Improved Signature Schemes for Secure Multi-Party Computation with Certified Inputs

Marina Blanton
Department of Computer Science and Engineering
University at Buffalo (SUNY)
mblanton@buffalo.edu

Myoungin Jeong
Department of Mathematics
University at Buffalo (SUNY)
myoungin@buffalo.edu

Abstract

The motivation for this work comes from the need to strengthen security of secure multi-party protocols with the ability to guarantee that the participants provide their truthful inputs in the computation. This is outside the traditional security models even in the presence of malicious participants, but input manipulation can often lead to privacy and result correctness violations. Thus, in this work we treat the problem of combining secure multi-party computation (SMC) techniques based on secret sharing with signatures to enforce input correctness in the form of certification. We modify two currently available signature schemes to achieve private verification and efficiency of batch verification and show how to integrate them with two prominent SMC protocols.

1 Introduction

Secure multi-party computation (SMC) deals with protecting confidentiality of private data during computation in distributed or outsourced settings. This is a mature research field with a variety of techniques for securely evaluating arbitrary functions by two or more computational parties who are not permitted to have access to the inputs in the clear. Recent rapid advances in this field significantly reduced SMC overhead, and we are witnessing growing deployment of SMC solutions in practice [4, 3, 24].

A standard formulation of SMC defines the problem as evaluating some function f on k (≥ 1) private inputs $\mathsf{in}_1, \ldots, \mathsf{in}_k$ from different sources by m (≥ 2) computational parties and producing s (≥ 1) outputs which get revealed to the designated parties. The security objective is that no information about the private data is revealed to any party beyond the agreed upon output (or no information at all if a party receives no output). Standard security definitions model the participants as semi-honest (who correctly follow the prescribed computation, but might analyze the messages they receive in the attempt to learn unauthorized information) or malicious (who can arbitrary deviate from the computation in the attempt to learn unauthorized information). Output correctness guarantees must also hold in these respective models. These definitions, however, provide no guarantees with respect to what inputs are entered into the computation. That is, a malicious participant can modify its real input in the attempt 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 he can compensate for the error and learn the correct result. Or, alternatively, the participant can modify his input in such a way as to learn the maximum amount of information about private data of others', beyond what would be available if the computation was run on truthful inputs ([2] gives an example of this 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 enforcement of correct (i.e., truthful) inputs being used in SMC via input certification. That is, at the time of computation initiation, a party supplying input accompanies it by 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 is generated or can be verified by an authority (such as the government, a medical facility, etc.) who can issue private certification to the user at that time.

The problem of enforcing input correctness via certification has been studied for specific SMC applications (e.g., anonymous credentials [8] or set operations [10, 15]) and, more recently, for general functions [2, 23, 32]. However, all of the efforts we are aware of for the general case have been for secure two-party computation based on garbled circuits (GCs). It is an interesting problem to study because GC evaluation does not naturally combine with signature or certification techniques, but we also believe that this problem deserves attention beyond GCs. For this reason, in this work we 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. We note that, unlike many other conventional uses of signatures, this problem setting requires that signature verification is 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 (consider, e.g., genomic data). Because signatures are built using rather expensive public-key techniques, which in the privacy-preserving setting often need to be combined with zero-knowledge proofs, we are interested in improving signature verification time using batch verification of multiple signatures.

In this work we study two types of signatures in the context of this problem: (i) CL-signatures [6, 7] 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 [17]. After formulating the necessary security guarantees of private signature verification, we show that signature showing in [7] can be simplified to meet our definition of message privacy and construct a batch verifier for the resulting signature. In the case of ElGamal signatures, we first modify a provably secure ElGamal signature scheme from [28] to achieve private verification and consequently construct a batch verifier for the resulting algorithm. Our batch verifiers use the small exponents technique [1] to randomize multiple signatures to ensure that batch verification can succeed only when all individual signatures are valid.

Another component of this work deals with combining the developed signature schemes with SMC techniques secure in the malicious model. Toward this goal, we identify two prominent constructions of SMC based on secret sharing: (i) Damgård-Nielsen solution [13] of low communication complexity where the number of corrupted parties is below $m/3$ and (ii) SPDZ [14] with a very fast online phase and tolerating any number of corruptions. We show how to modify their input phase to use our signatures with an additional optimization of utilizing a single commitment to multiple signatures instead of using individual commitments.

