheme is information-theoretically hiding and computationally binding (when the discrete logarithm of h to the base g is not known to the user) under the discrete logarithm assumption.

2.5 Signature Scheme with Privacy

Specifically, we are interested in signature schemes which allow for private verification of signature validity without revealing any information about the signed message. We refer to such schemes as *signature schemes with privacy* and refer to the corresponding verification process as *private verification* to distinguish it from the conventional signature verification process. This property implies that the signature itself reveals no information about the signed message. The rest of this section provides the necessary definitions for signature schemes with privacy. We start by re-defining the traditional formulation of a signature scheme as follows:

Definition 2.5.1 (Signature scheme with privacy)**.** A *signature scheme with privacy* consists of the following polynomial-time algorithms:

- KeyGen is a PPT algorithm that, on input a security parameter 1^κ, generates a public-private key pair (pk, sk).

- Sign is a PPT algorithm that, on input a secret key sk and message m from the message space, outputs signature σ and optional auxiliary data x_σ.

- PrivVerify is a potentially interactive algorithm, in which both the prover and the verifier hold a public key pk. The prover has access to m and (σ, x_σ) output by Sign, then supplying a message encoding x_m and a (possibly modified) signature $\tilde{\sigma}$ to the verifier. Lastly the verifier outputs a bit.

This definition brings up the interesting case of allowing for two possibilities: either x_σ produced during signing can be used to form x_m used during verification, or x_σ is

empty and anyone with access to σ and m can compute a suitable (possibly randomized) x_m for signature verification.

Again to ensure unforgeability, PrivVerify must verify signature $\tilde{\sigma}$ similar to the way Verify would and must enforce that the prover knows the message m (encoded in x_m) to which the $\tilde{\sigma}$ corresponds. Because x_m in our work always takes the form of a commitment to m, $com(m, r)$, we explicitly incorporate this in new security definition.

The security (unforgeability) experiment of a signature with privacy is similar to the conventional definition of ForgeSig with two conceptual differences: (i) After producing the challenge pair $(\tilde{\sigma}^*, x_{m^*})$, the adversary $\mathcal{A}$ is required to prove in zero-knowledge that x_{m^*} corresponds to a message, a signature on which has not been queried before. (ii) The signature forging experiment now invokes modified verification algorithm PrivVerify instead of Verify. The signature is verified against a committed value $x_{m^*} = com(m^*, r)$, but the prover is also required to prove the knowledge of message m^* itself encoded in the commitment. Thus, we obtain the following:

Experiment $\mathsf{ForgePrivSig}_{\mathcal{A},\Pi}(\kappa)$:

1. The challenger creates a key pair $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and gives pk to $\mathcal{A}$.

2. $\mathcal{A}$ has oracle access to $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$. For each message m that $\mathcal{A}$ queries the oracle, m is stored in list $\mathcal{Q}$ and $\mathcal{A}$ learns $(\sigma, x_\sigma) = \mathsf{Sign}_{sk}(m)$.

3. The challenger and $\mathcal{A}$ engage in PrivVerify, as part of which $\mathcal{A}$ reveals the challenge pair $(x_{m^*}, \tilde{\sigma}^*)$. $\mathcal{A}$ proves in ZK that it knows the opening of the commitment $x_{m^*} = com(m^*, r)$ and that $m^* \notin \mathcal{Q}$.

4. Output 1 if $\mathsf{PrivVerify}$ returns 1 and all other checks succeed; otherwise, output returns 0.

To model private verification, we define the following message indistinguishability experiment for a signature scheme with privacy $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrivVerify})$:

Experiment $\mathsf{MesInd}_{\mathcal{A},\Pi}(\kappa)$:

1. The challenger creates a key pair $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and gives pk to $\mathcal{A}$.

2. $\mathcal{A}$ has oracle access to $\mathsf{Sign}_{sk}(\cdot)$ and learns the algorithm's output for messages of its choice. $\mathcal{A}$ eventually outputs a pair (m_0, m_1).

3. The challenger draws a random bit $b \in \{0, 1\}$. Upon $\mathcal{A}$'s request, it executes $(\sigma_b, x_{\sigma_b}) \leftarrow \mathsf{Sign}_{sk}(m_b)$. It computes x_{m_b} and returns $(\tilde{\sigma}_b, x_{m_b})$ where $\tilde{\sigma}_b$ is derived from σ_b. If x_{m_b} and/or $\tilde{\sigma}_b$ are probabilistic, $\mathcal{A}$ can request multiple encodings $(\tilde{\sigma}_b^{(i)}, x_{m_b}^{(j)})$ for the same signature and $i, j \in \mathbb{N}$. These signature verification queries are repeated the desired number of times.

4. $\mathcal{A}$ eventually outputs a bit b'. The experiment outputs 1 if $b = b'$, and 0 otherwise.

Definition 2.5.2 (Private Verification)**.** Signature Scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrivVerify})$ is said to achieve *private verification* if for all PPT adversaries $\mathcal{A}$ there is a negligible function negl such that $\Pr[\mathsf{MesInd}_{\mathcal{A},\Pi}(\kappa) = 1] \leq \frac{1}{2} + \mathsf{negl}(\kappa)$.

On the relationship of private verification and proving possession of a signature in zero-knowledge. Prior work on using signatures in privacy-preserving contexts [8, 9] allows for proving possession of a signature in ZK. Their definition implies that no information

about the signed message is revealed and two instances of proving knowledge of a signature cannot be linked to each other. Our new definition of private verification is weaker in the sense that we do not attempt to hide whether the same or different signature is verified at two different times, but ultimately we fully protect the signed data itself. Unlinkability of signature showings is generally not needed in this application, as a user can use its data (e.g., DNA data) in multiple computations and does not need to hide the fact that the same data was used (which can be determined from the computation itself). Therefore, we need only to protect information about the signed values and this difference allows for a faster signature verification process while maintaining the necessary level of security.

When we consequently discuss batch verification of signatures with privacy, we modify the interface of Batch to match that of PrivVerify.

2.6 Security Models

In this section, we describe the ideal/real simulation paradigm discussed later in Chapter 4. We construct protocols for a secure computation of a functionality in the presence of malicious adversaries based on the general simulation-based definition [21]. While we consider the two party case that the participants are (P_1, P_2), this can be extended to the multi-party cases. Again for security reasons, we do not want participants to get any information except their own inputs and results in the computation.

First, we will describe definitions in order to define a secure computation. We call $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ a *functionality*, that is for input (x, y) the desired output $f(x, y)$ is a pair of variables. Two-distribution X and Y is considered *computa-*

tionally indistinguishable, if for every non-uniform polynomial-time algorithm D, there exists a negligible function negl such that:

$$\Big|Pr[D(X) = 1] - Pr[D(Y) = 1]\Big| \leq \mathsf{negl}(\kappa)$$

and denote $X \stackrel{c}{\equiv} Y$.

Next, we define the security of two party computation protocol between P_1 and P_2 using a simulation paradigm in a standard real-ideal model setting in the presence of malicious participants. In this simulation, we define an ideal model and define the security such that the execution of a protocol in the real model can be simulated by the ideal model.

The Ideal Model: In the ideal model, we assume that there is a $\mathsf{trusted\ third\ party}$ (TP) and it helps the computation of f. In the execution of protocol, each party holds an input, denoted x and y and sends it to the TP. An honest party always sends his truthful input but a malicious party may either send some arbitrary input $x' \in \{0,1\}^{|x|}$, $y' \in \{0,1\}^{|y|}$ or abort depending on their real input to the TP. If the TP received an input (x, y) then we must answer P_1 with $f_1(x, y)$ to P_2 with $f_2(x, y)$. Otherwise, the TP answers to one of two/both parties $\perp$. In this case, it may stop the process. Note, an honest party always outputs the result received from the TP. On the other hand, a malicious party may output an arbitrary function of its initial input and the message received from the TP.

The Real Model: In the real model, the real protocol is executed and a malicious party may follow any probabilistic polynomial-time algorithm. The protocol π is executed to compute the functionality f.

In the ideal model, let $S = (S_1, S_2)$ be a pair of non-uniform probabilistic expected polynomial-time machines. S_1 and S_2 are two parties of the protocol. We can say that the pair is *admissible* if at least one of $i \in \{1, 2\}$, S_i is honest and follows the ideal execution. Then the *joint execution of f in the ideal model on input* (x, y) is denoted $\text{IDEAL}_{f,S}(x, y)$ and defined as the result of the protocol executed by S_1 and S_2.

By using the definition of ideal and real models, we can define security of protocols. Take the assertion that a secure two-party protocol in the real model follows the ideal model in which a TP exists. This means that execution of a secure protocol of admissible pairs in the ideal model could simulate admissible pairs in the real model's executions. The execution of the protocol π in the real model should not reveal any information that is not revealed from the ideal model. Defining the secure computation of a function f is as follows.

Definition 2.6.1. Let f be a functionality and let π be a two-party protocol that computes f. Protocol π is said to securely compute f in the malicious model if for every admissible non-uniform probabilistic polynomial-time machines $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in the real model, there exists a pair of admissible non-uniform probabilistic polynomial-time machines $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ in the ideal model such that:

$$\{\text{IDEAL}_{f,\mathcal{S}}(x, y)\} \stackrel{c}{\equiv} \{\text{REAL}_{\pi,\mathcal{A}}(x, y)\}$$

Where $x, y \in \{0, 1\}^*$.

2.7 Oblivious Transfer

Oblivious transfer (OT) is the key element of Gargled Circuit, which can be used to obtain a wire label that corresponds to the input of an evaluator. OT is two-party protocol that at the begins with the sender having an input and at the end of the protocol, the receiver obtaining some output according to the sender's input to his input bit. In this work, we consider 1-out-of-2 OT protocol, that is the sender's input is two strings (a_0, a_1) and the receiver obtaining either a_0 or a_1 and learning nothing about the other string. The OT functionality is defined as a function f with two inputs (a_0, a_1) and b where $f((a_0, a_1), b) = (\lambda, a_b)$ and λ is a empty string. This definition posits that sender outputs nothing, whereas receiver outputs the string according to his input bit b, correspondingly learning nothing about $a_{\neg b}$. The OT extension allows a number of OTs to be executed with small additional overhead per OT, after a certain number of regular and costly OT protocols (depending on the security parameter) have taken place. There are several publications containing [2, 26, 38, 42] about OT/OT extensions and they are the cornerstone of our construction.

We use the notation OT_l^m to express the m independent oblivious transfers of l-bits strings. The functionality OT_l^m is defined as followed:

Inputs: The sender holds m pairs of l-bit strings, (a_i^0, a_i^1) for $1 \leq i \leq m$. The receiver holds m bits $b = b_1 \dots b_m$.

Outputs: The receiver outputs $a_i^{b_i}$ for $1 \leq i \leq m$ and the sender outputs nothing.

We can analyze the security of the protocol by comparing what an adversary can do with

the protocol and what can be done in the ideal scenario. [32, 34]

Definition 2.7.1. Let f be the function realizing oblivious transfer computation and let π be a two-party protocol that computes f. Protocol π is said to be a secure oblivious transfer protocol if for every pair of admissible non-uniform probabilistic polynomial-time machines $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ for the real model, there exists a pair of admissible non-uniform probabilistic expected polynomial-time machines $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ for the ideal model. This holds for every $a_i^0, a_i^1 \in \{0,1\}^*$ of the same length and every $b_i \in \{0,1\}$ where $i = 1, \ldots, m$ for some $m \in \mathbb{N}$,

$$\{\text{IDEAL}_{f,\mathcal{S}}((a_i^0, a_i^1)_{i=1}^m, b)\} \stackrel{c}{\equiv} \{\text{REAL}_{\pi,\mathcal{A}}((a_i^0, a_i^1)_{i=1}^m, b)\}$$

OT is run between two parties: sender S and receiver R. In our case, in addition to supplying its regular input b, the receiver R also provides auxiliary input known to S, which depends on b. We denote this auxiliary input as $\gamma(b)$. R engages in multiple invocations of OT with different senders S on the same input pair $\langle b, \gamma(b)\rangle$. However, again considering the presence of malicious participants that posit that senders can be corrupt and colluding, and the security guarantees of the original OT/OT extension must hold. Now, we re-define the *secure oblivious transfer protocol* with the additional input that correspond to receiver's input b. This definition is followed below:

Definition 2.7.2. Let f be the function realizing oblivious transfer computation and let π be a two-party protocol that computes f. The input of the sender is $A = (a_i^0, a_i^1)_{i=0}^m$ and the input of the receiver is $b = b_1 \ldots b_m$. Additionally, the extra input $\gamma(b)$ is accessible

for both parties. The protocol π could be executed multiple times by one user and the user can use the outputs of each execution at once. We denote $\bigcup$, the union of the view if the protocol is executed multiple times for one receiver's input and combine the view of each execution. Let $k \in \mathbb{N}$ be the number of executions of protocol π. The Protocol π is said to be a secure oblivious transfer protocol if for every pair of admissible non-uniform probabilistic polynomial-time machines $\mathcal{A}_j = (\mathcal{A}_1^j, \mathcal{A}_2)$ for $j = 1, \ldots, k$ for the real model, there exists a pair of admissible non-uniform probabilistic expected polynomial-time machines $\mathcal{S}_j = (\mathcal{S}_1^j, \mathcal{S}_2)$ for $j = 1, \ldots, k$ for the ideal model. This accounts for every $A_j = \left((a_i^0, a_i^1)_{i=0}^m\right)_j \in \{0,1\}^*$ of the same length for $j = 1, \ldots, k$ and every $b_i \in \{0,1\}$ where $i = 1, \ldots, m$ for some $m \in \mathbb{N}$, thereby showing that the union of real views of multiple execution is computationally indistinguishable with the union of simulated views, i.e.,

$$\{ \bigcup_{j=1,\ldots,k} \mathrm{IDEAL}_{f,\mathcal{S}_j}(\langle A_j, \gamma(b)\rangle, \langle b, \gamma(b)\rangle)\} \stackrel{c}{\equiv} \{ \bigcup_{j=1,\ldots,k} \mathrm{REAL}_{\pi,\mathcal{A}_j}(\langle A_j, \gamma(b)\rangle, \langle b, \gamma(b)\rangle)\}$$

2.8 Computational Diffie-Hellman Assumptions

Consider a group $\mathbb{Z}_q$ and its generator g. The *Computational Diffie-Hellman assumption* (CDH) states that given (g, g^a, g^b) with random $a, b \in \mathbb{Z}_q$, there is no probabilistic polynomial time machine which can compute g^{ab}. This means that probabilistic polynomial time machines have a negligible probability of calculating g^{ab} correctly from (g, g^a, g^b) for $a, b \in \mathbb{Z}_q$. In Chapter 4, we rely on the CDH assumption to analyze the security of the construction.

Chapter 3

Signature Schemes with Certified Inputs

In this chapter, we study two type of signatures, the CL signatures [8, 9] and conventional ElGamal signatures [18]. CL signature is designed for anonymity applications and achieves both message privacy and unlinkability of multiple uses of the same signature. After stating the necessary security guarantees of private signature verification, we show that signature showing in [9] can be simplified to meet the definition of message privacy. Next, a batch verifier is constructed for the resulting signature. In the case of ElGamal signatures, we first modify a provably secure ElGamal signature scheme from [40] to achieve private verification and consequently construct a batch verifier for the resulting algorithm. The batch verifiers in this work use the small exponents technique [3] to randomize multiple signatures to ensure that batch verification can succeed only when all individual signatures are valid.

3.1 Constructions based on CL Signatures

In this section, we start by demonstrating that the CL signatures protocol satisfies the conditions of signature privacy and discuss the cost of verifying multiple signatures using that construction. We consequently proceed with simplifying CL Scheme A's verification and construct the corresponding batch verifier.

3.1.1 CL Scheme A

Recall that a signature scheme with privacy is defined as $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrivVerify})$. To use CL Scheme A in this context, we leave KeyGen and Sign unmodified, except that KeyGen additionally computes $h = g^u$ for a random $u \in \mathbb{Z}_q$ and stores it in the public key, i.e., $pk = (q, G, \mathbf{G}, g, h, e, X, Y)$. PrivVerify is realized as follows:

PrivVerify**:** The prover holds signature $\sigma = (a, b, c)$ on private message $m \in \mathbb{Z}_q$ and both parties hold pk. The prover computes $x_m = com(m, r) = g^m h^r$ using random $r \in \mathbb{Z}_q$ and sends x_m to the verifier. The remaining steps are the same as in the proof of signature in Scheme A above, except that the ZKPK in step 2 is modified to: $PK\{(\mu, \rho, \gamma) : x_m = g^\mu h^\gamma \wedge \mathbf{v}_x^{-1} = \mathbf{v}_{xy}^\mu \mathbf{v}_s^\rho\}$.