Finally, we implement our ElGamal-based signature scheme and SPDZ-based use of certified inputs for a varying number of messages (SMC inputs) and show that they result in efficient performance. The techniques are general enough to be applicable to other signature algorithms (such as ElGamal-based DSA and others).

2 Related Work

The only publications on certified inputs for SMC that we are aware of were mentioned above (i.e., techniques for specific applications [8, 10, 15] and techniques for GCs [2, 23, 32]). There is also work on using game theory to incentivize rational players to enter their inputs truthfully (see, e.g., [21, 30] among others), but such techniques are complementary to this work and can be used for inputs which cannot be feasibly certified.

The first systematic treatment of batch signature verification appeared in [1], although interest in batch verification of signatures and other cryptographic operations in general goes further back. More recent techniques for batch verification include [5, 18] among others, although none of them target private verification (defined in section 3.1) which is central to this work.

A related concept is that of aggregate signatures. It allows a number of different signatures to be aggregated into a single short signature to save bandwidth in resource-constrained environments. Aggregate signature schemes were developed for CL signatures on which we build in this work [25]. There are, however, two central differences from our work: (i) aggregate signatures have strictly weaker security guarantees than batch verification [5] 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. Guo et al. [20] use privacy features of CL signatures and construct an aggregate CL signature, but the difference in the security guarantees still stands.

Privacy-preserving signature schemes have been studied in other contexts. Examples include anonymous signatures [31], confidential signatures [16], and pseudorandom signatures [19]. There is a connection between these concepts (especially, confidential signatures) and our notion of signatures with privacy, but these prior concepts provide message confidentiality guarantees only for high-entropy message spaces.

3 Preliminaries

3.1 Definitions

We next describe notation and definitions used in the rest of the paper. A function $\epsilon : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is *negligible* (denoted, negl) if for every positive polynomial $p(\cdot)$ there exists an integer N such that for all $\kappa > N$ $\epsilon(\kappa) < \frac{1}{p(\kappa)}$. The notation $G = \langle g \rangle$ means that g generates group G. We rely on groups with pairings, which we review next.

Definition 1 (Bilinear map) *A one-way function $e : G \times G \to \mathbf{G}$ is a bilinear map if it is:*

- *(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 a security parameter 1^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, $(q, G, \mathbf{G}, g, e) \leftarrow \mathsf{Setup}(1^\kappa)$.

Definition 2 (Signature scheme) *A* signature scheme *consists of 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, which we provide in Appendix A.

Batch verification [1] is a method for verifying a set of signatures on different messages signed by the same or different signers, which is intended to be more efficient than verifying them independently. In this work, we are primarily interested in batch verification of signatures produced by the same signer. Batch verification is defined as:

Definition 3 (Batch verification of signatures [5]) *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 $2^{-\kappa}$.*

In our constructions, we rely on *zero-knowledge proofs of knowledge* (ZKPKs) and *commitments*. A ZKPK is a two-party 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. Informally, a ZKPK should satisfy the following properties: (i) *completeness*: if the statement is true, then an honest verifier will be convinced of the statement's validity after interacting with an honest prover; (ii) *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); and (iii) *zero-knowledge*: if the statement is true, then no cheating verifier can learn anything other than the fact that the statement is true. We are interested in simple statements over discrete logarithms such as those described, e.g., in [11, 9].

A *commitment scheme* 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, and we omit their formal specification here. We only note that Commit is a randomized algorithm and for that reason we use notation $com(m, r)$ to denote a commitment to m using randomness r.

As mentioned before, in this work 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. In the rest of this subsection, we provide the necessary definitions for signature schemes with privacy. We start by re-defining the traditional formulation of a signature scheme as follows:

Definition 4 (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 possibly interactive algorithm, in which the prover and the verifier hold a public key pk, the prover has access to m and (σ, x_σ) output by* Sign*, supplies a message encoding x_m and a (possibly modified) signature $\tilde{\sigma}$ to the verifier, and the verifier outputs a bit.*

With this definition, we allow 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.