Note, that the signing algorithm is not modified and in the verification protocol we only extend the ZKPK statement. Because the original proof of signature protocol was shown to be a proof of knowledge and the signature was shown to be unforgeable, this modified scheme satisfies signature unforgeability and also achieves the private verification property we defined.

Theorem 3.1.1. *CL Scheme A above is a signature scheme with privacy.*

Proof. The original proof of signature protocol in [9] was shown to be zero-knowledge, meaning that no information about the original signature σ or the corresponding message m is revealed. We only modified the ZKPK statement to prove well-formedness of the commitment $com(m, r)$ (i.e., the fact that the prover knows m), which also reveals no information about m. Furthermore, when PrivVerify is executed multiple times using the same original signature σ, with the verifier learning no additional information.

□

To facilitate close comparison of different algorithms, we spell out the computation used in the ZKPK of PrivVerify above. This will allow us to determine the exact number of operations (such as modulo exponentiations and pairing function evaluations) needed. In this ZKPK, the prover first chooses random $v_1, v_2, v_3 \in \mathbb{Z}_q$, computes $T_1 = g^{v_1}h^{v_3}, T_2 = \mathbf{v}_{xy}^{v_1}\mathbf{v}_s^{v_2}$, and sends T_1, T_2 to the verifier. The verifier chooses a challenge $e \in \mathbb{Z}_q$ at random and sends it to the prover. The prover responds by sending $r_1 = v_1 + em \bmod q$, $r_2 = v_2 + er' \bmod q$, and $r_3 = v_3 + er \bmod q$. Finally, the verifier accepts if $g^{r_1}h^{r_3} = T_1 x_m^e$ and $\mathbf{v}_{xy}^{r_1}\mathbf{v}_s^{r_2} = T_2\mathbf{v}_x^{-e}$.

When certified inputs are used in SMC, we need to evaluate the time of signature verification and integration into an SMC protocol. Thus, we consider signature issuance as a one-time cost and concentrate on verification. Then to use this scheme with secure computation, the cost of (independent) private verification of n signatures is $3n$ modulo exponentiations (mod exp) for signature randomization, $2n$ mod exp for creating commitments (n of which are for messages and are thus short), and $10n$ mod exp and $5n$

pairings for proving the knowledge of signatures. This gives us $15n$ mod exp (n of which are short) and $5n$ pairings and serves as the baseline for comparison.

3.1.2 Modified CL Scheme A

We next introduce a simplification to CL Scheme A of the previous subsection to allow for more efficient private verification in the context of SMC. To construct a signature scheme with privacy $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrivVerify})$, we retain KeyGen and Sign algorithms of the previous subsection (i.e., the public key is augmented with h), but modify the verification algorithm $\mathsf{PrivVerify}$ as follows:

$\mathsf{PrivVerify}$**:** The prover has private message $m \in \mathbb{Z}_q$ and the corresponding signature $\sigma = (a, b, c) = (a, a^y, a^{x+mxy})$; both parties hold $pk = (q, G, \mathbf{G}, g, h, e, X, Y)$.

1. The prover forms a commitment to m as $x_m = com(m, r) = g^m h^r$ using randomly chosen $r \in \mathbb{Z}_q$ and sends x_m to the verifier.
2. The prover chooses random $r' \in \mathbb{Z}_q$, computes randomized signature $\tilde{\sigma} := (a, b, c^{r'}) = (a, b, \tilde{c})$, and communicates it to the verifier.
3. Let $\mathbf{v}_x = e(X, a)$, $\mathbf{v}_{xy} = e(X, b)$, and $\mathbf{v}_s = e(g, \tilde{c})$. The prover and verifier execute ZKPK: $PK\{(\mu, \rho, \gamma) : x_m = g^\mu h^\gamma \wedge \mathbf{v}_x^{-1} = \mathbf{v}_{xy}^\mu \mathbf{v}_s^\rho\}$.
4. If the verifier accepts the proof in step 3 and $e(a, Y) = e(g, b)$, output 1; otherwise, output 0.

In this verification, part of signature randomization is removed, which means that the verifier will be able to link two showings of the same signature together. This change,

however, does not affect the unforgeability property of the scheme. The privacy property can be stated as follows.

Theorem 3.1.2. *Modified CL Scheme A above is a signature scheme with privacy.*

Proof. Let $\mathcal{A}$ be a PPT adversary attacking new modified CL Scheme A. Recall that $\mathcal{A}$ has the ability to query the signing oracle and obtain signature on messages of its choice. Once $\mathcal{A}$ submits the challenge (m_0, m_1), it will be given pairs $(\tilde{\sigma}_b^{(i)}, x_{m_b}^{(j)})$, where b is a random bit, $\tilde{\sigma}_b^{(i)} = (a, a^y, a^{r_i'(x+m_b xy)}) = (a, b, \tilde{c}^{(i)})$ for random $r_i' \in \mathbb{Z}_q$, and $x_{m_b}^{(j)} = g^{m_b} h^{r_j}$ for random $r_j \in \mathbb{Z}_q$, for any combination of i and j and the number of queries polynomial in κ. In other words, $\mathcal{A}$ has access to a signature with different randomizations (using r_i's) and different commitments to m_b (using r_js for randomness).

Before we proceed with further analysis, note that the ZKPK in PrivVerify is zero-knowledge and thus does not reveal information about m_b to $\mathcal{A}$. Furthermore, other signatures on m_0 and m_1 that $\mathcal{A}$ can obtain using its access to the signing oracle doesn't contribute additional information (and use unrelated randomness) and thus do not help in answering the challenge. Therefore what remains is to analyze $\tilde{c}^{(i)}$ and $x_{m_b}^{(j)}$ values. Now note that each $\tilde{c}^{(i)}$ and $x_{m_b}^{(j)}$ are random elements in G because r_i', r_j are chosen uniformly and independently at random. This means that if we modify $\mathcal{A}$'s view to replace m_b in $\tilde{c}^{(i)}$s and $x_{m_b}^{(j)}$s with a random value, this modified view will be identically distributed to that of the original $\mathcal{A}$'s view. To expand on this, suppose that we modify the signature scheme to use a random value z instead of the actual message and the commitment is formed consistently to use the same z as well. Let's call the resulting scheme Π'. Clearly, we have that $\mathbf{Pr}[\mathsf{MesInd}_{\mathcal{A},\Pi'}(\kappa) = 1] = \frac{1}{2}$. Because the views of $\mathcal{A}$ are identical in the security

experiments for Π and Π', we obtain that $|\mathbf{Pr}[\mathsf{MesInd}_{\mathcal{A},\Pi'}(\kappa) = 1] - \mathbf{Pr}[\mathsf{MesInd}_{\mathcal{A},\Pi}(\kappa) = 1]| = 0$. This means that $\mathcal{A}$ cannot learn any information about m_b during verification in Π and the security property follows.

□

When we use this scheme for secure computation, we reduce the randomization cost by $2n$ mod exp. Thus the cost of private verification of n signatures is $13n$ mod exp (n of which are short) and $5n$ pairings.

3.1.3 Batch Verification of Modified CL Scheme A

The next step is to design batch verification for verifying n signatures. Because for this application we are primarily interested in verifying multiple signatures issued by the same signer (e.g., information about one's genome represented as a large number of individual values), we present batch verification of signatures issued using the same key. We use a version of the small exponent test [3] that instructs the verifier to choose security parameters l_b such that the probability of accepting a batch that contains an invalid signature is at most 2^{-l_b} (e.g., l_b is set to 60 or 80 in prior work).

Batch: The prover holds signatures $\sigma_i = (a_i, b_i, c_i)$ on messages $m_i \in \mathbb{Z}_q$ for $i = 1, \ldots, n$, and both parties hold $pk = (q, G, \mathbf{G}, g, h, e, X, Y)$.

1. The prover forms commitments $x_{m_i} = com(m_i, r_i) = g^{m_i} h^{r_i}$ using randomly chosen $r_i \in \mathbb{Z}_q$ for $i = 1, \ldots, n$ and sends them to the verifier.
2. The prover chooses random $r'_i \in \mathbb{Z}_q$, computes blinded signatures $\tilde{\sigma}_i = (a_i, b_i, c_i^{r'_i}) = (a_i, b_i, \tilde{c}_i)$ for $i = 1, \ldots, n$, and sends them to the verifier.

3. The verifier chooses and sends random $\delta_1, \ldots, \delta_n \in \{0,1\}^{l_b}$ to the prover.

4. The parties compute $\hat{\mathbf{v}}_x = e(X, \prod_{i=1}^n a_i^{\delta_i})$, $\hat{\mathbf{v}}_{xy_i} = e(X, b_i^{\delta_i})$ and $\hat{\mathbf{v}}_{s_i} = e(g, \tilde{c_i}^{\delta_i})$ for $i = 1, \ldots, n$, and engage in the ZKPK: $PK\{(\mu_1, \ldots, \mu_n, \rho_1, \ldots, \rho_n, \gamma_1, \ldots, \gamma_n) : \hat{\mathbf{v}}_x^{-1} = \prod_{i=1}^n \hat{\mathbf{v}}_{xy_i}^{\mu_i} \hat{\mathbf{v}}_{s_i}^{\rho_i} \wedge x_{m_1} = g^{\mu_1} h^{\gamma_1} \wedge \cdots \wedge x_{m_n} = g^{\mu_n} h^{\gamma_n}\}$.

5. If this proof passes and $e(\prod_{i=1}^n a_i^{\delta_i}, Y) = e(g, \prod_{i=1}^n b_i^{\delta_i})$, the verifier outputs 1; otherwise, the verifier outputs 0.

Theorem 3.1.3. Batch *above is a batch verifier for Modified CL Scheme A.*

Proof. First, we show that success of PrivVerify on (pk, m_i, σ_i) for all $i \in [1, n]$ implies that Batch also outputs 1 on $pk, (m_1, \sigma_1), \ldots, (m_n, \sigma_n)$. When all PrivVerify output 1, for each $i = 1, \ldots, n$ $\mathbf{v}_x^{-1} = \mathbf{v}_{xy}^{m_i} \mathbf{v}_s^{r_i}$, which is expanded as

$$e(X, a_i)^{-1} = e(X, b_i)^{m_i} \cdot e(g, \tilde{c_i})^{r_i}$$

Then $e(X, a_i^{\delta_i})^{-1} = e(X, b_i^{\delta_i})^{m_i} \cdot e(g, \tilde{c_i}^{\delta_i})^{r_i}$ for all i and consequently

$$\prod_{i=1}^n e(X, a_i^{\delta_i})^{-1} = \prod_{i=1}^n e(X, b_i^{\delta_i})^{m_i} e(g, \tilde{c_i}^{\delta_i})^{r_i}$$

Because $\prod_{i=1}^n e(X, a_i^{\delta_i}) = e(X, \prod_{i=1}^n a_i^{\delta_i})$, we obtain equivalence with

$$\hat{\mathbf{v}}_x^{-1} = \prod_{i=1}^n \hat{\mathbf{v}}_{xy_i}^{m_i} \hat{\mathbf{v}}_{s_i}^{r_i}$$

To show the other direction, assume that Batch accepts. We know that $\tilde{c_i}, a_i, b_i \in G$,

thus $\tilde{c}_i = g^{\gamma_i}, a_i = g^{s_i}, b_i = g^{t_i}$ for some $\gamma_i, a_i, b_i \in \mathbb{Z}_q$. Then

$$\begin{aligned}
\hat{\mathbf{v}}_x^{-1} &= \prod_{i=1}^{n} \hat{\mathbf{v}}_{xy_i}^{m_i} \hat{\mathbf{v}}_{s_i}^{r_i} = \prod_{i=1}^{n} e(X, b_i^{\delta_i})^{m_i} \cdot e(g, \tilde{c}_i^{\delta_i})^{r_i} \\
&= \prod_{i=1}^{n} e(X, g^{t_i \delta_i})^{m_i} \cdot e(g, g^{\gamma_i \delta_i})^{r_i} \\
&= \prod_{i=1}^{n} e(g, g)^{x t_i \delta_i m_i} \cdot e(g, g)^{\gamma_i \delta_i r_i} \\
&= \prod_{i=1}^{n} e(g, g)^{\delta_i (x t_i m_i + \gamma_i r_i)}
\end{aligned}$$

Because

$$\hat{\mathbf{v}}_x^{-1} = e(X, \prod_{i=1}^{n} a_i^{\delta_i})^{-1} = e(g^x, \prod_{i=1}^{n} g^{s_i \delta_i})^{-1} = \prod_{i=1}^{n} e(g, g)^{-x s_i \delta_i}$$

so

$$e(g, g)^{-x \sum_i s_i \delta_i} = e(g, g)^{\sum_i \delta_i (x t_i m_i + \gamma_i r_i)}$$

and consequently

$$\sum_i x s_i \delta_i + \sum_i \delta_i (x t_i m_i + \gamma_i r_i) \equiv 0 \pmod{q}$$

Let us set $\beta_i = x(s_i + t_i m_i) + \gamma_i r_i$, then

$$\sum\nolimits_{i=1}^{n} \delta_i \beta_i \equiv 0 \pmod{q} \tag{3.1.1}$$

Now suppose that Batch returned output 1, while for at least one i PrivVerify returns 0 on the corresponding input (pk, m_i, σ_i). Without loss of generality, let $i = 1$. This means that $e(X, a_1)^{-1} \neq e(X, b_1)^{m_1} \cdot e(g, \tilde{c}_1)^{r_1}$ and consequently $\beta_1 = x(s_1 + t_1 m_1) + \gamma_1 r_1 \neq 0$.

Because G and $\mathbf{G}$ are cyclic groups of prime order q, β_1 has an inverse α_1 such that

$$\beta_1 \alpha_1 \equiv 1 \pmod{q}.$$

We re-write equation (3.1.1) as $\delta_1\beta_1 + \sum_{i=2}^{n} \delta_i\beta_i \equiv 0 \pmod{q}$, and substitute β_1 with α_1^{-1} to obtain $\delta_1\alpha_1^{-1} + \sum_{i=2}^{n} \delta_i\beta_i \equiv 0 \pmod{q}$. This gives us

$$\delta_1 \equiv -\alpha_1 \sum_{i=2}^{n} \delta_i\beta_i \pmod{q} \tag{3.1.2}$$

Let E be an event such that $\mathsf{PrivVerify}(pk, m_1, \sigma_1) = 0$, but $\mathsf{Batch}(pk, (m_1, \sigma_1), \ldots, (m_n, \sigma_n)) = 1$. Also, let vector $\triangle = (\delta_2, \ldots, \delta_n)$ and $|\triangle|$ denote the number of possible values of $\triangle$. By equation (3.1.2), when $\triangle$ is fixed, there exists only one value of δ_1 that results in event E happening. In other words, for a fixed $\triangle$ the probability of E given a randomly chosen δ_1 is $\Pr[\mathrm{E} \,|\, \triangle'] = 2^{-l_b}$. Thus, we bound the probability of E for randomly chosen δ_1 by summing over all possible choices of $\triangle$, i.e., $\Pr[\mathrm{E}] \leq \sum_{i=1}^{|\triangle|}(\Pr[\mathrm{E} \,|\, \triangle] \cdot \Pr[\triangle])$. We obtain $\Pr[\mathrm{E}] \leq \sum_{i=1}^{2^{l_b(n-1)}}(2^{-l_b} \cdot 2^{-l_b(n-1)}) = \sum_{i=1}^{2^{l_b(n-1)}}(2^{-l_b n}) = 2^{-l_b}$.

□

As before, we spell out the ZKPK computation in the Batch protocol: The prover chooses random $v_i, v_i', v_i'' \in \mathbb{Z}_q$ and computes $\mathrm{T}_i = g^{v_i} h^{v_i''}$ for $i = [1, n]$ as well as $\mathrm{T} = \prod_{i=1}^{n}(\mathbf{v}_{xy_i}^{v_i} \mathbf{v}_{s_i}^{v_i'})$, and sends T_is and T to the verifier. After receiving challenge $e \in \mathbb{Z}_q$ from the verifier, the prover responds with $u_i = v_i + em_i \bmod q$, $u_i' = v_i' + er_i' \bmod q$, and $u_i'' = v_i'' + er_i \bmod q$ for all i. The verifier accepts if $g^{u_i} h^{u_i''} = \mathrm{T}_i x_{m_i}^{e}$ for $i = 1, \ldots, n$ and $\prod_{i=1}^{n}(\mathbf{v}_{xy_i}^{u_i} \mathbf{v}_{s_i}^{u_i'}) = \mathrm{T}\mathbf{v}_x^{-e}$.