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 in this work x_m always takes the form of a commitment to m, $com(m, r)$, we explicitly incorporate this in our security definition.

The security (unforgeability) experiment of a signature with privacy is similar to the conventional definition (see ForgeSig in Appendix A) 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 $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ 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 PrivVerify returns 1 and all other checks succeed; otherwise, output 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 5 (Private verification) *A 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 [6, 7] 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 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 fully protect the signed data itself. Unlinkability of signature showings is generally not needed in our application, where 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). Thus, we only protect information about the signed values and this difference allows for faster signature verification while still 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.

3.2 Signature and Commitment Schemes

In this work, we build on Camenisch-Lysyanskaya signature Scheme A from [7] (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 [26], while demonstrating the zero-knowledge property of proving possession of a signature uses no additional assumptions other than hardness of discrete logarithm.

We also build on ElGamal signature scheme [17]. Because the original construction allows for existential forgeries, we use its provably secure variant by Pointcheval and Stern [29, 28]. 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 for 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 and is shown 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.

Lastly, we also utilize well-known Pedersen commitment scheme [27] based on discrete logarithm. The setup consists of a group G of prime order q and two generators g and h. To commit to message $m \in \mathbb{Z}_q$, we choose 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.

4 Constructions based on CL Signatures

In this section, we discuss constructions based on CL signatures. We start by demonstrating that CL signatures with protocols satisfy our notion of signatures with 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. The next section treats ElGamal signatures.

4.1 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 our 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 proof of knowledge and the signature was shown to be unforgeable, this scheme satisfied signature unforgeability. We also achieve the private verification property:

Theorem 1 *CL Scheme A above is a signature scheme with privacy.*

Proof The original proof of signature protocol in [7] was shown to be zero-knowledge, meaning that no information about the original signature σ or the corresponding message m is revealed. We only modify 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. When PrivVerify is executed multiple times using the same original signature σ, the verifier learn 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). 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 our comparisons.

4.2 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 2 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 2 *Modified CL Scheme A above is a signature scheme with privacy.*

Proof Let $\mathcal{A}$ be a PPT adversary attacking our 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'_is) and different commitments to m_b (using r_js for randomness).

Before we proceed with further analysis, note that the ZKPK in $\mathsf{PrivVerify}$ 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 do not contribute additional information (and use unrelated randomness) and thus do not help in answering the challenge. It thus remains 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. In more detail, 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 call the resulting scheme Π'. Clearly, we have that $\mathbf{Pr}[\mathsf{MesInd}_{\mathcal{A},\Pi'}(\kappa) = 1] = \frac{1}{2}$. Also, 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.

4.3 Batch Verification of Modified CL Scheme A

The next step is to design batch verification for verifying n signatures. Because for our 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 [1] that instructs the verifier to choose security parameter 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 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

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

Because $\hat{\mathbf{v}}_x^{-1} = e(X, \prod_{i=1}^{n} a_i^{\delta_i})^{-1} = e(g^x, \prod_{i=1}^{n} g^{s_i\delta_i})^{-1} = \prod_{i=1}^{n} e(g,g)^{-xs_i\delta_i}$, $e(g,g)^{-x\sum_i s_i\delta_i} = e(g,g)^{\sum_i \delta_i(xt_i m_i + \gamma_i r_i)}$ and consequently $\sum_i xs_i\delta_i + \sum_i \delta_i(xt_i m_i + \gamma_i r_i) \equiv 0 \pmod q$. Let us set $\beta_i = x(s_i + t_i m_i) + \gamma_i r_i$, then

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

Now suppose that Batch returned 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 (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

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

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 (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 can 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 $\mathrm{T} = \prod_{i=1}^{n} \left(\mathbf{v}_{xy_i}^{v_i} \mathbf{v}_{s_i}^{v_i'}\right)$, 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 $\prod_{i=1}^{n} \left(\mathbf{v}_{xy_i}^{u_i} \mathbf{v}_{s_i}^{u_i'}\right) = \mathrm{T}\mathbf{v}_x^{-e}$.