The cost of using this construction for n certified inputs in SMC is n mod exp for signature randomization, $2n$ mod exp for creating commitments (n of which are short), $12n+1$ mod exp ($3n$ of which are short) and $2n+3$ pairings for the ZKPK. This gives us $15n+1$ mod exp ($4n$ of which are short) and $2n+3$ pairings, significantly reducing the number of pairing operations, which we consider to be the costliest operation, compared to private verification of individual messages.

Because of the way inputs are entered in the computation for the SMC constructions considered in Section 3.3, a single commitment to all inputs of a participant is permissible. Thus, instead of using separate commitments for each m_i, we could form a single commitment to n messages $com(m_1, \ldots, m_n, r) = g_1^{m_1} \cdots g_n^{m_n} h^r$ and modify the ZKPK to use it instead of the individual commitments. This reduces the cost of forming commitments to n short and one regular mod exp, and the cost of ZKPK is reduced by $3n-3$ mod exp (i.e., only one v'' needs to be formed and we compute only one T_i instead of n of them). This gives us the total of $11n+5$ mod exp ($4n$ of which are short) and $2n+3$ pairings.

3.2 Construction based on ElGamal Signature

In this section, we show how to modify (provably secure) ElGamal signature schemes to achieve private verification and consequently provide a batch verifier for the resulting construction.

3.2.1 Modified ElGamal Scheme

The starting point was provably secure ElGamal [40] described in section 2.2.2. To enable private verification, the idea is to use signatures on commitments to messages instead of on

messages themselves. We also modify the setup to work in a group of prime order q, i.e., a subgroup of $\mathbb{Z}_p^*$, instead of entire $\mathbb{Z}_p^*$. This simplifies the design and opens up additional possibilities, without compromising security guarantees. In particular, the small exponent test used for batch verification is not applicable to groups of non-prime order [3]. The signature scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrivVerify})$ is given as:

KeyGen: On input a security parameter 1^κ, choose a group G of large prime order q and its generator g. Then choose random $x, u \in \mathbb{Z}_q$ and compute $y = g^x$ and $h = g^u$. Set $sk = x$, $pk = (q, G, g, y, h)$.

Sign: On input message m, secret key $sk = x$ and public key $pk = (q, G, g, y, h)$, choose random $k, r \in \mathbb{Z}_q$ and compute $t = g^k$, $x_m = com(m, r) = g^m h^r$, and $s \equiv (H(x_m||t) - xt)k^{-1} \pmod q$. The algorithm outputs $\sigma = (t, s)$ and $x_\sigma = r$. The recipient computes $com(m, x_\sigma)$ and verifies the signature on $com(m, x_\sigma)$.

PrivVerify: The prover has private m and x_σ, the corresponding signature $\sigma = (t, s)$ on x_m, where $x_m = g^m h^{x_\sigma}$, and both parties hold pk. The prover gives the verifier σ and x_m and they engage in the following ZKPK: $PK\{(\mu, \gamma) : x_m = g^\mu h^\gamma\}$. If this proof passes and the equality $g^{H(x_m||t)} = y^t t^s$ holds, the verifier outputs 1; otherwise, the verifier outputs 0.

Note, that in this scheme the signer chooses g, h and thus will be able to open a commitment $com(m, r)$ to a message different from m (but the users will not be able to do so). If this poses a security risk, h will need to be produced by an independent party or parties so that the signer does not know the discrete logarithm of h to the base g.

This signature scheme remains unforgeable, and we prove it using the standard definition (Definition 2.2.2) with ForgePrivSig experiment that accommodates privacy as described in section 2.1. The intuition is that the prover now has a signature on a commitment, but has to demonstrate knowledge of the commitment opening (i.e., the message itself) and the use of groups of prime order only simplifies the analysis in [40].

Theorem 3.2.1. *Modified ElGamal signature scheme is existentially unforgeable against an adaptive chosen-message attack in a random oracle model.*

Proof. The ElGamal signature scheme on which we build [40, 41] was shown to be secure in the random oracle model assuming α-hard prime moduli. This assumption is satisfied by groups of prime order which the modification uses.

The proof in [41] proceeds in two steps: First, security against a no-message attack is shown. Second, it is shown that the signer can be simulated with an indistinguishable distribution, which using the forking lemma implies security against an adaptively chosen-message attack. If we examine the proof of the first step, we can see that two cases are considered: t is prime to q mod $p-1$ and t is not prime to q mod $p-1$, where $\sigma = (t, s)$. In this case, the second option does not exist as the computation is mod q, and thus the rest of the analysis goes through with only one option to consider. The second part of the proof in [41] can also be simplified as we only need to consider computation modulo q in the exponent, as opposed to computation modulo $qR = p - 1$. Thus, it is easier to show security when the setup assumes groups of prime order.

As far as the modification to replace message m with a commitment to m x_m goes, x_m can be treated as m in the proof above. Furthermore, because H is modeled as a

random oracle, there is flexibility with respect to the values to which its output can be set. The verification process also requires that the prover proves the knowledge of the discrete logarithm representation of x_m, which means that the prover must know m upon which x_m was formed (because the protocol is a proof of knowledge). This also means that $\mathcal{A}$ will be able to correctly prove (in zero-knowledge) that m included in x_m for every message that $\mathcal{A}$ used to query the signing oracle is different from the messages m^* output during its forgery, as required by the security definition.

□

Next, we state the privacy property of modified ElGamal signature.

Theorem 3.2.2. *Modified ElGamal scheme is a signature scheme with privacy.*

Proof. Let $\mathcal{A}$ be a PPT adversary attacking the modified ElGamal signature scheme with access to the signing oracle. After $\mathcal{A}$ submits the challenge (m_0, m_1), it receives (σ_b, x_{m_b}), where $x_{m_b} = com(m_b, r)$ for some random private r and σ_b is an ElGamal signature on x_{m_b}. Note that in this scheme, that the signature σ_b and commitment x_{m_b} that $\mathcal{A}$ observes are fixed, meaning that even if $\mathcal{A}$ executes multiple instances of PrivVerify. It will still observe the same values of σ_b and x_{m_b} and only the execution traces of the ZKPK can differ. Based on the properties of the underlying building blocks, we know that the ZKPK is zero-knowledge and the commitment scheme is information-theoretically hiding, which means that $\mathcal{A}$ cannot have a non-negligible advantage in determining any information about m_b (assuming the difficulty of the discrete logarithm problem). In other words, if we replace m_b in the challenge with a randomly chosen message (in which case $\mathcal{A}$ cannot

do better than a random guess), the execution trace of PrivVerify will be indistinguishable to that of the actual challenge.

Recall that $\mathcal{A}$ has access to the signing oracle and can obtain its own signatures on m_0 and m_1 prior to the challenge phase. We note any such previously issued signatures on m_0 and m_1 cannot help $\mathcal{A}$ to answer the challenge because $\mathcal{A}$ observes only a commitment to m_b which information-theoretically hides and new randomness is used for each new signature.

□

The ZKPK in this PrivVerify proceeds similar to previous ZK proofs, where the prover chooses $v_1, v_2 \in \mathbb{Z}_q$, computes $T = g^{v_1} h^{v_2}$, and sends T to the verifier. After receiving the challenge e from the verifier, the prover responds by sending $r_1 = v_1 + em \bmod q$, $r_2 = v_2 + er \bmod q$, and the verifier accepts if $g^{r_1} h^{r_2} = x_m^e T$. The cost of using this construction in SMC is 5 mod exp for the ZKPK and 3 for signature verification, giving us a total of 8 mod exp. (If the user does not store commitment x_m, its re-computation is another 1 regular and 1 short mod exp.)

3.2.2 Batch Verification of Modified ElGamal Signatures

The batch verifier for the modified ElGamal signature is given next. It uses the same security parameter l_b as before.

Batch: The prover holds commitments $x_{m_i} = com(m_i, x_{\sigma i}) = g^{m_i} h^{x_{\sigma i}}$ on messages $m_i \in \mathbb{Z}_q$ using randomness $x_{\sigma_i} \in \mathbb{Z}_q$ and signatures $\sigma_i = (t_i, s_i)$ on x_{m_i} for $i = 1, \ldots, n$. Both parties hold $pk = (q, G, g, y, h)$.

1. The prover sends signatures σ_i and commitments x_{m_i} to the verifier for $i = 1, \ldots, n$.

2. The prover and verifier engage in the following ZKPK: $PK\{(\mu_1, \ldots, \mu_n, \gamma_1, \ldots, \gamma_n) : x_{m_1} = g^{\mu_1} h^{\gamma_1} \wedge \cdots \wedge x_{m_n} = g^{\mu_n} h^{\gamma_n}\}$. If the proof fails, the verifier outputs 0 and aborts.

3. The verifier chooses $\delta_1, \ldots, \delta_n \in \{0,1\}^{l_b}$ at random, computes $u_1 = \sum_{i=1}^{n} H(x_{m_i}||t_i)\delta_i$ and $u_2 = \sum_{i=1}^{n} t_i \delta_i$, and checks whether $g^{u_1} = y^{u_2} \prod_{i=1}^{n} t_i^{s_i \delta_i}$. If the check succeeds, the verifier outputs 1, and 0 otherwise.

Theorem 3.2.3. Batch *above is a batch verifier for the modified ElGamal scheme.*

Proof. First we show that $\mathsf{PrivVerify}(pk, m_1, \sigma_1) = \ldots = \mathsf{PrivVerify}(pk,\ m_n, \sigma_n) = 1$ implying that $\mathsf{Batch}\left(pk, (m_1, \sigma_1), \ldots, (m_n, \sigma_n)\right) = 1$. Suppose that the individual signatures verified, i.e., $g^{H(x_{m_i}||t_i)} = y^{t_i} t_i^{s_i}$ for all $i = 1, \ldots, n$. Then

$$\begin{aligned} g^{u_1} &= g^{\sum_{i=1}^{n} H(x_{m_i}||t_i)\delta_i} = \prod_{i=1}^{n} \left(g^{H(x_{m_i}||t_i)}\right)^{\delta_i} \\ &= \prod_{i=1}^{n} \left(y^{t_i} t_i^{s_i}\right)^{\delta_i} = \prod_{i=1}^{n} y^{t_i \delta_i} \prod_{i=1}^{n} t_i^{s_i \delta_i} \\ &= y^{\sum_{i=1}^{n} t_i \delta_i} \prod_{i=1}^{n} t_i^{s_i \delta_i} = y^{u_2} \prod_{i=1}^{n} t_i^{s_i \delta_i} \end{aligned}$$

is deduced as desired. To show the other direction, assume that Batch accepts. We know that each t_i was computed as $t_i = g^{k_i}$ for some $k_i \in \mathbb{Z}_q$, thus we can write:

$$g^{\sum_{i=1}^{n} H(x_{m_i}||t_i)\delta_i} = y^{\sum_{i=1}^{n} t_i \delta_i} \prod_{i=1}^{n} t_i^{s_i \delta_i} = g^{\sum_{i=1}^{n} x_i t_i \delta_i} \prod_{i=1}^{n} g^{k_i s_i \delta_i} = g^{\sum_{i=1}^{n} x_i t_i \delta_i + \sum_{i=1}^{n} k_i s_i \delta_i}$$

It follows that

$$\sum_{i=1}^{n} H(x_{m_i}||t_i)\delta_i - \sum_{i=1}^{n} x_i t_i \delta_i - \sum_{i=1}^{n} k_i s_i \delta_i \equiv 0 \pmod{q} \tag{3.2.1}$$

Let $\beta_i = H(x_{m_i}||t_i) - x_i t_i - k_i s_i$. Then equation (3.2.1) can be written as $\sum_{i=1}^{n} \delta_i \beta_i \equiv 0 \pmod{q}$, which is the same as equation (3.1.1) in the proof of Theorem 3.1.3. Thus, the remainder of the proof proceeds in the same way as the proof of Theorem 3.1.3 to obtain that the probability of Batch successfully completing when at least one signature does not verify is at most 2^{-l_b}.

□

The ZKPK in Batch above consists of n invocations of the ZKPK in modified ElGamal's PrivVerify. Thus, the cost of batch verification of n messages is $5n$ mod exp for the ZKPK and $n + 2$ mod exp for signature verification, or $6n + 2$ mod exp total. (If the commitments are to be re-computed, we add n regular and n short mode exp.)

Taking into account the way messages are input into SMC allows us to use a single commitment to all n messages. For the modified ElGamal, this optimization results in great savings because this means that we can use a single signature. Thus, the signer now issues a signature on $x_m = com(m_1, \ldots, m_n, r)$ and x_σ still contains the randomness r. This significantly simplifies the Batch algorithm above as only 1 signature and 1 commitment are communicated in step 1, with step 2 only involving the proof of knowledge of the discrete logarithm representation of x_m. Lastly step 3 consists of verifying a single signature without the use of δ_is. This has significant performance improvement implications, with the cost of step 2 reduced to $2n + 3$ mod exp and the overall cost of Batch

Scheme	Single message	Batch with n commitments	Batch with 1 commitment
Modified CL Scheme A	11 mod exp and 5 pairings	$10n + 1$ regular and $3n$ short mod exp and $2n + 3$ pairings	$6n + 5$ regular and $3n$ short mod exp $2n + 3$ pairings
Modified ElGamal	8 mod exp	$6n + 2$ mod exp	$2n + 6$ mod exp

Table 3.1: Performance of private verification for a single signature and a batch of size n.

reduced to $2n + 6$ mod exp (again, if the commitment is to be re-computed, we add 1 regular and n short mod exp).

The performance of new constructions of signature and batch verification can be found in Table 3.1. It is assumed that commitments are stored pre-computed.

3.3 Using Certified Inputs in Secure Computation

Having described the private verification protocols, we now address the question of integrating them with SMC techniques based on secret sharing in the presence of malicious adversaries. For that purpose, we have chosen two prominent constructions of Damgård and Nielsen [15] and SPDZ [16] and provide discourse on their work. These constructions were chosen based on their attractive performances and distinct security guarantees that they provide, including cases such as when the computation is performed by k parties. The former solution tolerates fewer than $k/3$ corruptions, while the latter can handle any number of corrupt parties. To compare, our solution uses signatures with privacy to guarantee that inputs entered into secure computation are identical to those generated or observed by an authority. However, in general certification can take on different forms.

As far as security properties go, the privacy guarantees of SMC in the presence of malicious adversaries must hold as in the standard formulation of the problem (see, e.g.,

[20] for a formal definition). We additionally require the condition that it is not feasible for a participant to enter (certified) inputs into the computation without possessing a signature on them. More formally, if a participant supplying input x enters a value different from what was certified by a certification authority, this behavior will be detected by the participants with overwhelming probability. In other words, if the computation completes successfully, there is only a negligible chance that a corrupt input owner can enter an input that has not be signed by the certification authority.

In what follows, we denote the computational parties as $P_1, \ldots, P_k$ and assume that they are connected by pairwise secure channels. These constructions use (k, t)-threshold linear secret sharing, and we denote a secret shared version of x by $[x]$.

3.3.1 Damgård-Nielsen Scalable and Unconditionally SMC

The construction of Damgård-Nielsen [15] is unconditionally secure (assuming secure channels) in the presence of at most $t < k/3$ malicious participants. It was the first model to achieve unconditional security with communication complexity where the part that depends on the circuit size is only linear in k. The computation proceeds in two stages: offline pre-computation that generates random multiplication triples/other random values and the online phase which is executed once the inputs become available.

As far as input into the computation (during the online phase) goes, let x denote party P_ℓ's input into the secure computation for some ℓ (the same will apply to all other parties holding inputs; participants with input who are not computational parties can be accommodated as well). To secret-share x among the parties, P_ℓ computes $\delta = x + r$, where r is a random value chosen during precomputation in such a way that the parties

hold shares of r $[r]$ and the value of r is known in the clear to P_ℓ (i.e., $[r]$ was opened to P_ℓ). Both the shares $[r]$ and the value r that P_ℓ possesses are guaranteed to be correct in the presence of malicious participants. Then once P_ℓ computes δ, P_ℓ broadcasts it to all parties who compute $[x] = \delta - [r]$ and use $[x]$ in consecutive computation.