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 and significantly reduces the number of pairing operations, which we consider to be the costliest operation, compared to private verification of individual messages.

The way inputs are entered in the SMC constructions considered in Section 6, 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.

5 Construction based on ElGamal Signature

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

5.1 Modified ElGamal Scheme

Our starting point was provably secure ElGamal [28] described in section 3.2. 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 [1]. Our 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 6) with $\mathsf{ForgePrivSig}$ experiment that accommodates privacy as described in section 3.1. The intuition is that the prover now has a signature on a commitment, but has to demonstrate the knowledge of the commitment opening, i.e., the message itself, and the use of groups of prime order only simplifies the analysis in [28]. We state unforgeability and privacy properties next.

Theorem 4 *Modified ElGamal signature scheme is existentially unforgeable against an adaptive chosen-message attack in a random oracle model.*

Proof (Sketch) The ElGamal signature scheme on which we build [28, 29] 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 our modification uses.

The proof in [29] 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 our 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 [29] can also be simplified because we only need to consider computation modulo q in the exponent as opposed to computation modulo $qR = p - 1$. Thus, it is only easier to show security when the setup assumes groups of prime order.

As far as our 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 using 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. □

Theorem 5 *Modified ElGamal scheme is a signature scheme with privacy.*

Proof Let $\mathcal{A}$ be a PPT adversary attacking our 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 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 it and new randomness is used for each new signature. □

The ZKPK in this PrivVerify proceeds similar to prior 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 8 mod exp. (If the user does not store commitment x_m, its re-computation is another 1 regular and 1 short mod exp.)

5.2 Batch Verification of Modified ElGamal Signatures

Our 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 6 Batch *above is a batch verifier for the modified ElGamal scheme.*

Proof First we show that $\mathsf{PrivVerify}(pk, m_1, \sigma_1) = \ldots = \mathsf{PrivVerify}(pk,\ m_n, \sigma_n) = 1$ implies that $\mathsf{Batch}\,(pk, (m_1, \sigma_1), \ldots, (m_n, \sigma_n)) = 1$. 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

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

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

$$\sum_{i=1}^{n} H(x_{m_i}||t_i)\delta_i - \sum_{i=1}^{n} x_i t_i \delta_i - \sum_{i=1}^{n} k_i s_i \delta_i \equiv 0 \pmod{q} \tag{3}$$

Let $\beta_i = H(x_{m_i}||t_i) - x_i t_i - k_i s_i$. Then equation (3) can be written as $\sum_{i=1}^{n} \delta_i \beta_i \equiv 0 \pmod{q}$, which is the same as equation (1) in the proof of Theorem 3. Thus, the remainder of the proof proceeds in the same way as the proof of Theorem 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.)

Now recall that the way messages are input into SMC allows us to use a single commitment to all n messages. For our modified ElGamal this optimization results in great savings because this means that we can use only 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 because only 1 signature and 1 commitment are communicated in step 1, step 2 only involves the proof of knowledge of the discrete logarithm representation of x_m, and 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 reduced to $2n + 6$ mod exp (if the commitment is to be re-computed, we add 1 regular and n short mod exp).

Table 1 summarizes performance of our constructions.

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 1: Performance of private verification for a single signature and a batch of size n. It is assumed that commitments are stored pre-computed.

6 Using Certified Inputs in Secure Computation

Having described our 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 [13] and SPDZ [14] and discuss them consequently. These were chosen based on their attractive performance and distinct security guarantees that they provide: when the computation is performed by m parties, the former solution tolerates fewer than $m/3$ corruptions, while the latter can handle any number of corrupt parties. Our solution uses signatures with privacy to guarantee that inputs entered into secure computation are identical to those generated or observed by an authority, but in general certification could take 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, which we provide in Appendix A. We additionally require 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 $P_1, \ldots, P_m$ and assume that they are connected by pairwise secure channels. These constructions use (m, t)-threshold linear secret sharing, and we denote a secret shared version of x by $[x]$.