To enable the use of certified inputs, we need to modify the above input sharing procedure to guarantee that x that P_ℓ uses in computing δ was indeed certified. Then to ensure that the correct x is input into the computation, the parties could compute a commitment to r and verify (in zero-knowledge) that δ corresponds to the sum of r and x. This could be implemented by having the parties broadcast commitments to their shares of r and interpolating them to compute a commitment to r. In that case, reconstructing a reliable commitment to r presents the main challenge because any participant can be malicious. If the input owner P_ℓ is honest, it can verify correctness of commitments from other parties and discard incorrect transmissions. Dealing with malicious P_ℓ, however, is more difficult because P_ℓ can influence through its share the value of r in the commitment which the parties reconstruct. In this case, because the validity of P_ℓ's share cannot be verified, P_ℓ can adjust its share to modify the reconstructed r by the amount it wants to change x from its certified version, getting around the certification process.

To solve this issue, we chose to proceed with directly entering input x into the computation as opposed to supplying the delta. To accomplish this, we utilize one of the building blocks from [15] for dealing consistent shares of a value (which is input x in our case). It has a mechanism for resolving conflicts and upon successful termination provides a set of parties holding consistent shares. We use this set to form a commitments to shares of x and interpolate them to reconstruct a commitment to x.

Because each P_ℓ often enters multiple inputs into the computation, we will associate inputs $x_1, \ldots, x_n$ with party P_ℓ. In what follows, we describe our version with a single commitment to all x_is, allowing for improved performance. The case of a single certified input x will also follow from that construction. When P_ℓ's inputs are certified by multiple authorities, this procedure is performed for each public key separately. Because the solution uses (Pedersen) commitments, we assume that a group setup (G, q) where the discrete logarithm problem is hard with generators $g_1, \ldots, g_n, h$ is available to the parties. All signature schemes that we considered in this work already use commitments, and therefore we will assume that this setup comes from the public key of the corresponding signature scheme.

We use notation $[y]_j$ to denote the jth share of y held by party P_j. As in [15], we assume that operations on secret shares take place in a field $\mathbb{F}$ and secret shares correspond to the evaluation of a polynomial of degree t on different points. For concreteness, we set $\mathbb{F} = \mathbb{F}_p$ for a prime p ($q \gg p$). The computation encompasses:

Input: The parties collectively hold $[r_1], \ldots, [r_n]$ and the public key pk of the certification authority. P_ℓ has private input $x_1, \ldots, x_n$, $com(x_1, \ldots, x_n, \hat{r})$, and signatures with privacy $\sigma_1, \ldots, \sigma_n$ on $x_1, \ldots, x_n$, respectively.[1]

Output: $[x_1], \ldots, [x_n]$ are available to the parties and their certification has been verified.

1. The parties execute the protocol for P_ℓ to deal consistent shares of $x_1, \ldots, x_n$ and another value α that P_ℓ randomly chooses from $\mathbb{F}_p$ (as specified in Figure 7 from [15]). If P_ℓ is honest, there are at least $2t+1$ parties who hold consistent shares of each x_i

[1] Note that in the case of the modified ElGamal signatures, P_ℓ will hold a single signature on $com(x_1, \ldots, x_n, \hat{r})$.

and we denote this set by S. (Otherwise, the protocol fails and the parties restart it as specified in [15].)

2. P_ℓ broadcasts commitments $com([x_1]_j, \ldots, [x_n]_j, [\alpha]_j) = g_1^{[x_1]_j} \cdots g_n^{[x_n]_j} h^{[\alpha]_j}$ and each $P_j \in S$ verifies that the jth commitment is consistent with its shares.

3. The parties compute interpolation coefficients β_j (in $\mathbb{F}_p$) for each $P_j \in S$ and then compute $c'_x = com(x'_1, \ldots, x'_n, \alpha') = \prod_{P_j \in S} com([x_1]_j, \ldots, [x_n]_j, \alpha_j)^{\beta_j}$. Note that $x_i = \sum_{P_j \in S} \beta_j [x_i]_j$ (in $\mathbb{F}_p$) for each i.

4. P_ℓ computes $\alpha' = \sum_{P_j \in S} \beta_j [\alpha]_j$ (in $\mathbb{Z}_q$) and $x'_i = \sum_{P_j \in S} \beta_j [r_i]_j$, $s_i = \lfloor x'_i / p \rfloor$ (over integers) for $i = 1, \ldots, n$. It creates commitment $c_s = com(s_1, \ldots, s_n, \tilde{r}) = g_1^{s_1} \cdots g_n^{s_n} h^{\tilde{r}}$ and broadcasts it to the other parties.

5. P_ℓ broadcasts $c_x, \sigma_1, \ldots, \sigma_n$ and the parties execute $\mathsf{Batch}(pk, x_1, \sigma_1, \ldots, x_n, \sigma_n)$ with P_ℓ playing the role of the prover.

6. The parties additionally execute $\mathrm{PK}\{(x_1, \ldots, x_n, x'_1, \ldots, x'_n, s_1, \ldots, s_n, \alpha', \hat{r}, \tilde{r}) : c_x = g_1^{x_1} \cdots g_n^{x_n} h^{\hat{r}} \wedge c'_x = g_1^{x'_1} \cdots g_n^{x'_n} h^{\alpha'} \wedge c_s = g_1^{s_1} \cdots g_n^{s_n} h^{\tilde{r}} \wedge \bigwedge_{i=1}^{n} \left(x'_i = x_i + s_i p\right)\}$ where P_ℓ plays the role of the prover.

Because different moduli are used for exponents in G and arithmetic in $\mathbb{F}_p$, to guarantee correctness we need to compensate for reduction modulo p in field operations. To accomplish that, we interpolate each x_i over integers and thus have that $x'_i = x_i + s_i p$ for some unique integer s_i, which is the relationship that P_ℓ proves in step 6. This computation requires that $|q| > 2t|p|$, which is the case in practice for typical values of q, t, and p (i.e.,

threshold t is usually low and set to 1–2, $|p|$ is set to accommodate integers of 64 or fewer bits, and $|q|$ is at least in hundreds to guarantee security).

Lastly, note that step 6 already includes a PK of the discrete logarithm representation of $com(x_1, \ldots, x_n, \hat{r})$ and thus the same ZKPK in Batch is no longer executed in step 6.

To show security, we prove that this modification complies with the definition of secure multi-party computation and it is not feasible for a dishonest participant to supply inputs different from what was signed by a certification authority.

Theorem 3.3.1. *Assuming security of Pedersen commitment,* Batch *is a batch verifier for a signature scheme with privacy, and the proof of knowledge is zero-knowledge, our modification to the Damgård-Nielsen construction above is a t-secure multiparty protocol for $t < k/3$.*

Proof (Sketch). In the context of our problem, we treat c_x, $\sigma_1, \ldots, \sigma_n$ as public values accessible to the adversary in both ideal and real models. This may be of particular importance when the same certification is used in multiple secure function evaluations, possibly with a different set of participants and may be observable by the adversary. We consider two cases: 1) P_ℓ is not among the corrupted parties and 2) P_ℓ is among the corrupted parties.

Case 1. When P_ℓ is not among the corrupt parties, the simulator is unable to obtain access to P_ℓ's input and simulates the adversarial view on randomly chosen data. In particular, the simulator uses randomly chosen values in place of $x_1, \ldots, x_n$ in step 1 and the parties hold shares of these values at the end of step 1. In step 2, the simulator forms commitments on behalf of P_ℓ consistent with the shares generated in step 1, and each

party computes c'_x in step 3. Step 5 uses true $c_x, \sigma_1, \ldots, \sigma_n$ (recall that no ZKPKs are executed in that step). To carry out the ZKPK in step 6, the simulator can compute all values and the commitment in step 4 based on the information it used in earlier steps of the protocol (i.e., shares $[x_i]_j$ produced in step 1) and needs to invoke the ZKPK simulator in step 6 to simulate the verifier view. Once the computation completes, the simulator sets the shares of the output that the corrupt parties receive as in the original protocol, so that they re-assemble to the output the parties are entitled to learn.

This simulation achieves indistinguishability because secret shares and commitments are perfectly hiding and reveal no information about the values they encode (and thus the adversary is unable to tell that randomly chosen inputs are used to generate shares and commitments that use the shares), signature verification maintains privacy of the inputs, and the ZKPK reveals no information about the values used in its statement as well. Finally, the computation on secret-shared data that follows maintains security guarantees as well.

Case 2. In this case, the honest parties whose participation the simulator is to simulate contribute no input. Therefore, the simulator simply follows the protocol the way honest participants would.

□

Observe that the same signature (and possibly the same commitment to the signed message) can be used in multiple secure function evaluations for possibly different functions. To capture this formally, one would need to modify the standard definition to allow for the participants to evaluate multiple functions with the same observable information

associated with an input (i.e., signatures and commitments in our case). Here we only note that our construction remains secure in those circumstances as well. This is because the modification uses perfectly hiding Pedersen commitments, zero-knowledge proofs, and signatures that provably protect the messages being signed. If the commitment associated with a signature does not change across different secure function evaluations, the steps above (for entering certified inputs into secure computation) can be executed only once for multiple invocations of secure multi-party computation (with possibly different functions). Otherwise, the steps above can be executed using fresh randomness for the commitments, still maintaining privacy of the inputs.

Theorem 3.3.2. *If the computation above does not abort, PK is a proof of knowledge,* Batch *is unforgeable, and commitments are binding, a dishonest P_ℓ can enter $x'_i \neq x_i$ for at least one $i \in [1, n]$ with at most negligible probability.*

Proof. First of all, because we only consider computation that could successfully complete, the checks performed in steps 2, 5, and 6 must hold. Second, because of the properties of the signature scheme, commitment c_x has to be on true input $x_1, \ldots, x_n$ with all but negligible probability to pass signature verification. We also have that ZKPK is secure, which means that the relationship $x'_i = x_i + s_i p$ must hold in step 6 for some integer s_i, which means that x'_i and x_i are equivalent for the purposes of the computation that follows. Therefore, it remains to show that it is not feasible to tamper with the values that lead to the computation of x'_is.

Next, based on the properties of the original construction, we have that the parties in S collectively hold consistent shares of the inputs entered by P_ℓ. The most crucial

step here is to demonstrate that transition from shares to commitments does not let P_ℓ modify the values that others view as its inputs in this process. Then if dishonest P_ℓ broadcasts commitments inconsistent with the shares $[x_i]_j$ of some party $P_j \in S$, P_ℓ's behavior will be detected. On the other hand, if dishonest $P_j \in S$ claims that P_ℓ did not send a correct commitment, the dispute can be resolved as in the original solution with dispute resolution by possibly eliminating some parties from S (but the number of honest parties is guaranteed to be at least $2t+1$, which allows for successful dispute resolution). We obtain that if step 2 successfully completes, the shares must re-assemble to x'_i over integers, which has identical meaning to x_i in $\mathbb{F}_p$. Therefore, the computation that follows proceeds on consistent shares of x_is and P_ℓ must possess signatures on x_i or $x_i + z_i p$ for some $z_i \in \mathbb{Z}$ (which have identical meaning when computing in $\mathbb{F}_p$).

□

3.3.2 SPDZ

The second of our proposed solutions is built on SPDZ [16]. This is an SMC protocol that achieves security in the presence of any number of malicious parties $t < k$ (and thus offers stronger security guarantees than the previous solution) and has a fast online phase. This construction enters private inputs into the computation similar to the way [15] did. For example, to secret share input x_i, the input owner P_ℓ uses a random value r_i computed during the preprocessing phase known only to P_ℓ and the parties jointly holding $[r_i]$. P_ℓ then computes and broadcasts $\delta_i = x_i - r_i$ (in $\mathbb{F}_p$) and the players compute $[x_i] = [r_i] + \delta_i$. The difference is that now additive secret sharing (i.e., $(k-1)$-out-of-k) is used instead of threshold secret sharing, with each secret-shared value y also using a secret-shared MAC

$\gamma(y)$ in the form of $\alpha(y+\tau)$, where α is a global secret key and τ is public, to authenticate its value. Simply put, a secret shared value $[y]$ is represented by each party P_i holding $\langle \tau, [y]_i, [\gamma(y)]_i \rangle$, where $[y]_1 + \ldots + [y]_n = y$ and $[\gamma(y)]_1 + \ldots + [\gamma(y)]_k = \alpha(y + \tau)$. The value of α is opened at the end of secure computation and is used to verify consistency of certain values used during computation, before the parties can learn the result (see [16] for detail).

Unlike our previous solution considered in section 3.3.1, we could proceed with the approach where the parties compute the input as $x_i = r_i + \delta_i$, reconstruct a commitment to r_i, and use it to verify the relationship between x_i and r_i. Verification of correct r_i used in the commitment is then deferred to the end of the computation where the value of α is opened. If the parties determine that the commitment to r_i was correctly formed, they proceed with reconstructing the output. In order for our security analysis to go through, we require that the field size is sufficiently large so that the probability $1/|\mathbb{F}_p|$ can be considered to be negligible. This assumption is already present in SPDZ itself, which states that the field size must be large to have the desired error probability of $(1/|\mathbb{F}_p|)^c$ for a small constant c.

The inputs of the procedure remain unchanged and the computation proceeds as follows:

1. Each P_j (including P_ℓ) chooses random $\alpha'_j \in \mathbb{Z}_q$, sends its shares $[r_1]_j, \ldots, [r_n]_j$ and α'_j to P_ℓ, and also broadcasts $com([r_1]_j, \ldots, [r_n]_j, \alpha'_j) = g_1^{[r_1]_j} \cdots g_n^{[r_n]_j} h^{\alpha'_j}$.

2. P_ℓ verifies that $\sum_{j=1}^{k} [r_i]_j = r_i$ (in $\mathbb{F}_p$) for each $i = 1, \ldots, n$ and that the received commitments are consistent with $[r_i]_j$s and α'_js.

3. The parties compute $c'_r = com(r'_1, \ldots, r'_n, \alpha') = \prod_{j=1}^{k} com([r_1]_j, \ldots, [r_n]_j, \alpha'_j)$.

4. Each P_j (including P_ℓ) chooses random $\alpha''_j \in \mathbb{Z}_q$ and broadcasts $com([\gamma(r_1)]_j, \ldots, [\gamma(r_n)]_j, \alpha''_j) = g_1^{[\gamma(r_1)]_j} \cdots g_n^{[\gamma(r_n)]_j} h^{\alpha''_j}$.

5. The parties compute $c'_\gamma = com(\gamma'_1, \ldots, \gamma'_n, \alpha'') = \prod_{j=1}^{k} com([\gamma(r_1)]_j, \ldots, [\gamma(r_n)]_j, \alpha''_j)$.

6. P_ℓ computes $\delta_i = x_i - r_i$ (in $\mathbb{F}_p$) and broadcasts δ_i for $i = 1, \ldots, n$.

7. P_ℓ computes $\alpha' = \sum_{j=1}^{k} \alpha_j$ (in $\mathbb{Z}_q$) and $r'_i = \sum_{j=1}^{k} [r_i]_j$, $s_i = \lfloor (r'_i + \delta_i - x_i)/p \rfloor$ (over integers) for $i = 1, \ldots, n$. It creates commitment $c_s = com(s_1, \ldots, s_n, \tilde{r}) = g_1^{s_1} \cdots g_n^{s_n} h^{\tilde{r}}$ and broadcasts it to the other parties.

8. P_ℓ broadcasts $c_x = com(x_1, \ldots, x_n, \hat{r}), \sigma_1, \ldots, \sigma_n$ and the parties execute $\mathsf{Batch}(pk, x_1, \sigma_1, \ldots, x_n, \sigma_n)$ with P_ℓ playing the role of the prover.

9. The parties additionally execute $\mathrm{PK}\{(x_1, \ldots, x_n, r'_1, \ldots, r'_n, s_1, \ldots, s_n, \alpha', \hat{r}, \tilde{r}) : c_x = g_1^{x_1} \cdots g_n^{x_n} h^{\hat{r}} \wedge c'_r = g_1^{r'_1} \cdots g_n^{r'_n} h^{\alpha'} \wedge c_s = g_1^{s_1} \cdots g_n^{s_n} h^{\tilde{r}} \wedge \bigwedge_{i=1}^{n} \left(r'_i = x_i - \delta_i + s_i p\right)\}$ where P_ℓ plays the role of the prover.

As before, the ZKPK of $x_1, \ldots, x_n, \hat{r}$ is redundant and no longer executed in Batch.

Next, once the computation is complete and the value of α is opened (but prior to reconstructing the output of the computation from the shares), the parties perform additional computations and checks:

1. Each P_j sends α''_j and $[\gamma(r_i)]_j$ for $i = 1, \ldots, n$ to P_ℓ.

2. P_ℓ checks that each $com([\gamma(r_1)]_j, \ldots, [\gamma(r_n)]_j, \alpha''_j)$ is consistent with $[\gamma(r_i)]_j$s and α''_j and aborts otherwise.

3. P_ℓ computes $\alpha' = \sum_{j=1}^{k} \alpha_j$ (in $\mathbb{Z}_q$), $\gamma'_i = \sum_{j=1}^{k} [\gamma(r_i)]_j$, $u_i = \lfloor r'_i/p \rfloor$, $w_i = \lfloor \gamma'_i/p \rfloor$ (over integers) for $i = 1, \ldots, n$. P_ℓ creates commitments $c_u = com(u_1, \ldots, u_n, z) = g_1^{u_1} \cdots g_n^{u_n} h^z$, $c_w = com(w_1, \ldots, w_n, z') = g_1^{w_1} \cdots g_n^{w_n} h^{z'}$ and broadcasts them to other parties.

4. P_ℓ proves the following statement $\mathrm{PK}\{r'_1, \ldots, r'_n, \gamma'_1, \ldots, \gamma'_n, u_1, \ldots, u_n, w_1, \ldots, w_n, \alpha', \alpha'', z, z' : c'_r = g_1^{r'_1} \cdots g_n^{r'_n} h^{\alpha'} \wedge c'_\gamma = g_1^{\gamma'_1} \cdots g_n^{\gamma'_n} h^{\alpha''} \wedge c_u = g_1^{u_1} \cdots g_n^{u_n} h^z \wedge c_w = g_1^{w_1} \cdots g_n^{w_n} h^{z'} \wedge \bigwedge_{i=1}^{n} \big(\gamma'_i = \alpha(r'_i - u_i p + \tau_i) + w_i p\big)\}$, where τ_i was the public value in r_i's MAC.

As described above, we consider the construction that the parties compute the shared secret input $[x_i]$s using the relation $[x_i] = [r_i] + \delta_i$ where $[r_i]$ is shared random value of r_i. The $[r_i]$s are computed in the preprocessing phase so each party holds $[r_i]$ before the protocol starts. Thus we use the both the commitment to r_i and the commitment to x_i in this construction.

As in the previous section, we show that augmenting SPDZ with certified inputs maintains security of the construction in the presence of malicious players and furthermore it is not feasible for a dishonest participant to supply inputs different from what the values that the certification authority signed.

Theorem 3.3.3. *Assuming security of Pedersen commitment,* Batch *is a batch verifier for a signature scheme with privacy, and the proof of knowledge is zero-knowledge, our modification to the SPDZ above is a t-secure multiparty protocol.*

Proof (Sketch). As before, we need to analyze two cases: when P_ℓ is among the corrupted parties and when it is not. We start with the latter.

Case 1: When P_ℓ is not among the corrupt parties, the simulator does not have access to its inputs and uses randomly chosen inputs to simulate the adversarial view (while presenting authentic c_x and σ_is).

Unlike using certified inputs with the Damgård-Nielsen construction, this solution relies on values generated as part of the offline phase. For that reason, the simulator needs to participate in the offline phase as well. Because we make no modifications to the offline phase and because the original SPDZ construction has been previously shown secure, we could call the offline computation as a black box. Also note that offline computation is only used to produce shares of random values and uses no private inputs. Thus, the simulator could simply play the roles of the honest parties (without involvement of the trusted party) and store the shares that they generate. Then because the offline computation opens randomly generated values r_is to P_ℓ, the simulator will store them as well (on behalf of P_ℓ).

Once the online computation starts, the simulator will receive shares from the corrupt parties in step 1, contribute its shares for honest parties stored during the offline computation, and perform the check in step 2 as honest P_ℓ would using the r_is. The simulator proceeds with the computation as prescribed and in step 6 computes the values of δ_i using previously generated r_is and any values of its choice in place of x_is. It consequently uses the same values for x_is in step 7, while step 8 is performed using authentic commitment c_x and signatures $\sigma_1, \ldots, \sigma_n$ which encode true inputs. The simulator finishes the first portion of the online computation by invoking a simulator for the ZKPK in step 9.

Once the main computation on private data completes and the value of α is opened, the simulator participates in the verification steps. It receives values α''_j and $[\gamma(r_i)]_j$ from

each corrupt party in step 1 and retrieves the corresponding values chosen on behalf of honest participants. The simulator performs the same checks as an honest P_ℓ would in step 2. Finally, the simulator can compute all values in step 3 honestly and has enough information to execute the ZKPK in step 4 on behalf of P_ℓ.

The main difference between the real and simulated views is that the simulator has no access to the x_is and uses randomly generated values in place of them in steps 6–7, as well as simulates the ZK proof. We note that this inconsistency cannot be detected by the adversary because it is not feasible to gather information about (true) x_is from the corresponding commitment and signature verification. Similarly, it is not feasible to gather information about r_is from their shares or commitments (to use that information in combination with δ_is). Finally, the ZK proofs reveal no information about their private inputs.

Case 2. Similar to the use of certified inputs with the Damgård-Nielsen construction, simulating the adversarial view is straightforward when P_ℓ is corrupt. In that case the honest parties contribute no input and the simulator simply follows the protocol on behalf of them.

□

It is also not difficult to see that the security guarantees will hold even if we invoke multiple secure function evaluations with the same certification. The same reasoning used in the previous section applies here as well.

Theorem 3.3.4. *If the computation above does not abort, PK is a proof of knowledge,* Batch *is unforgeable, and commitments are binding, a dishonest P_ℓ can enter $x_i' \neq x_i$ for*

at least one $i \in [1, n]$ with at most negligible probability for a sufficiently large $\mathbb{F}_p$..

Proof. Note that in this setting (where all but one party can be corrupt) detectable misbehavior of at least one party leads to computation abort, therefore for the computation to finish, all checks must succeed. Combined with the fact that the signature scheme is unforgeable, we obtain that the commitment c_x has to be on truthful inputs that P_ℓ possesses. Also, based on the security of the original SPDZ construction, the offline generation of the shares of r_is is correct (which in part is due to post-computation checking of the corresponding MACs). What remains to show is that r_is are correctly converted to commitments and r_is are correctly linked to true inputs x_is included in c_x (i.e., it is not feasible to cheat at the time of creating δ_is).

To convert the shares of r_is to commitments, each P_j broadcast a commitment to its own share of both r_is and the MAC on each r_i, which are consequently combined into aggregate commitments c'_r and c'_γ, respectively. Consider what happens when some parties cheat in this process. If some $P_j \neq P_\ell$ are dishonest and provide shares that do not sum to the r_is that P_ℓ expects in step 2, the computation aborts. Note that it is possible for 2 or more dishonest parties to modify their shares from the originally distributed shares in such a way that the sum (over $\mathbb{F}_p$) remains correct, but in that case correctness if not affected. Now suppose that P_ℓ modifies its shares $[r_i]_\ell$ that it uses form its commitment in step 1 so that $\sum_{j=1}^{m}[r_i]_j \neq r_i$ (in $\mathbb{F}_p$). This change, if not detected, would allow P_ℓ to cheat on its inputs by silently modifying the value of r_i. We, however, note that it is not feasible for P_ℓ to make this change without being detected because, in order to succeed, P_ℓ has to consistently modify the corresponding MAC (and commit to it in step 5). Because the value of α is information-theoretically protected from P_ℓ (or

any coalition of parties that includes P_ℓ), it has only $1/|\mathbb{F}_p|$ probability of successfully matching the MAC. This would result in negligible probability for sufficiently large field $\mathbb{F}_p$. Because of the soundness of the ZK proof performed in step 4 of post-computation, P_ℓ must have a correct MAC in order for the protocol to finish.

Once a commitment to the r_is is formed, the correct link between x_is and r_is (i.e., the fact that δ_is were computed correctly) is shown through the ZK proof in step 9 that connects commitments c_x and c'_r. Because of its soundness property, a dishonest P_ℓ is unable to successfully finish the proof if at least one δ_i does not correspond to the difference between x_i and r_i in $\mathbb{F}_p$. This completes the proof.

□

3.3.3 Performance Evaluation

Before we conclude this chapter, we provide a brief performance evaluation of the developed techniques. To summarize, we have implemented the modified ElGamal with private verification that uses a single commitment to n messages (and thus a single signature). Additionally, we have implemented SPDZ-based input of certified inputs into SMC using the same signature. All programs were written in C using OpenSSL's elliptic curve implementation with a 224-bit modulus (equivalent to a 2048-bit modulus in the standard setting) and SHA-256 as the hash function. The experiments were run on an 8-core 2.1GHz machine with a Xeon E5-2620 processor and 64GB of memory running CentOS using a single thread and the times were averaged over at least 20 executions. The results are given in Table 3.2.

The table shows the time of Sign, the cumulative computation of Batch (the prover and

			Number of messages n						
			1	10	10^2	10^3	10^4	10^5	10^6
Modified ElGamal Signatures	Signing		0.69ms	0.70ms	1.0ms	5.2ms	56.1ms	675ms	8.59s
	Verification		1.8ms	2.7ms	12.6ms	111ms	1.11s	12.6s	134s
	Communication		140B	392B	2.84KB	27.4KB	274KB	2.67MB	26.7MB
SPDZ-based entering of certified inputs	Input party	comp.	4.61ms	8.69ms	49.5ms	462ms	4.63s	52.9s	N/A
		comm.	1.02KB	2.81KB	20.7KB	200KB	1.95MB	19.5MB	195MB
	Other party	comp.	5.45ms	8.90ms	43.4ms	393ms	3.92s	45.4s	N/A
		comm.	232B	304B	1.00KB	8.03KB	78.3KB	781KB	7.63MB

Table 3.2: Performance of batch signatures and using certified inputs in SMC.

verifier work), and communication amount in Batch (which is $n + 4$ group elements, with a 28-byte group element in this experiments). Recall that the ZKPK of Batch becomes a part of the ZKPK used during entering certified inputs into SMC and is not executed separately then. For the SPDZ-based solution of section 3.3.2, we used a setup with $k = 3$ computational parties and $|p| = 32$. We report the computation time of input party P_ℓ and all other parties (who do identical work) as well as the amount of communication sent by P_ℓ and other parties, respectively. A broadcast message is counted multiple times using direct transmissions to each party and an EC point is counted as 1 group element.

In our construction, P_ℓ does a slightly larger amount of work per input x_i than other parties, which is reflected in Table 3.2 for large n. When, however, n is small, the constant terms (e.g., batch verification carried out by everyone except P_ℓ) noticeably contribute to the overall time making P_ℓ's time slightly faster. It is important to note that in all cases, each party's work is not substantially higher than the work of private signature verification itself.

An improvement to SPDZ [14] reports for p near 2^{32} in the malicious model (without certified input) 7.5–134 thousand multiplications per second (for 1 to 50 operations in parallel) during the online phase. This is about 7.5–130μs per multiplication (including communication), while our work associated with input certification is about 400μs with

on the order of hundred bytes of communication per message. This is in not drastically higher than that of an online multiplication (all of which can be improved with parallel execution using multiple cores). For many computations, the number of multiplications is significantly greater than the number of inputs, which means that the cost of computation will exceed that of entering and verifying inputs in our solution. Furthermore, offline work per multiplication triple in SPDZ is significantly higher at 28.7ms per triple. All of this suggests that the performance of our solution is demonstrates promise and is not expected to encounter bottleneck in secure computation.

Chapter 4

Oblivious Transfer with Certified Inputs

Secure multi-party computation is a popularly studied field in recent times due to privacy being a hot topic issue today. Present research has continued to prevent attacks for existing security protocols under the two-standard adversarial model accounting for semi-honest and malicious participants. However, we build beyond the standard model to study the problem of correct input enforcement in the secure computation in the presence of malicious participants. Among them, we present the method to enforce the correct evaluator's input in SMC system using GC based OT protocol. To do so we utilize a certification issued by the certified authority (CA) and use it later in secure computation to ensure that the input is not manipulated in the transfer of information.

In this chapter, we treat the issue of enforcing evaluator's correct input to the GC based OT in the presence of malicious participants. Section 4.1 provides solutions to

enforce correct evaluator's input depending on the size of input using Naor-Pinkas' non-interactive efficient OT and Asharov et al.'s efficient OT extension.

4.1 Solutions

We discuss two possibilities: (i) when the evaluator's input size is small, i.e., $m \leq \kappa$ for a computational security parameter κ and (ii) when the evaluator's input is large, i.e., $m > \kappa$. In the first case, regular OT is preferred, while realizing the second option using an OT extension results in a faster solution (since OT extensions invoke κ instance of OT).

Our constructions consist of two protocols: (i) input certification, in which a user obtains certification of its private input from a trusted signer, and (ii) secure two-party computation, where authenticity of the evaluator's private input is to be verified. Because an evaluator enters its input into secure two-party computation by means of OT (or OT extension), for the purposes of current discussion it is sufficient to consider only the OT/OT extension portion of the computation.

In what follows, κ is a symmetric key security parameter, ρ is a statistical security parameter, and $\tau = \kappa + \rho$.

4.1.1 OT-based Input Certification

Our first proposed solution is based on the oblivious transfer of Naor and Pinkas [38], which we provide for reference in Appendix 1, and uses its setup. The common setup to all parties consists of a group $\mathbb{G}$ of prime order q in which the discrete logarithm problem is hard, generated by g, and a random element $C \in \mathbb{G}$. This setup may be chosen by a

Scheme 1:

Common setup: Group $\mathbb{G}$ of prime order q in which the discrete logarithm problem is hard, $\langle g \rangle = \mathbb{G}$, and $C \in \mathbb{G}$.

Input certification: A user U has input $y = y_1 \dots y_m$ to be certified. The signer CA holds a key pair (pk, sk) of a secure signature scheme and the verification key pk is publicly available.

1. For $i \in [1, m]$, CA chooses random $k_i \in \mathbb{Z}_q$ and sets public keys $PK_i^{y_i} = g^{k_i}$ and $PK_i^{\neg y_i} = C / PK_i^{y_i}$.
2. CA concatenates PK_i^0 for $i \in [1, m]$, and stores the resulting string as c.
3. CA signs c as $\sigma(c) = \mathsf{Sign}_{sk}(c)$ and returns $\langle c, \sigma(c), (PK_i^1)_{i=1}^m, (k_i)_{i=1}^m \rangle$ to U.

Oblivious transfer: Evaluator E holds $y = y_1 \dots y_m$ and $\langle c = (PK_i^0)_{i=1}^m, \sigma(c), (PK_i^1)_{i=1}^m, (k_i)_{i=1}^m \rangle$ received from CA. Garbler G holds κ-bit strings (ℓ_i^0, ℓ_i^1) for $i \in [1, m]$.

1. E sends $c, \sigma(c)$ to G. G verifies the signature using CA's public key pk and aborts if verification fails.
2. If G does not abort, it chooses random $r \in \mathbb{Z}_q$ and computes C^r and g^r. G computes $(PK_i^0)^r$ and $(PK_i^1)^r = C^r / (PK_i^0)^r$ for $i \in [1, m]$ and sends g^r and two encryptions $H((PK_i^0)^r, i, 0) \oplus \ell_i^0$, $H((PK_i^1)^r, i, 1) \oplus \ell_i^1$ for each i to E.
3. For $i \in [1, m]$, E computes $H((g^r)^{k_i}, i, y_i) = H((PK_i^{y_i})^r, i, y_i)$ and uses it to recover $\ell_i^{y_i}$.

Figure 4.1: OT with Evaluator's Input Certification.

certification authority CA if it is trusted to properly produce it. Otherwise, the setup may be jointly produced by the interested parties (and in particular C could be decided upon in a distributed manner). Our construction is given as Scheme 1.

We will demonstrate security of Scheme 1 by using the simulation paradigm as specified in definition 2.7.2 and then show that our construction enforces input correctness.

Theorem 4.1.1. *Assuming that the CDH problem is hard and that H is modeled as a random oracle, Scheme 1's oblivious transfer is secure oblivious transfer with auxiliary*

input according to definition 2.7.2.

Proof. Recall that the CDH problem is defined as the inability of a PPT adversary to compute g^{ab}, given (g, g^a, g^b) with random $a, b \in \mathbb{Z}_q$ and the appropriate group setup.