6.1 Damgård-Nielsen Scalable and Unconditionally Secure Multiparty Computation

The construction of Damgård-Nielsen [13] is unconditionally secure (assuming secure channels) in the presence of at most $t < m/3$ malicious participants. It was the first to achieve unconditional security with communication complexity where the part that depends on the circuit size is only linear in m. The computation proceeds in two stages: offline pre-computation that generates random multiplication triples and 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 pre-computation 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 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. Then because 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 the 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 [13] 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 the version with a single commitment to all x_is which allows for improved performance. The case of a single certified input x will follow from that construction. Also, 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.

In what follows, we use notation $[y]_j$ to denote the jth share of y held by party P_j. As in [13], 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 is then as follows:

Input: The parties collectively hold the public key pk of the certification authority. P_ℓ has private input $x_1, \ldots, x_n$, $c_x = com(x_1, \ldots, x_n, \hat{r})$, 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 [13]). If P_ℓ is honest, there are at least $2t + 1$ parties who hold consistent shares of each x_i and we denote this set by S. (Otherwise, the protocol fails and the parties restart it as specified in [13].)

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.

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

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 $x'_i = \sum_{P_j \in S} \beta_j [x_i]_j$, $s_i = \lfloor x'_i / p \rfloor$ (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 for 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).

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 multiparty computation and it is not feasible for a dishonest participant to supply inputs different from what was signed by a certification authority.

Theorem 7 *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 $t < m/3$.*

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 8 *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 $\mathbb{F}_p$). □

6.2 SPDZ

The second solution that we study is built on SPDZ [14]. This is an SMC protocol that achieves security in the presence of any number of malicious parties $t < m$ (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 [13] did. That is, 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., $(m-1)$-out-of-m) is used instead of threshold secret sharing and each secret-shared value y also uses 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. In other words, 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 $[\gamma(y)]_1 + \ldots + [\gamma(y)]_m = \alpha(y+\tau)$. 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 [14] for detail).

Unlike the solution considered in section 6.1, 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 deferred to the end of the computation where the value of α is opened. Then 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 $\sum_{j=1}^m [r_i]_j = r_i$ (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 $c'_r = com(r'_1, \ldots, r'_n, \alpha') = \prod_{j=1}^m com([r_1]_j, \ldots, [r_n]_j, \alpha'_j)$.
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 $c'_\gamma = com(\gamma'_1, \ldots, \gamma'_n, \alpha'') = \prod_{j=1}^m com([\gamma(r_1)]_j, \ldots, [\gamma(r_n)]_j, \alpha''_j)$.
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 $\alpha' = \sum_{j=1}^m \alpha'_j$ (in $\mathbb{Z}_q$) and $r'_i = \sum_{j=1}^m [r_i]_j$, $s_i = \lfloor (r'_i + \delta_i - x_i)/p \rfloor$ (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.

Then 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 computation 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 $\alpha' = \sum_{j=1}^{m} \alpha'_j$ (in $\mathbb{Z}_q$), $\gamma'_i = \sum_{j=1}^{m} [\gamma(r_i)]_j$, $u_i = \lfloor r'_i/p \rfloor$, $w_i = \lfloor \gamma'_i/p \rfloor$ (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 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 9 *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 10 *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.

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

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. □

7 Performance Evaluation

Before we conclude, we provide a brief performance evaluation of the developed techniques. 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 2.

The table shows the time of Sign, cumulative computation of Batch (the prover and verifier work), and communication amount in Batch (which is $n + 4$ group elements, with a 28-byte group element in our 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 6.2, we used a setup with $m = 3$ computational parties and $|p| = 32$. We report 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 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. But in all cases, each party's work is not substantially higher than the work of private signature verification itself.

An improvement to SPDZ [12] 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, which 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 quite good and is not expected to be the bottleneck in secure computation.

8 Conclusions

In this work, we showed how to modify CL and ElGamal signature schemes to achieve efficient private batch verification for use in SMC with certified inputs and integrate them with two secret-sharing-based protocols. Our results demonstrate that the techniques are efficient even for a large number of inputs and the ideas behind private verification are rather general to have a potential application to other signature schemes.