The evaluator's security (G is corrupt). First, we show that no malicious garbler G in the real model, denoted as $\mathcal{A}_G$, can learn information about E's private input. Per definition 2.7.2, the OT protocol can be executed multiple times, where E contributes the same input and corresponds to the same participant, while we denote the (real-world) garbler participating in the jth protocol is execution by $\mathcal{A}_G^j$ for $j = 1, \ldots, k$. In the ideal model, the simulator $\mathcal{S}_G$ has access to each $\mathcal{A}_G^j$ and the trusted party TP and simulates $\mathcal{A}_G^j$'s view. We construct $\mathcal{S}_G$ as follows:

1. Prior to engaging in any simulations, $\mathcal{S}_G$ obtains certification $\langle c, \sigma(c), (PK_i^1)_{i=1}^m, (k_i)_{i=1}^m \rangle$ of a random m-bit input y from the certification authority.

2. For the jth protocol execution ($j = 1, \ldots, k$), $\mathcal{S}_G$ invokes $\mathcal{A}_G^j$ and executes the steps below.

3. $\mathcal{S}_G$ sends $c, \sigma(c)$ to $\mathcal{A}_G^j$ and consequently receives from $\mathcal{A}_G^j$ g^r and encryptions (e_i^0, e_i^1) for $i = 1, \ldots, m$.

4. $\mathcal{S}_G$ extracts r that matches g^r from $\mathcal{A}_G^j$ and sets $\ell_i'^0 = H((PK_i^0)^r, i, 0) \oplus e_i^0$, $\ell_i'^1 = H((PK_i^1)^r, i, 1) \oplus e_i^1$ for $i \in [1, m]$.

5. $\mathcal{S}_G$ sends the pairs $(\ell_i'^0, \ell_i'^1)$ to the TP, which allows honest E to obtain the result from the TP.

6. $\mathcal{S}_G$ outputs whatever $\mathcal{A}_G^j$ outputs.

The only difference between the real and the ideal model executions is the fact that random input y was used instead of E's real input. Each $\mathcal{A}_G^j$ only observes $PK_1^0, \ldots, PK_m^0$ and CA's signature on these values (note that the collective view of all $\mathcal{A}_G^j$s is the same as that of a single $\mathcal{A}_G^j$). Each PK_i^0 is a random element of $\mathbb{Z}_q$, regardless of whether it was computed as a random value g^{k_i} or random value C/g^{k_i}. This provides information-theoretic hiding of the input and thus the real and simulated views have identical distributions.

The garbler's security (E is corrupt). To show security of the protocol in the presence of a malicious evaluator, we need to consider a single $\mathcal{A}_E$ who engages in different OT executions with possibly different garblers. We thus consequently build $\mathcal{S}_E$ with access to $\mathcal{A}_E$ and the TP who needs to simulate a protocol execution view for $\mathcal{A}_E$.

Recall that H is modeled as a random oracle and let $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ for some $\lambda \geq m$. As in the original Naor-Pinkas OT constructions, we add 0 or 1 as a suffix to H's input. This ensures that the result of hashing $(PK_i^0)^r$ is different and independent from the result of hashing $(PK_i^1)^r$, even if PK_i^0 was maliciously chosen to be equal to PK_i^1 by setting it to be $\sqrt{C}$. Similarly, index i is included as a suffix to H's input to ensure that the result of hashing a key for index i_1 is independent from the resulting of hashing a key for index $i_2 \neq i_1$. For general-purpose OT protocols, this was needed to defend against malicious receivers (evaluators in our case), while in Scheme 1 all key pairs (PK_1^0, PK_i^1) are chosen by certification authorities. Thus, if CAs are trusted to compute these keys as prescribed by the construction, the solution can be slightly simplified by removing all suffixes from H's input. Note that a malicious E is unable to replace a CA-generated key by its own because the keys come with a CA's signature. We choose to proceed with

the variant with stronger security guarantees as specified in Scheme 1, which places fewer assumptions on CAs.

The simulator $\mathcal{S}_E$ works as follows:

1. $\mathcal{S}_E$ invokes $\mathcal{A}_E$.

2. When $\mathcal{A}_E$ engages in the jth invocation of OT, $\mathcal{S}_E$ receives the message $\langle c, \sigma(c) \rangle$ that $\mathcal{A}_E$ sends to the garbler. Note that per definition 2.7.2, the auxiliary input (i.e., certification) does not change across different executions of the OT protocol.

3. If $\langle c, \sigma(c) \rangle$ does not pass verification with the CA's public key, $\mathcal{S}_E$ aborts the execution.

4. $\mathcal{S}_E$ chooses random $r \in \mathbb{Z}_q$ and selects random λ-bit strings α_i^0 and α_i^1 for each $i = 1, \ldots, m$. $\mathcal{S}_E$ sends g^r and all (α_i^0, α_i^1) pairs to $\mathcal{A}_E$.

5. $\mathcal{S}_E$ computes $(PK_i^1)^r = C^r/(PK_i^0)^r$ using the common input C and r he chooses and monitors $\mathcal{A}_E$'s queries to H. If a query does not contain $(PK_i^0)^r$ or $(PK_i^1)^r$, $\mathcal{S}_E$ responds at random. Otherwise, if a query contains $((PK_i^b)^r, i, b)$ for some bit $b(= y_i)$, $\mathcal{S}_E$ queries the TP with index i and input bit y_i and stores the received output as $\ell_i^{y_i}$. Next, $\mathcal{S}_E$ sets $H((PK_i^{y_i})^r, i, y_i) = \alpha_i^{y_i} \oplus \ell_i^{y_i}$ and uses this value to answer $\mathcal{A}_E$'s query.

6. $\mathcal{S}_E$ outputs whatever $\mathcal{A}_E$ outputs.

Now, we need to analyze the differences in the real and simulated views. The main difference comes from the fact that the simulator $\mathcal{S}_E$ responds with random strings α_i^0 and α_i^1 in step 2 of the OT instead of the values derived from G's input. However, $\mathcal{A}_E$ is

unable to detect the difference because G's inputs are XORed with the random output of the hash function and thus the result is also random. The only case when $\mathcal{A}_E$ would be able to determine that the protocol is not followed is when it queries both $((PK_i^0)^r, i, 0)$ and $((PK_i^1)^r, i, 1)$, because in that case $\mathcal{S}_E$ would not be able to retrieve both ℓ_i^0 and ℓ_i^1. This event (i.e., the fact that $\mathcal{A}_E$ can compute both $(PK_i^0)^r$ and $(PK_i^1)^r$), however, happens with a negligible probability by the CDH assumption. That is, given g^a and g^b, we could fix $g^a = C$ and $g^b = g^r$; then we use $\mathcal{A}_E$'s ability to compute $(PK_i^0)^r$ and $(PK_i^1)^r$ to answer the challenge as $(PK_i^0)^r \cdot (PK_i^1)^r = C^r = g^{ab}$. This requires the ability to set C, which in our setting is more difficult than in a stand-alone execution of an OT protocol, but still can be accomplished if the CDH problem is given at the setup time.

To conclude, we obtain that the real and simulated views are indistinguishable when either G or E is corrupt, as required.

□

We next proceed with showing that the construction enforces that correct input is entered by E into the computation.

Theorem 4.1.2. *Assuming that the signature scheme used during certification is secure, the CDH problem is hard, and that H is modeled as a random oracle, any dishonest PPT E has at most negligible probability of successfully entering incorrect input in Scheme 1, i.e., learning $\ell_i^{\neg y_i}$ for at least one $i \in [1, m]$*

Proof. In Scheme 1, E computes the label $\ell_i^{y_i}$ for its input y_i using encryptions $w_i^0 = H((PK_i^0)^r, i, 0) \oplus \ell_i^0$ and $w_i^1 = H((PK_i^1)^r, i, 1) \oplus \ell_i^1$ supplied by G. An honest E computes its output as $\ell_i^{y_i} = w_i^{y_i} \oplus H((g^r)^{k_i}, i, y_i)$, where $PK_i^{y_i} = g^{k_i}$ comes from the CA and $PK_i^{\neg y_i}$

was set to $C/PK_i^{y_i}$. The goal of a malicious E is to determine $\ell_i^{\neg y_i}$ using its knowledge of $PK_i^{y_i}$, $PK_i^{\neg y_i}$, and k_i. We show that the probability that malicious E learns $\ell_i^{\neg y_i}$ for at least one $i \in [1, m]$ is negligible in the security parameter κ.

By contradiction, assume that a malicious E has a non-negligible chance in obtaining $\ell_i^{\neg y_i}$ for at least one i. Because we assume security (i.e., unforgeability) of the signature scheme, any malicious E can forge a valid $(c, \sigma(c))$ pair with at most a negligible probability and thus must attack other aspects of the solution. This means that encryptions w_i^0 and w_i^1 are formed using authentic PK_i^0 and PK_i^1 set to C/PK_i^0.

Next, the adversary could recover information about $\ell_i^{\neg y_i}$ by learning some information about $H((PK_i^{\neg y_i})^r, i, \neg y_i)$. However, because H is modeled as a random oracle, its output on all inputs is random and the only way to predict the output of $H((PK_i^{\neg y_i})^r, i, \neg y_i)$ is to query H on that input. This gives us that the input must be recovered with a non-negligible probability in order for the attack to succeed.

Lastly, recovering $(PK_i^{\neg y_i})^r$ is at least as hard as solving the CDH problem. That is, given an instance (g, g^a, g^b) of the CDH problem, set $C = g^a$ and $g^b = g^r$. If the adversary who is given g^r, w_0^i, and w_i^1 is able to recover $(PK_i^{\neg y_i})^r$, we obtain the answer to the CDH problem by setting $g^{ab} = C^r = (PK_i^{\neg y_i})^r (PK_i^{y_i})^r$. Therefore, recovering $(PK_i^{\neg y_i})^r$ with a non-negligible probability contradicts the CDH assumption and we obtain that any malicious PPT E is unable to successfully enter incorrect input, i.e., recover $\ell_i^{\neg y_i}$ with a non-negligible probability.

□

4.1.2 Improved OT extension

Our starting point is the OT extension of Asharov et al. [1]. It provides security against malicious participants and has good performance and is given in Appendix 2. We improve its performance from $O((m+\kappa)\tau^2)$ to $O((m+\kappa)\tau)$ and consequently use it in our input certification construction. The improvement stems from the use of a hash function h that supports homomorphic XOR as in $h(x \oplus y) = h(x) \oplus h(y)$, which is used in the consistency check. The improved OT extension is given as Scheme 2.

The OT extension of Asharov et al. [1] is based on the solution of Ishai et al. [26] who provide an OT extension secure against semi-honest participants. The authors of [1] extend the construction to make it secure against malicious participants by augmenting it with a consistency check. For completeness of this work, we provide the OT extension of [1] in Appendix 2. The consistency check corresponds to steps 7 and 8 of the protocol and ensures that the receiver uses consistent input in forming $\mathbf{u}^i$s during the extension. Our construction (Scheme 2) uses a different consistency check (steps 6 and 7 in Scheme 2), while leaving the remainder of the construction intact.

Because security of the OT extension in [1] in the presence of a malicious receiver is heavily tied to the details of the consistency check, we re-evaluate security of our construction against a malicious receiver. To achieve security in the presence of a malicious sender, it was only required that in the semi-honest construction of Ishai et al. [26] the input $\mathbf{b}$ is padded with a sufficient number of random bits $\mathbf{r}$ (as in step 1 of Scheme 2). In other words, the consistency check does not contribute to security of the construction in the presence of a malicious sender. Therefore, to show security of Scheme 2 against a

Scheme 2:

Input: Sender S holds private binary strings (a_i^0, a_i^1) for $i \in [1, m]$ and receiver R holds m private bits $\mathbf{b} = b_1 \dots b_m$.

Output: R obtains $a_1^{b_1}, \dots, a_m^{b_m}$.

1. R chooses random $\mathbf{r} \in \{0,1\}^\kappa$ and sets $\mathbf{y'} = \mathbf{b}||\mathbf{r}$.
2. S chooses a random string $\mathbf{s} = s_1 \dots s_\tau \in \{0,1\}^\tau$.
3. R chooses a pair of random κ-bit strings (k_i^0, k_i^1) for $i \in [1, \tau]$.
4. S and R perform τ OTs secure against malicious parties where S's input into the ith OT is s_i and R's input is (k_i^0, k_i^1).
5. R computes $\mathbf{u}^i = \mathsf{PRG}(k_i^0) \oplus \mathsf{PRG}(k_i^1) \oplus \mathbf{y'}$ for $i \in [1, \tau]$ and sends them to S. For $i \in [1, \tau]$, let $\mathbf{t}^i = \mathsf{PRG}(k_i^0)$. Also let $T = [\mathbf{t}^1 \| \dots \| \mathbf{t}^\tau]$ denote the $(m + \kappa) \times \tau$ bit matrix where the ith columns is $\mathbf{t}^i$ and ith row is $\mathbf{t}_i$.
6. For $i \in [1, \tau]$, R computes $h_i^0 = h(\mathsf{PRG}(k_i^0))$ and $h_i^1 = h(\mathsf{PRG}(k_i^1))$ and sends them to S.
7. S computes $h_{\mathbf{y'}} = h(\mathbf{u}^1) \oplus h_1^0 \oplus h_1^1$. S checks that $h_i^{s_i} = h(\mathsf{PRG}(k_i^{s_i}))$ for $i \in [1, \tau]$ and $h(\mathbf{u}^i) = h_i^0 \oplus h_i^1 \oplus h_{\mathbf{y'}}$ for $i \in [2, \tau]$. S aborts if at least one of these checks fails.
8. For $i \in [1, \tau]$, S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus \mathsf{PRG}(k_i^{s_i})$. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{y}) \oplus \mathbf{t}^i$.)
9. Let $Q = [\mathbf{q}^1 \| \dots \| \mathbf{q}^\tau]$ denote the $(m + \kappa) \times \tau$ bit matrix where the ith columns is $\mathbf{q}^i$ and ith row is $\mathbf{q}_i$. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{y}) \oplus \mathbf{t}^i$ and $\mathbf{q}_i = (y_i \cdot \mathbf{s}) \oplus \mathbf{t}_i$.)
10. S computes $w_i^0 = a_i^0 \oplus H(i, \mathbf{q}_i)$ and $w_i^1 = a_i^1 \oplus H(i, \mathbf{q}_i \oplus \mathbf{s})$ for $i \in [1, m]$ and sends them to R.
11. R computes $a_i^{b_i} = w_i^{b_i} \oplus H(i, \mathbf{t}_i)$ for $i \in [1, m]$.

Figure 4.2: Improved OT extension.

malicious sender, we only need to analyze the new consistency check and show that the information revealed to S in steps 6 and 7 of Scheme 2 (more precisely, h_i^0 and h_i^1 values) does not allow S to learn information about R's input.

Theorem 4.1.3. *Assuming that* $h : \{0,1\}^{m+\kappa} \rightarrow \{0,1\}^{g(\kappa)}$ *is a universal hash function supporting homomorphic XOR using a suitable choice of function* $g(\cdot)$*, the OT is secure against malicious participants, and* $\mathsf{PRG} : \{0,1\}^{\tau} \rightarrow \{0,1\}^{m+\kappa}$ *is a pseudo-random generator, the release of* $h_i^0 = h(\mathsf{PRG}(k_i^0))$ *and* $h_i^1 = h(\mathsf{PRG}(k_i^1))$ *for* $i \in [1,\tau]$ *in step 6 of Scheme 2 preserves security of the OT extension in the presence of a malicious sender.*

Proof. We show that releasing $h_i^0 = h(\mathsf{PRG}(k_i^0))$, $h_i^1 = h(\mathsf{PRG}(k_i^1))$ for $i \in [1,\tau]$ in steps 6–7 of Scheme 2 reveals no information about R's input to S. Recall that R's private input consists of $\mathbf{b} = b_1 \ldots b_m$ and it also generates k_i^0, k_i^1 for $i \in [1,\tau]$. During the protocol execution, S learns $k_i^{s_i}$, $\mathbf{u}^i$, h_i^0, and h_i^1. This knowledge allows S to consequently compute and learn $h_i^{s_i} = \mathsf{PRG}(k_i^{s_i})$ and $h(\mathbf{y}') = h(\mathbf{u}^i) \oplus h_i^0 \oplus h_i^1$ using the XOR homomorphic property of h (without loss of generality, in our construction consistency of supplied $\mathbf{y}'$ is checked using $\mathbf{u}^1$ with $i = 1$, but S can compute the same value for all other i). Because information about $\mathbf{b}$ is included in the computation only as part of $\mathbf{y}'$, S could learn information about $\mathbf{b}$ (i) by either recovering $\mathsf{PRG}(k_i^{\neg s_i})$ (or, $k_i^{\neg s_i}$ itself) or (ii) by using $h(\mathbf{y}')$.

In the first case, because we assume security of the oblivious transfer used in step 4, malicious S can learn information directly about $k_i^{\neg s_i}$ with a negligible probability. Otherwise, S could attempt to invert $h_i^{\neg s_i}$ to get $\mathsf{PRG}(k_i^{\neg s_i})$ or hash its guesses for $\mathsf{PRG}(k_i^{\neg s_i})$ until the result equals $h_i^{\neg s_i}$. By definition, h is one-way and a polynomial-time adversary

cannot learn any information about preimage of h from its image. Furthermore, guessing pseudo-random string $\mathsf{PRG}(k_i^{\neg s_i})$ of $m + \kappa$ bits long cannot be accomplished by a polynomial-time adversary with a non-negligible probability.

Similarly, retrieving $\mathbf{y}'$ from $h(\mathbf{y}')$ cannot be accomplished by S with a non-negligible probability for the same reasons. In particular, while $\mathbf{b}$ can be guessable by the adversary, $\mathbf{y}' = \mathbf{b}||\mathbf{r}$ has κ random bits and cannot be guessed by S with a non-negligible probability.

□

Theorem 4.1.4. *Assuming that h is a universal hash function with the XOR homomorphic property, $H : \{0,1\}^* \to \{0,1\}^n$ is modeled as a random oracle, the OT is secure against malicious participants, and $\mathsf{PRG} : \{0,1\}^\tau \to \{0,1\}^{m+\kappa}$ is a pseudo-random generator, Scheme 2 is a secure OT extension in the presence of a malicious receiver.*

Proof. Before we proceed with building a simulator, we analyze the cheating options that a malicious receiver can try. Our goal is to ensure that each $\mathbf{u}^i$ is formed correctly and in particular uses the same input $\mathbf{y}'$ with each $\mathbf{u}^i$. This is enfoced using $\mathbf{u}^i$, h_i^0, h_i^1 for each i, as well as $h_{\mathbf{y}'}$ computed using $i = 1$. Let $\mathbf{y}^i = \mathbf{u}^i \oplus \mathsf{PRG}(k_i^0) \oplus \mathsf{PRG}(k_i^1)$ for each $i \in [1, \tau]$. This value is the "input" implicitly defined by $\mathbf{u}^i$ and the base OTs (which may or may not be the same for different i or what R uses as its input to form $\mathbf{u}^i$), which we use in the analysis. Because consistency of inputs is checked against $h_{\mathbf{y}'}$, we say that input $\mathbf{y}^i$ ($i \in [2, \tau]$) is consistent if $h(\mathbf{y}^i) = h_{\mathbf{y}'}$ ($\mathbf{y}^1$ is defined to be consistent with itself). Because malicious R may use different $\mathbf{y}^i$s in the computation, we define $\mathbf{y}'$ to be the most frequently used value. In our consecutive analysis, we will treat the cases when $h(\mathbf{y}') = h_{\mathbf{y}'}$ and $h(\mathbf{y}') \neq h_{\mathbf{y}'}$. In what follows, we use $\hat{\mathbf{y}}'$ to denote the pre-image of $h_{\mathbf{y}'}$

(where $\hat{\mathbf{y}}'$ may or may not be equal to $\mathbf{y}'$).

1. **R supplies information correctly:** If R uses consistent $\mathbf{y}^i$ and correct h_i^0, h_i^1 to supply information to S for $i \in [1, \tau]$, then the verification passes for every s_i for each $i \in [1, \tau]$.

2. **One of the hashes supplied by R is incorrect:** One of the hashes h_i^0 and h_i^1 provided by R is correct (i.e., computed as $h(\mathsf{PRG}(k_i^0))$ or $h(\mathsf{PRG}(k_i^1))$, respectively), while the other one is not and R sets $\mathbf{u}^i$ to match these values and $h_{\mathbf{y}'}$. In other words, recall that S checks whether $h(\mathbf{u}^i) = h_i^0 \oplus h_i^1 \oplus h_{\mathbf{y}'}$. Let $h_i^b = h(v) \neq h(\mathsf{PRG}(k_i^b))$ be the incorrect hash for some $b \in \{0,1\}$, $v \in \{0,1\}^{m+\kappa}$. Then in order to pass the verification, R needs to use $\mathbf{u}^i$ to satisfy $h(\mathbf{u}^i \oplus v \oplus \mathsf{PRG}(k_i^{\neg b})) = h_{\mathbf{y}'}$ or $\mathbf{u}^i \oplus v \oplus \mathsf{PRG}(k_i^{\neg b}) = \hat{\mathbf{y}}'$. This implicitly defines $\mathbf{y}^i$ to be $\hat{\mathbf{y}}' \oplus v \oplus \mathsf{PRG}(k_i^b)$.

   Then if $s_i = b$, the computation is aborted because the value that S computes as $h_i^{s_i} = h(\mathsf{PRG}(k_i^{s_i}))$ does not match the value h_i^b supplied by R. When, however, $s_i = \neg b$, S is unable to detect R's misbehavior and the computation continues. Because honest S chooses each s_i with uniform distribution, a cheating R is caught in this case with probability 1/2 and learns s_i also with probability 1/2.

Other cheating strategies (when, for instance, R uses inconsistent $\mathbf{y}^i$, but does not adjust h_i^0, h_i^1 values to compensate for the difference) result in aborting the computation without giving advantage to R and thus are not beneficial for R to use.

A complicating factor in the analysis of the OT extension in [1] was the fact that R's cheating strategy for some $i \in [1, \tau]$ was not independent of its strategy for another $i' \neq i$. Therefore, computation and adversarial success for different indices i had to be considered

together. In our OT extension, on the other hand, we notice that adversarial success in the computation and verification associated with some $i \in [1, \tau]$ does not depend on its cheating strategy used with another $i' \neq i$. This, in particular, means that the probability that a cheating adversary (who does not use consistent values) will pass verification in rounds i and $i' \neq i$ is at most 1/4 and we detect its misbehavior with probability at least 3/4.

Before we proceed further, we analyze how R's misbehavior is treated in the real model. For the purposes of current discussion, let $h_{\mathbf{y}'} = h(\mathbf{y}')$. Suppose that R cheats during the computation of some $\mathbf{u}^i$ in the attempt to learn information about s_i as described above and does not get detected. If $s_i = 0$ (and consequently b was 1), S sets $\mathbf{q}^i = \mathsf{PRG}(k_0^i)$ as desired and $\mathbf{q}^i = \mathbf{t}^i$. If $s_i = 1$, S sets $\mathbf{q}^i = \mathbf{u}^i \oplus \mathsf{PRG}(k_i^1) = v \oplus \mathbf{y}'$ instead of the desired $\mathsf{PRG}(k_i^0) \oplus \mathbf{y}'$. In both cases, we obtain that $\mathbf{q}^i = (s_i \cdot \mathbf{y}^i) \oplus \mathbf{t}^i$. This is identical to the way $\mathbf{q}^i$s are formed in the presence of a cheating R in [1] and thus, as in [1], we can represent $\mathbf{q}_j$ as $\mathbf{q}_j = (y'_j \cdot \mathbf{s}) \oplus \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j)$, where $*$ denotes the entry-wise multiplication of two vectors and $\mathbf{e}_j$ is some (computable) error vector with the Hamming distance at most ρ from $\mathbf{0}$ (because R cannot cheat on more than ρ values of i without being detected, as discussed later). The case when $h_{\mathbf{y}'} \neq h(\mathbf{y}')$ will be analyzed later.

Now we are ready to proceed with building an ideal-world simulator $\mathcal{S}_R$ for the real-world malicious receiver $\mathcal{A}_R$. The simulator interacts with $\mathcal{A}_R$ and can query TP's response. We construct simulator $\mathcal{S}_R$ as follows:

1. $\mathcal{S}_R$ invokes $\mathcal{A}_R$.

2. $\mathcal{S}_R$ obtains $(k_i^0, k_i^1)_{i=1}^{\tau}$ from $\mathcal{A}_R$ as $\mathcal{A}_R$'s input to the OTs and simulates $\mathcal{A}_R$'s view

using arbitrary input.

3. $\mathcal{S}_R$ receives $\mathbf{u}^i$, h_i^0, and h_i^1 for each $i \in [1, \tau]$ from $\mathcal{A}_R$.

4. $\mathcal{S}_R$ performs verification using $(k_i^0, k_i^1, \mathbf{u}^i, h_i^0, h_i^1)_{i=1}^{\tau}$ and the chosen secret $\mathbf{s}$ the way an honest S would in step 7 in Scheme 2 in the real model. If one of the verification checks fails, $\mathcal{S}_R$ aborts.

5. $\mathcal{S}_R$ computes the matrices T, Q and Y, the ith columns of which are $\mathbf{t}^i = \mathsf{PRG}(k_i^0), \mathbf{q}^i = (s_i \cdot \mathbf{y}^i) \oplus \mathbf{t}^i$ and $\mathbf{y}^i = \mathbf{u}^i \oplus \mathsf{PRG}(k_i^0) \oplus \mathsf{PRG}(k_i^1)$ for $i \in [1, \tau]$.

6. $\mathcal{S}_R$ determines the most frequent value, $\mathbf{y}'$, among $\mathbf{y}^1, \ldots, \mathbf{y}^\tau$ and stores its first m bits as $\mathbf{b}$. It sends $\mathbf{b}$ to the TP and receives $(a_1^{b_1}, \ldots, a_m^{b_m})$.

7. $\mathcal{S}_R$ computes $\mathbf{e}_j = (b_j \cdot \mathbf{1}) \oplus \mathbf{y}_j$ for $j \in [1, m]$, where $\mathbf{y}_j$ is the jth row of Y.

8. $\mathcal{S}_R$ sets $w_j^{b_j} = a_j^{b_j} \oplus H(j, \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j))$ and sets $w_j^{\neg b_j}$ to a string chosen uniformly at random. Then $\mathcal{S}_R$ sends $(w_j^0, w_j^1)_{j=1}^m$ to $\mathcal{A}_R$.

9. $\mathcal{S}_R$ outputs whatever $\mathcal{A}_R$ outputs.

We next analyze the difference in the real and simulated views. The differences include using $\mathbf{y}'$ as $\mathcal{A}_R$'s input (which may not be equal to $\hat{\mathbf{y}}'$ against which inputs are checked) and setting $w_j^{\neg b_j}$ to a random string instead of $w_j^{\neg b_j} = a_j^{\neg bj} \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$.

First, consider the case when $\hat{\mathbf{y}}' \neq \mathbf{y}'$ (e.g., $\mathcal{A}_R$ cheats in the computation of $\mathbf{u}^1$). This means that there are $< \tau/2 - 1$ other inputs consistent with $\hat{\mathbf{y}}'$. This implies that the probability that the computation does not terminate (after S's checks in the real model and $\mathcal{S}_R$'s checks in the ideal model) is $< \frac{1}{2^{\tau/2}}$, which is negligible in both statistical and computational security parameters. We state this as a separate claim:

Claim 4.1.5. The probability that $h'_{\mathbf{y}}$ computed in step 7 of Scheme 2 does not equal to the hash of the most frequent input $\mathbf{y}'$ and this is not detected by S is negligible.

Consequently, because the probability of the event that $\hat{\mathbf{y}}' \neq \mathbf{y}'$ is negligible, the simulator can use $\mathbf{y}'$ in place of $\hat{\mathbf{y}}'$ and obtain indistinguishability of the real and simulated views.

Second, consider the fact that $w_j^{\neg b_j}$ is set to a random string in the simulated view. Note that in the construction the inputs a_j^0 and a_j^1 are masked with $H(j, \mathbf{q}_j)$ and $H(j, \mathbf{q}_j \oplus \mathbf{s})$, respectively. Thus, an adversary who could sufficiently predict $\mathbf{s}$ would be able to tell the simulated view from the real view with a non-negligible probability. It is, however, infeasible for a computationally limited $\mathcal{A}_R$ to have sufficient knowledge of $\mathbf{s}$. First, observe that an adversary who cheats in ρ computations of $\mathbf{u}^i$ has only a negligible chance ($< \frac{1}{2^\rho}$) of not being detected and learning ρ bits of $\mathbf{s}$. We state this as a separate claim.

Claim 4.1.6. A malicious receiver R who learns ρ or more bits of $\mathbf{s}$ is not detected with a negligible probability.

But even that chance is not enough because there are 2^κ options for the remaining κ bits of $\mathbf{s}$, which $\mathcal{A}_R$ will not be able to try. In particular, consider $\mathcal{A}_R$'s ability to make specially-crafted queries to H using the fact that it has control over $\mathbf{y}^i$s that it uses and thus control over vectors $\mathbf{e}_i$s. Recall that in the presence of inconsistent inputs $\mathbf{q}_j$ becomes equal to $(y'_j \cdot \mathbf{s}) \oplus \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j)$. Thus, if the adversary could query H on input $(j, \mathbf{q}_j \oplus \mathbf{s})$ for some j, it would be able to recover $a_j^{\neg b_j}$. This is, however, beyond the abilities of a computationally-limited $\mathcal{A}_R$ who can only query a negligible fraction of the unknown 2^κ strings. We obtain that the values $H(j, \mathbf{q}_j)$ and $H(j, \mathbf{q}_j \oplus \mathbf{s})$ remain unpredictable and

random to A_R, which results in the real and simulated executions being indistinguishable.

□

Say that H doesn't have to be modeled as a random oracle and instead can be strongly κ-min-entropy correlation robust as in [1]. Provide the definition.

4.1.3 OT extension-based Input Certification

Based on the certification provided by the certification authority, we modify the improved OT extension to provide input correctness as the new feature of OT extension. In addition, the certification is reusable for multiple circuits as long as Evaluator's input has not been changed (we call it reusable OT extension with input certification). You can find the details of reusable OT extension with input certification as Scheme 3.

In the following, we prove no information about y is revealed in scheme 3. We also provide the proof of security of scheme 3 based on simulation paradigm in definition 2.7.2 and show that structure enforces input correctness. At the end, we prove the proposed certification in scheme 3 is reusable. To prove that no information leakage takes place during protocol execution, we rely on the following result:

Theorem 4.1.7. *Assuming that h is a universal hash function supporting homomorphic XOR, H is a strongly κ-min-entropy correlation robust function, $\mathsf{PRG} : \{0,1\}^{\tau} \rightarrow \{0,1\}^{m+\kappa}$ is a pseudo-random generator, and the base OT is secure against malicious participants, Scheme 3's oblivious transfer is a secure OT extension with auxiliary input according to definition 2.7.2.*

Proof (Sketch). First, consider evaluator E's security (who plays the role of the receiver)

Scheme 3:

Common setup: Security parameters κ and τ, universal hash function h with XOR homomorphic property.

Input certification. U has input $\mathbf{y} = y_1 \dots y_m$ to be certified. CA holds a key pair (pk, sk) of a secure signature scheme and the verification key pk is publicly available.

1. CA chooses random $\mathbf{r} \in \{0,1\}^\kappa$ and sets $\mathbf{y}' = \mathbf{y}||\mathbf{r}$.
2. CA computes $c = h(\mathbf{y}')$, signs c as $\sigma(c) = \mathsf{Sign}_{\mathsf{sk}}(m)$ and returns $\langle c, \sigma(c), \mathbf{r}\rangle$ to U.

OT Extension: E holds $\mathbf{y} = y_1 \dots y_m$ and the corresponding certification $\langle c, \sigma(c), \mathbf{r}\rangle$. G holds κ-bit strings (ℓ_i^0, ℓ_i^1) for $i \in [1, m]$. E receives $\ell_1^{y_1}, \dots, \ell_m^{y_m}$.

1. E provides $c, \sigma(c)$ to G. G verifies the signature on c using CA's pk and aborts if verification fails.

2–6. E sets $\mathbf{y}' = \mathbf{y}||\mathbf{r}$ and the parties execute steps 2–6 of OT extension in Scheme 2, where G plays the role of S and E plays the role of R.

7. For $i \in [1, \tau]$, G checks that $h_i^{s_i} = h(\mathsf{PRG}(k_i^{s_i}))$ and $h(\mathbf{u}^i) = h_i^0 \oplus h_i^1 \oplus c$. G aborts if at least one of these checks fails.