Acknowledgments

We thank anonymous reviewers for their valuable feedback. This work was supported in part by grant 1319090 from the National Science Foundation (NSF). Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of NSF.

References

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

[2] M. Blanton and F. Bayatbabolghani. Efficient server-aided secure two-party function evaluation with applications to genomic computation. *PoPET*, 4:1–22, 2016.

[3] 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.

[4] 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.

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

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

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

[8] 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.

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

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

[11] D. Chaum and T. Pedersen. Wallet databases with observers. In *CRYPTO*, 1992.

[12] 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 *ESORICS*, pages 1–18, 2013.

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

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

[15] 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.

[16] A. Dent, M. Fischlin, M. Manulis, M. Stam, and D. Schröder. Confidential signatures and deterministic signcryption. In *Public Key Cryptography (PKC)*, pages 462–479, 2010.

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

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

[19] N. Fleischhacker, F. Günther, F. Kiefer, M. Manulis, and B. Poettering. Pseudorandom signatures. In *ASIACCS*, pages 107–118, 2013.

[20] 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.

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

[22] J. Katz and Y. Lindell. *Introduction to modern cryptography.* Chapman Hall/CRC, second edition, 2014.

[23] 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.

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

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

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

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

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

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

[30] J. Wallrabenstein and C. Clifton. Equilibrium concepts for rational multiparty computation. In *GameSec*, pages 226–245, 2013.

[31] G. Yang, D. Wong, X. Deng, and H. Wang. Anonymous signature schemes. In *Public-Key Cryptography (PKC)*, pages 347–363, 2006.

[32] Y. Zhang, M. Blanton, and F. Bayatbabolghani. Enforcing input correctness via certification in garbled circuit evaluation. In *ESORICS*, pages 552–569, 2017.

A Additional Background

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 $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ 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.

Definition 6 (Security of a signature scheme [22]) *A signature scheme* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is* existentially unforgeable under an adaptive chosen-message attack *if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* negl *such that* $\Pr[\mathsf{ForgeSig}_{\mathcal{A},\Pi}(\kappa) = 1] \leq \mathsf{negl}(\kappa)$.

Security of a protocol in the malicious model is shown according to the ideal/real simulation paradigm. In the ideal execution of the protocol, there is a *trusted third party* (TTP) that evaluates the function on participants' inputs. The goal is to build a simulator S who can interact with the TTP and the malicious party and construct a protocol's view for the malicious party. A protocol is secure in the malicious model if the view of the malicious participants in the ideal world is computationally indistinguishable from their view in the real world where there is no TTP. Also the honest parties in both worlds receive the desired output. This gives us the following definition of security in the malicious model.

Definition 7 *Let parties $P_1, \ldots, P_m$ engage in a protocol Π that computes a (possibly probabilistic) n-ary function $f : (\{0,1\}^*)^m \to (\{0,1\}^*)^m$, with party P_i contributing input in_i. Let $\mathcal{A}$ be an arbitrary algorithm with auxiliary input x, S be an adversary/simulator in the ideal model and I be a set of indices of the corrupted parties. Let $REAL_{\Pi,\mathcal{A}(x),I}(\mathsf{in}_1, \ldots, \mathsf{in}_m)$ denote the view of adversary $\mathcal{A}$ controlling parties in I together with the honest parties' outputs after real protocol Π execution. Similarly, let $IDEAL_{f,S(x),I}(\mathsf{in}_1, \ldots, \mathsf{in}_m)$ denote the view of S and outputs of honest parties after ideal execution of function f. We say that Π t-securely computes f if for each coalition I of size at most t, every probabilistic adversary $\mathcal{A}$ in the real model, all $\mathsf{in}_i \in \{0,1\}^*$ and $x \in \{0,1\}^*$, there is probabilistic S in the ideal model that runs in time polynomial in $\mathcal{A}$'s runtime and $\{\mathrm{IDEAL}_{f,S(x),I}(\mathsf{in}_1, \ldots, \mathsf{in}_m)\} \equiv \{\mathrm{REAL}_{\Pi,\mathcal{A}(x),I}(\mathsf{in}_1, \ldots, \mathsf{in}_m)\}$.*