8-11. The parties execute steps 8–11 of OT extension in Scheme 2, where G plays the role of S, E plays the role of R, $a_i^z = \ell_i^z$ and $b_i = y_i$ for $i = [1, m]$ and $z \in \{0, 1\}$.

Figure 4.3: OT Extension with Evaluator's Input Certification.

in the presence of a malicious garbler G (who plays the role of the sender). When the OT extension in Scheme 3 is executed only once, the sender receives the same information about the receiver's input $\mathbf{y}$ in the form of $h(\mathbf{y}')$, plus it learns h_i^0, h_i^1, $k_i^{s_i}$, and $\mathbf{u}^i$ as in Scheme 2. (Recall that in Scheme 2, in an OT extension execution that does not abort, the observed $h_{\mathbf{y}'} \neq h(\mathbf{y}')$ with a negligible probability and therefore is ignored. See Claim 4.1.5.) Therefore, the modification preserves security of the OT extension in Scheme 2 after a single execution. When the construction is invoked multiple times with the same $h(\mathbf{y}')$, $h(\mathbf{y}')$ is the only value that remains unchanged across the different executions, with all other values chosen anew uniformly at random or produced as a function of new (pseudo)random values and are independent of the input $\mathbf{y}'$. This means that multiple invocations of the OT extension with the same $h(\mathbf{y}')$ does not allow G to learn information about E's input and security of the protocol is preserved.

Next, consider garbler G's security in the presence of a malicious evaluator E. Compared to Scheme 2, this construction limits the way malicious receiver E could misbehave because $h(\mathbf{y}')$ is guaranteed to be authentic. (Recall that in Scheme 2, malicious R could cheat during the computation of $\mathbf{u}^1$ and supply $h_{\mathbf{y}'} \neq h(\mathbf{y}')$.) This is the only difference between the OT extension in Scheme 2 and Scheme 3 and security of a single execution of the OT extension of Scheme 3 follows. When we consider multiple executions of the OT extensions, they are executed independently using new input of G and new randomness used by both parties. The only constant information is $h(\mathbf{y}')$ which gives no advantage to a malicious E.

□

Now, we proceed with showing that the construction using certification enforces E to enter the correct input into the computation.

Theorem 4.1.8. *Assuming that the signature scheme used during certification is secure, H is a strongly κ-min-entropy correlation robust function, the base OTs are secure against malicious participants, any dishonest PPT E has at most negligible probability of successfully entering incorrect input in Scheme 3, i.e., learning $\ell_i^{\neg y_i}$ for at least one $i \in [1, m]$.*

Proof. In scheme 3, E recovers valid label $\ell_i^{y_i}$ for its input bit y_i using the encryption $w_i^{y_i}$ supplied by G and $\mathbf{t}_j$ since the valid label could be computed by the equation $\ell_i^{y_i} = w_i^{y_i} \oplus H(i, \mathbf{t}_i)$. The goal of malicious E is evaluate $\ell_i^{\neg y_i}$ using the information it has (e.g., $\mathbf{y}', \mathbf{t}_i, \mathbf{t}^i, \mathbf{u}^i$). We show that the probability that the malicious E learns $\ell_i^{\neg y_i}$ for at least one $i \in [1, m]$ is negligible. First of all, since we assumed the security of signature scheme and it is computed by CA the signature is unforgeable. Thus malicious E provides valid c and $\sigma(c)$ to G. Next, recall that $\ell_i^{y_i} = w_i^{y_i} \oplus H(j, \mathbf{t}_i \oplus (\mathbf{s} * \mathbf{e}_i))$ for error vector $\mathbf{e}_i$ with hamming distance at most ρ from $\mathbf{0}$. Suppose that the malicious E are trying to find valid input $\mathbf{s}$ of G as part of an effort to recover $\ell_i^{\neg y_i}$. However, as shown in the proof of Theorem 4.1.4 malicious E can not recover correct $\mathbf{t}_i \oplus \mathbf{s}$ because it can recover at most ρ bit of $\mathbf{s}$. Also we assumed that H is universal hash so it can find valid hash value with negligible probability. Therefore, any dishonest PPT E can only succeed in entering incorrect input with negligible probability which concludes the proof.

□

Chapter 5

Conclusions

In this Chapter, we conclude our discussion in this dissertation and describe the direction of future research.

5.1 Signature Scheme with Certified Inputs

Our work showed how to modify CL and ElGamal signature schemes to achieve efficient private batch verification for use in SMC. This process certified inputs and integrated them with two secret-sharing-based protocols. The results demonstrate that the techniques are efficient even in the cases of a large number of inputs, showing that the ideas behind private verification have potential applications to other signature schemes.

Our future research will focus on exploring ways to implement private verification ideas for enforcing input correctness with other existing signature schemes. Furthermore, we plan to study methods to further increase efficiency, so that our research can be applied to various SMC applications.

5.2 Oblivious Transfer with Certified Inputs

In our examination of this topic, we have discussed the security enhancements to the traditional formulation of SMC. This is accomplished by using certificates issued by certified authorities. It guarantees the input correctness and when combined with secure function evaluation, strengthens security. We focused on enforcing correctness of the evaluator's input in secure two-party computation (Garbled Circuit evaluation) in the presence of malicious participants. To summarize, we constructed protocol combining the certificate with the Naor-Pinkas' efficient OT protocol, allowing us to evaluate the function securely while enforcing the evaluator's correct input in case of short input. In case of long input, our construction consisted of a combination using Asharov et al.'s OT extension protocol and certificate. In both cases, we showed how to integrate a certificate with a secure computation to ensure that the evaluator's input is correct, and formally proved the security of our construction.

Current literature is emerging that has explored the possibility of selective failure attacks on OT. A selective failure attack is an attack in which the malicious garbler manipulates the input to know a input bit of the evaluator. This attack makes it possible for an adversarial garbler to know evaluator's input bit depending on whether the protocol is terminated normally or the output indicate failure. In the face of this new challenge, we will study the possibility of selective failure attack on our construction, and finally will carry out further research to devise a construction that resilient to selective failure attacks in the context of evaluator's input certification.

Appendix

1 Naor-Pinkas committing OT

We provide committing Naor-Pinkas OT that is committing version of Naor-Pinkas OT protocol [38] to prevent cheating of sender by providing a commitment on the sender's input r.

Initial Setting: Common input consists of prime p, a generator g of subgroup of $\mathbb{Z}_q^*$ of prime order q, a random element C from the group generated by g, $h = g^u$ for random $u \in \mathbb{Z}_q$ and a hash function H.

Inputs: G has ℓ_0 and ℓ_1, E has $\sigma \in \{0, 1\}$

Output: E learns ℓ_σ and G learns nothing

1. G chooses $r \in \mathbb{Z}_q$ and computes C^r and g^r.
2. E chooses $k \in \mathbb{Z}_q^*$, sets public key $PK_\sigma = g^k$ and $PK_{1-\sigma} = C/PK_\sigma$, and sends PK_0 to G.
3. G computes $(PK_0)^r$ and $(PK_1)^r = C^r/(PK_0)^r$.
4. G sends g^r and two encryptions $H((PK_0)^r, 0) \oplus \ell_0$, $H((PK_1)^r, 1) \oplus \ell_1$ to E.

5. E computes $H((g^r)^k, \sigma) = H((PK_\sigma)^r, \sigma)$ and uses it to recover ℓ_σ. After that E checks $com(r, s) = g^r h^s$ holds. If not E aborts, otherwise E returns ℓ_σ.

2 Asharov et al.'s OT extension [1]

Common setup: Symmetric security parameter κ and statistical security parameter ρ. Assume that $\tau = \kappa + \rho$.

Input: Sender S holds private binary strings (a_i^0, a_i^1) for $i \in [1, m]$ and receiver R holds m private bits $b = b_1 \dots b_m$.

Output: R receives $(a_1^{b_1}, \dots, a_m^{b_m})$ and S learns nothing.

1. S generates wire labels (a_i^0, a_i^1) correspond to R's input for $i = 1, \dots, m$ and chooses a random string $s = s_1 \dots s_\tau \in \{0,1\}^\tau$.

2. R chooses τ pairs of random κ-bit strings (k_i^0, k_i^1) for $i = 1, \dots, \tau$.

3. S and R perform τ OTs secure against malicious parties where S's input into the ith OT is s_i and R's input is (k_i^0, k_i^1) for $i = 1, \dots, \tau$.

4. R chooses random $r \in \{0,1\}^\kappa$ and sets $b' = b||r$.

5. R computes $\mathbf{t}^i = \mathsf{PRG}(k_i^0)$, $\mathbf{u}^i = \mathsf{PRG}(k_i^0) \oplus \mathsf{PRG}(k_i^1) \oplus b'$ and sends $\mathbf{u}^i$ to S for $i = 1, \dots, \tau$

6. Let $T = [\mathbf{t}^1 \| \dots \| \mathbf{t}^\tau]$ denote the $(m + \kappa) \times \tau$ bit matrix where the i-th columns is $\mathbf{t}^i$ and i-th row is $\mathbf{t}_i$ for $i = 1. \dots, \tau$.

7. For every pair $(i,j) \in [1,\tau]^2$, R computes four values:

$$h_{(i,j)}^{(0,0)} = H(\mathsf{PRG}(k_i^0) \oplus \mathsf{PRG}(k_j^0)) \qquad h_{(i,j)}^{(0,1)} = H(\mathsf{PRG}(k_i^0) \oplus \mathsf{PRG}(k_j^1))$$

$$h_{(i,j)}^{(1,0)} = H(\mathsf{PRG}(k_i^1) \oplus \mathsf{PRG}(k_j^0)) \qquad h_{(i,j)}^{(1,1)} = H(\mathsf{PRG}(k_i^1) \oplus \mathsf{PRG}(k_j^1))$$

and sends them to S.

8. For every pair $(i,j) \in [1,\tau]^2$, S checks that $h_{(i,j)}^{(s_i,s_j)} = H(\mathsf{PRG}(k_i^{s_i}) \oplus \mathsf{PRG}(k_j^{s_j}))$, $h_{(i,j)}^{(\overline{s_i},\overline{s_j})} = H(\mathsf{PRG}(k_i^{s_i}) \oplus \mathsf{PRG}(k_j^{s_j}) \oplus \mathbf{u}^i \oplus \mathbf{u}^j)$, and $\mathbf{u}^i \neq \mathbf{u}^j$. S aborts if at least one of these checks fails.

9. For $i = 1, \ldots, \tau$, S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus \mathsf{PRG}(k_i^{s_i})$. (Note that $\mathbf{q}^i = (s_i \cdot b) \oplus \mathbf{t}^i$.)

10. Let $Q = [\mathbf{q}^1 \| \ldots \| \mathbf{q}^\tau]$ denote the $(m+\kappa) \times \tau$ bit matrix where the i-th columns is $\mathbf{q}^i$ and i-th row is $\mathbf{q}_i$. (Note that $\mathbf{q}^i = (s_i \cdot y) \oplus \mathbf{t}^i$ and $\mathbf{q}_i = (y_i \cdot s) \oplus \mathbf{t}_i$.)

11. S sends (w_i^0, w_i^1) for every $i = 1, \ldots, m$, where $w_i^0 = a_i^0 \oplus H(i, \mathbf{q}_i)$ and $w_i^1 = a_i^1 \oplus H(i, \mathbf{q}_i \oplus s)$.

12. For $i = 1, \ldots, m$, R computes $a_i^{b_i} = w_i^{b_i} \oplus H(i, \mathbf{t}_i)$.

Bibliography

[1] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 673–701. Springer, 2015.

[2] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. *Journal of Cryptology*, 30(3):805–858, 2017.

[3] M. Bellare, J. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT*, pages 236–250, 1998.

[4] M. Blanton and F. Bayatbabolghani. Efficient server-aided secure two-party function evaluation with applications to genomic computation. *Proceedings on Privacy Enhancing Technologies (PoPET)*, 4:1–22, 2016.

[5] D. Bogdanov, M. Joemets, S. Siim, and M. Vaht. How the Estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation. In *Financial Cryptography and Data Security*, pages 227–234, 2015.

[6] P. Bogetoft, D. Christensen, I. Damgard, M. Geisler, T. Jakobsen, M. Kroigaard,

J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, and M. Schwartzbach. Secure multiparty computation goes live. In *Financial Cryptography and Data Security*, pages 325–343, 2009.

[7] J. Camenisch, S. Hohenberger, and M. Pedersen. Batch verification of short signatures. In *EUROCRYPT*, pages 246–263, 2007.

[8] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *International Conference on Security in Communication Networks (SCN)*, pages 268–289, 2002.

[9] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, pages 56–72, 2004.

[10] J. Camenisch, D. Sommer, and R. Zimmermann. A general certification framework with applications to privacy-enhancing certificate infrastructures. In *Security and Privacy in Dynamic Environments*, pages 25–37, 2006.

[11] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, Department of Computer Science, ETH Zurich, 1997.

[12] J. Camenisch and G. Zaverucha. Private intersection of certified sets. In *Financial Cryptography and Data Security (FC)*, pages 108–127, 2009.

[13] D. Chaum and T. Pedersen. Wallet databases with observers. In *CRYPTO*, pages 89–105, 1992.

[14] I. Damgård, M. Keller, E. Larraia, V. Pastro, Peter Scholl, and N. Smart. Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits. In *European Symposium on Research in Computer Security (ESORICS)*, pages 1–18, 2013.

[15] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.

[16] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662. 2012.

[17] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security (FC)*, pages 143–159, 2010.

[18] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[19] A. Ferrara, M. Green, S. Hohenberger, and M. Pedersen. Practical short signature batch verification. In *CT-RSA*, pages 309–324, 2009.

[20] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[21] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[22] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game.

In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[23] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.

[24] N. Guo, T. Gao, and J. Wang. Privacy-preserving and efficient attributes proof based on selective aggregate CL-signature scheme. *International Journal of Computer Mathematics*, 93(2):273–288, 2016.

[25] J. Halpern and V. Teague. Rational secret sharing and multiparty computation. In *ACM Symposium on Theory of Computing (STOC)*, pages 623–632, 2004.

[26] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference*, pages 145–161. Springer, 2003.

[27] J. Katz, A. J. Malozemoff, and X. Wang. Efficiently enforcing input validity in secure two-party computation. IACR Cryptology ePrint Archive Report 2016/184, 2016.

[28] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Chapman Hall/CRC, second edition, 2014.

[29] Vladimir Kolesnikov, Ranjit Kumaresan, and Abdullatif Shikfa. Efficient verification of input consistency in server-assisted secure function evaluation. In *International Conference on Cryptology and Network Security*, pages 201–217. Springer, 2012.

[30] B. Kreuter. Secure multiparty computation at Google. Real World Crypto, 2017. Available from https://www.youtube.com/watch?v=ee7oRsDnNNc.

[31] K. Lee, D. H. Lee, and M. Yung. Aggregating CL-signatures revisited: Extended functionality and better efficiency. In *Financial Cryptography and Data Security*, pages 171–188, 2013.

[32] Andrew Y Lindell. Efficient fully-simulatable oblivious transfer. In *Topics in Cryptology–CT-RSA 2008*, pages 52–70. Springer, 2008.

[33] Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.

[34] Yehuda Lindell. How to simulate it–a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017.

[35] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 52–78. Springer, 2007.

[36] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography*, volume 1758, pages 184–199. Springer, 1999.

[37] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In *International Workshop on Public Key Cryptography*, pages 458–473. Springer, 2006.

[38] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 448–457. Society for Industrial and Applied Mathematics, 2001.

[39] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[40] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT*, volume 96, pages 387–398, 1996.

[41] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[42] a. shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In *EUROCRYPT*, pages 386–405, 2011.

[43] Yoav Shoham and Moshe Tennenholtz. Non-cooperative computation: Boolean functions with correctness and exclusivity. *Theoretical Computer Science*, 343(1-2):97–113, 2005.

[44] J. Wallrabenstein and C. Clifton. Equilibrium concepts for rational multiparty computation. In *International Conference on Decision and Game Theory for Security (GameSec)*, pages 226–245, 2013.

[45] David P Woodruff. Revisiting the efficiency of malicious two-party computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 79–96. Springer, 2007.

[46] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[47] Y. Zhang, M. Blanton, and F. Bayatbabolghani. Enforcing input correctness via certification in garbled circuit evaluation. In *European Symposium on Research in Computer Security (ESORICS)*, pages 552–569, 2017.